



COM interface

(AxisVM Library **15.3**)

Rev. 1
Reference Guide

Updated: 2019.12.05

CONTENTS

CONTENTS	3
Introduction	8
Registration.....	9
Starting the COM server	11
Units, Data types and general errors	12
Compatibility	14
Indexing	14
Case sensitivity and Python.....	14
IAxisVMAApplication	15
Properties.....	17
IAxisVMCatalog.....	19
IAxisVMCrossSectionEditor	22
IAxisVMMODELS	25
IAxisVMMODEL.....	26
IAxisVMACTUALREINFORCEMENT	43
IAxisVMAATTACHMENTS.....	47
Properties	47
IAxisVMAATTRIBUTES.....	48
Properties	51
IAxisVMCALCULATION	52
IAxisVMCOLUMNREBARS.....	59
IAxisVMCRITICALGROUPCOMBINATIONS.....	63
IAxisVMCROSSSECTIONS	64
IAxisVMCROSSSECTION.....	86
IAxisVMCROSSSECTIONOPTIMIZATION	90
IAxisVMCUSTOMPARTS	98
IAxisVMCUSTOMPARTFOLDER	101
IAxisVMDIAPHRAGM	104
IAxisVMDIMENSIONS	106
IAxisVMDOMAINS.....	110
IAxisVMDOMAIN.....	121
IAxisVMDOMAINSUPPORTS	131
IAxisVMDOMAIN SUPPORT	132
IAxisVMDRAWINGSLIBRARY	133
IAxisVMDYNAMICLOADFUNCTIONS	134
IAxisVMEEDGECONNECTIONS	137
IAxisVMEENVELOPES	139
IAxisVMIINCREMENTFUNCTIONS	142
IAxisVMLAYERS	144
IAxisVMLINES	148
IAxisVMLINE	157
IAxisVMLINESUPPORTS	169
IAxisVMLINE SUPPORT	173
IAxisVMLINKELEMENTS	176
Functions	177
Properties	179
IAxisVMLOADCASES.....	180
IAxisVMLOADCOMBINATIONS.....	189
IAxisVMLOADGROUPS	193
IAxisVMLOADGROUP	195
IAxisVMLOADPANELS.....	196
IAxisVMLOADPANEL	198
IAxisVMLoads	200
IAxisVMLOGICALPARTS	227
IAxisVMMATERIALS	232
IAxisVMMATERIAL	239
IAxisVMMATHTEXTS	242
IAxisVMMEMBERS	244
IAxisVMMEMBER	250
IAxisVMMOVINGLOADS	259
IAxisVNODAL SUPPORTS	261

IAxisVMNodalSupport.....	266
IAxisVMNodesSupports	269
IAxisVMMembersSupports	280
IAxisVMDomainsSupports.....	285
IAxisVMNodes	287
IAxisVMPushoverHingeFunctions	293
IAxisVMRCBeamDesign	295
IAxisVMRCColumnChecking.....	306
IAxisVMReferences	312
IAxisVMReports	315
IAxisVMResults	317
Overview of AxisVM result objects	328
IAxisVMAcceleration	330
IAxisVMCalculatedReinforcement	332
IAxisVMCrackWidth	340
IAxisVMDisplacements	347
IAxisVMForces.....	367
IAxisVMReinforcementCheck	420
IAxisVMShearCapacity	425
IAxisVMStresses	432
IAxisVMSteelDesignResults	459
IAxisVMTimberDesignResults	472
IAxisVMVelocity	482
IAxisVMRebarSteelGrades	484
IAxisVMRigidBodies	487
IAxisVMSections.....	489
IAxisVMSettings	523
IAxisVMSeismicStoreys	531
IAxisVMSpectrum	532
IAxisVMSpringParams	536
IAxisVMSpringParam.....	538
IAxisVMSteelDesignMembers	539
IAxisVMStoreys	546
IAxisVMStructuralGrids	549
IAxisVMStructuralGrid	552
IAxisVMSurfaces	555
IAxisVMSurface	559
IAxisVMSurfaceSupports	563
IAxisVMSurfaceSupport.....	565
IAxisVMTask.....	566
IAxisVMTimberDesignMembers	567
IAxisVMTimeIncrementFunctions	570
IAxisVMVirtualBeams	573
IAxisVMWindows	579
IAxisVMWindow	601
IAxisVMWorkplanes	607
IAxisVXMLAmpans	610
IAxisVMOBJECTCreator	611
IAxisVMLine2d	611
IAxisVMLines3d	613
IAxisVMMovingLoadOnBeam	614
IAxisVMMovingLoadOnDomain	616
IAxisVMPolygon2d	618
IAxisVMPolygon2dList	618
Event handling.....	619
IAxisVMAccelerationEvents	619
IAxisVMActualReinforcementEvents.....	619
IAxisVMAplicationEvents	619
IAxisVMAttachmentsEvents	620
IAxisVMAtributesEvents	620
IAxisVMColumnRebarsEvents	620
IAxisVMCalculatedReinforcementEvents.....	620
IAxisVMCalculationEvents	620

IAxisVMCrackWidthEvents	621
IAxisVMCriticalGroupCombinationsEvents	621
IAxisVMCrossSectionsEvents	621
IAxisVMCrossSectionEditorEvents	621
IAxisVMCustomPartsEvents	621
IAxisVMCustomPartFolderEvents	621
IAxisVMDisplacementsEvents	621
IAxisVMDiaphragmEvents	622
IAxisVMDimensionsEvents	622
IAxisVMDomainsEvents	622
IAxisVMDomainSupportsEvents	622
IAxisVMDrawingsLibraryEvents	622
IAxisVMDynamicLoadFunctionsEvents	622
IAxisVMEdgeConnectionsEvents	622
IAxisVMEvelopesEvents	622
IAxisVMForcesEvents	622
IAxisVMIcrementFunctionsEvents	622
IAxisVMLayersEvents	622
IAxisVMLinesEvents	622
IAxisVMLineSupportsEvents	623
IAxisVMLinkElementsEvents	623
IAxisVMLoadsEvents	623
IAxisVMLoadCasesEvents	623
IAxisVMLoadCombinationsEvents	623
IAxisVMLoadGroupsEvents	623
IAxisVMLogicalPartsEvents	623
IAxisVMMaterialsEvents	623
IAxisVMMathTextsEvents	623
IAxisVMMembersEvents	624
IAxisVMMODELsEvents	624
IAxisVMMovingLoadsEvents	626
IAxisVMMovingLoadOnBeamEvents	626
IAxisVMMovingLoadOnDomainEvents	626
IAxisVMNodalSupportsEvents	626
IAxisVMNodesEvents	626
IAxisVMPushoverHingeFunctionsEvents	626
IAxisVMRCBeamDesignEvents	626
IAxisVMRCColumnCheckingEvents	626
IAxisVMRigidBodiesEvents	626
IAxisVMRebarSteelGradesEvents	626
IAxisVMReferencesEvents	626
IAxisVMReportsEvents	626
IAxisVMReinforcementCheckEvents	627
IAxisVMResultsEvents	627
IAxisVMSectionsEvents	627
IAxisVMSeismicStoreysEvents	627
IAxisVMSettingsEvents	627
IAxisVMShearCapacityEvents	627
IAxisVMSpectrumEvents	627
IAxisVMSpringParamsEvent	627
IAxisVMSpringParamsEvents	627
IAxisVMSteelCrossSectionOptimizationEvents	628
IAxisVMSteelDesignMembersEvents	628
IAxisVMSteelDesignResultsEvents	628
IAxisVMStoreysEvents	628
IAxisVMStrressesEvents	628
IAxisVMSstructuralGridsEvents	628
IAxisVMSurfacesEvents	628
IAxisVMSurfaceSupportsEvents	628
IAxisVMTimberDesignMembersEvents	628
IAxisVMTimberDesignResultsEvents	628
IAxisVMTimeIncrementFunctionsEvents	629
IAxisVMVelocityEvents	629

IAxisVMVirtualBeamsEvents	629
IAxisVMWindowsEvents.....	629
IAxisVMWorkplanesEvents	630
IAxisVXMLAmp PanelsEvents	630
CustomFunction parameters	631
Shear force reduction	632
Eccentricity and loads on ribs	633
Changes between versions 3.x and 5.0	635
Changes between versions 5.0 and 5.1	639
Changes between versions 5.1 and 5.3	641
Changes between versions 5.3 and 6.0	642
Changes between versions 6.0 and 6.1	643
Changes between versions 6.1 and 6.2	647
Changes between versions 6.2 and 6.3	649
Changes between versions 6.3 and 6.4	650
Changes between versions 6.3 and 7.0	651
Changes between versions 7.0 and 7.1	652
Changes between versions 7.1 and 7.2	654
Changes between versions 7.2 and 7.3	657
Changes between versions 7.3 and 7.4	658
Changes between versions 7.4 and 7.5	658
Changes between versions 7.5 and 8.0	659
Changes between versions 8.0 and 8.1	661
Changes between versions 8.1 and 8.2	661
Changes between versions 8.2 and 8.3	666
Changes between versions 8.3 and 8.4	669
Changes between versions 8.4 and 9.0	669
Changes between versions 9.0 and 9.1	671
Changes between versions 9.1 and 9.2	673
Changes between versions 9.2 and 9.3	673
Changes between versions 9.3 and 15.0	675
Changes between versions 15.0 and 15.1	677
Changes between versions 15.1 and 15.3	677
AxisVM COM Plugin, Addon or AddonPlugin	681
AxisVM COM Plugin	682
Win 32/64 versions 2.0, 2.1, 2.2 & 2.3	682
.NET	684
AxisVM .NET Plugin System v2.3 for .NET Framework 4.x	684
AxisVM .NET Plugin System v2.0 for .NET Framework 2.x, 3.x	686
AxisVM COM Addon.....	688
Win 32/64 versions 1.0 and 2.0	688
.NET (IAxisVMDotNetAddonPluginInterface_v1.0)	691
AxisVM COM AddonPlugin.....	692
Win32/64 version 1.0.....	692
.NET (IAxisVMDotNetAddonPluginInterface_v1.0)	694
Important Notes for .NET	698
Late binding and early binding	699
Find GUID of the record	699
Troubleshooting.....	700
AxisVM results	700
Concrete creep	700
Development.....	701
.NET	701
Excel	702
Python	704
Known issues	704
Importing type library.....	706
Determining the major and minor version of the COM server	706
Visual C++	707
Visual Basic	708
Visual Basic For Applications (MS Office)	708
Excel 95-2003	708
Excel 2010	708

Borland Delphi.....	708
Python	708
Examples	710
Example #1: Random lines (C++).....	710
Example #2: Truss model construction.....	712
Delphi example.....	712
C++ .NET example	717
Example #3: Line and load definition (Visual Basic)	721
Example #4: Portal frame based on example PF-ST-I. axs (Delphi)	724
Example #5: Concrete plate based on example CP-ST-I. axs (Delphi).....	728
Example #6: Steel frame (Python)	730

Introduction

AxisVM like many other Windows application supports Microsoft COM technology making its operations available for external programs. Programs implementing a COM server register their COM classes in the Windows Registry providing interface information.

Any external program can get these descriptions, read object properties or call the functions provided through the interface. A program can launch AxisVM, build models, run calculations and get the results through the AxisVM COM server. This is the best way to

- build and analyse parametric models
- finding solutions with iterative methods
- build specific design extension modules

DLL modules placed in the *Plugins* folder of AxisVM are automatically included in the *Plugins* menu imitating the subfolder structure of the *Plugins* folder.

Similarly, DLL modules placed in the *Addons* folder of AxisVM are automatically loaded with AxisVM. Icons of Addons are shown on the specified AxisVM toolbar.

Developers have also an option to make DLL modules, which behaves as plugin and addon (AddonPlugin).

See section [AxisVM COM Plugin, Addon or AddonPlugin](#) for more info on the subject.

Please note:

The bookmarks tree of this document corresponds to the object model tree of AxisVM Com server.

Registration

AxisVM COM server has to be registered in the registry the operating system. Registration makes AxisVM type library available in other IDEs. COM server is registered automatically after the installation of the AxisVM.

Depending on type of selected installation, the 32 bit or the 64 bit version of the AxisVM will be registered. Registering both 32-bit and 64-bit versions of the .NET interfaces may result in failing to unregister them later.

It is important to register only the version that is required and matches with version of installed AxisVM.

Older versions of AxisVM must be unregistered from the windows registry before new registration !

Please unregister the current AxisVM related registrations before registering the new ones.

Files referred below are in the same folder where AxisVM.exe is located.

Unregister the type library and others by running files:

`!UNREGISTER_AXISVM.BAT` for 32-bit version of AxisVM

`!UNREGISTER_AXISVM_X64.BAT` for 64-bit version of AxisVM

`IAxisVMDotNetPluginInterface_v2_3`

`!UNREGISTER_DOT_NET_PLUGIN_SERVER_x32.BAT` to unregister the 32-bit version

`!UNREGISTER_DOT_NET_PLUGIN_SERVER_x64.BAT` to unregister the 64-bit version

`IAxisVMDotNetAddonPluginInterface`

`!UNREGISTER_DOT_NET_ADDONPLUGIN_SERVER_x32.BAT` to unregister the 32-bit version

`!UNREGISTER_DOT_NET_ADDONPLUGIN_SERVER_x64.BAT` to unregister the 64-bit version

Register the type library and other by running files:

Run:

`!REGISTER_AXISVM.BAT` to register the 32-bit version (optionally you can also register the 64 bit version, run: `!REGISTER_AXISVM_X64.BAT`).

`IAxisVMDotNetAddonPluginInterface`(.NET [AddonPlugins](#)).

`!REGISTER_DOT_NET_ADDONPLUGIN_SERVER_x32.BAT` if you want to use 32-bit .NET plugins or addons

`!REGISTER_DOT_NET_ADDONPLUGIN_SERVER_x64.BAT` if you want to use 64-bit .NET plugins or addons

`IAxisVMDotNetPluginInterface_v2_3` ([.NET Plugins](#)).

`!REGISTER_DOT_NET_PLUGIN_SERVER_x32.BAT` if you want to use 32-bit .NET plugins

`!REGISTER_DOT_NET_PLUGIN_SERVER_x64.BAT` if you want to use 64-bit .NET plugins

User can check the version of registered AxisVM and related .NET interfaces with RefDIIView

You can find both 32 and 64 bit versions here: https://www.nirsoft.net/utils/registered_dll_view.html

Find keyword: "axis"

You can also check 32-bit version registrations with OLE-COM-Object-Viewer:

<http://www.softpedia.com/get/System/System-Miscellaneous/OLE-COM-Object-Viewer.shtml>

Usage:

1. open OLE-COM-Object-Viewer
2. Look for AxisVM in Type Library
3. Click on it and check the version and path

If registration was successful, then the path of the registered AxisVM will be located in the windows registry.

Example for registered version 12:

Key: HKEY_CURRENT_USER\Software\InterCad\AxisVM12\

ValueName: Path

Data: C:\AxisVM12

Type library must be imported in your IDE for early binding see [here](#) and [here](#).

Important note:

Registration of multiple AxisVM COM servers from different directories could lead to unexpected problems.

You can unregister previous or irrelevant registrations of the type library and related interfaces by following methods:

1. Using the previous version of AxisVM.exe (if present), which was registered before
2. If you are still experiencing problems with COM server registration, delete all AxisVM COM server related entries from the windows registry with our tool UnregComServer.exe.

Windows 10:

Run with *!UnregAxisVMComServer.bat* as an administrator.

Windows 8:

Start the *Command Prompt* as an administrator change directory to root AxisVM and type *!UnregAxisVMComServer.bat* in the *Command Prompt*.

3. Uninstall the AxisVM then delete all left registry entries from windows registry using regedit. See this for more info:<http://support.microsoft.com/kb/217180> Look for keyword axisvm. Delete all entries where axisvm keyword appears; use F3 for find next occurrence.

Starting the COM server

Start the COM server by creating an object of `IAxisVMAApplication` interface. While the COM server is launching AxisVM, the user has to wait until AxisVM is fully loaded.

There are two ways to determine whether AxisVM has been loaded or not.

- (1) Periodically checking the **Loaded** (Boolean) property of the `AxisVMAApplication` object
- (2) Handling the **Loaded** event of the `AxisVMAApplicationEvents` object.

If AxisVM is already running when COM server is starting, the server connects to the running application, but AxisVM has to be launched with `/multiinstancecomclients` parameter. Therefore, the `AxisVMAApplication.Loaded` property should be checked first even if the second method is used. If the server connects to a running application, the `AxisVMAApplicationEvents.Loaded` event will never be called.

NOTE:

If both 32 and 64-bit versions of AxisVM are registered, the 32-bit clients will open 32-bit version of AxisVM and the 64-bit clients will open the 64-bit version of AxisVM.

Units, Data types and general errors

Units

Default metric units unless noted otherwise.

length	meter [m]
area	metes square [m^2]
mass	kilogram [kg]
temperature	Celsius [$^{\circ}C$]
time	second [s]
degree	radian [rad]
force	kilo newton [kN]
frequency	hertz [Hz]

Other units have been derived from these.

Data types

enum	EBoolean { IbFalse = 0, IbTrue = 1 }
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 SafeArrayDestroy Windows function). Safearray's lower bound must be 1! It is recommended to set the safearrays to null (nil) before calling functions.

Colour 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.

User can also use predefined AxisVM colours from [EmaterialColour](#) enum.

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

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:

EGeneralErrors:

enum EGeneralError {	
errDatabaseNotReady = -101,	model database is not available (AxisVM has not been loaded)
errNotFound = -102,	search item cannot be found or file not found
errIndexOutOfBoundsException = -103,	index is not valid
errReadOnly = -104	attempt to change a read only property

errInternalException = -105,	internal exception, database corrupted or an unknown error
errNotSupportedByNationalDesignCode = -106,	reading specific results are not supported by used code
errCOMServerInternalError = -107,	COM server internal error
errNotImplemented = -108,	function not implemented
errEnvelopeIDOutOfBounds = -109,	index of the envelope is not valid
errMinMaxNotAllowed = -110,	MinMax type is not valid
errNoLoadCaseInLoadGroups = -111,	Load groups don't contain load case(s)
errNoResults = -112,	Results not found, re-run analysis
errCriticalCombinationNotAllowed = -113,	Cases: Seismic loads are missing, no exceptional load groups, etc. Check allowed combination types with function GetValidCombinationTypes
errInvalidName = -114,	Name already exists or empty
errCombinationTypeNotAllowed = -115,	Combination type not allowed for current national design code
errInvalidEnvelopeUID = -116,	Check valid EnvelopeUID in IAxisVMEvelopes interface
errInvalidPosition = -117,	the position or the section is invalid
errIndexDuplication = -118,	index duplication in an array
errJSONpropertyMissing = -119	JSON property is missing
errMembersNotAllowed = -120	The current license does not support members in AxisVM
errCreepNotSupported = -121	The national design code does not support creep. More here...
errOutOfMemory = -122,	A special case of the internal exceptions, when there wasn't enough memory to carry out the operation

}

NOTE: Interface-specific error codes appear in interface descriptions.

Function parameters

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]**

Input and output parameter of a function: **[i/o]**

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 (it is a procedure).

0x...: hexadecimal value

Notes for Python:

[out] parameters are not required, out parameters are added to the tuple returned by the function

[i/o] parameters are required, in/out parameters are added to the tuple returned by the function

SAFEARRAYs:

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).

Properties of interface type:

Where property of the interface returns another property (e.g. RCBeamDesign of IAxisVMMModel interface) the returned property can be nil (null) if the corresponding AxisVM design module (e.g. RC2) is not available.

Compatibility

With each AxisVM version, the COM server is updated as well.

New versions get new features: new interfaces, functions, properties but sometimes some functions are renamed. Therefore, it should be the API (COM) developer's responsibility to check whether his plugin/addon is compatible with the used AxisVM. Generally, new version only has newer features and more functions.

To make sure that the client application is 100% compatible with particular version of AxisVM, the developer should compile his COM client using type library of that particular version of AxisVM. Then application can check the major and minor library version in IAxisVMAApplication interface and decide whether allows it to run or not.

Indexing

All AxisVM elements nodes, lines, members, domains etc. have an index. Member would consist of one or more lines, but finite element is made by only one line.

Hence, member will be made of one or more finite elements.

AxisVM shows only the index of the finite element or the member. User can switch between these two labelling (numbering) options in Display Options / Labels dialog box.

Index of the element (node, line, etc.) is dynamic, therefore it can change when the model is modified (element removed, added, etc.). Use **UID** (unique index) property of the element if you want to use static (fixed) index of the element.

Case sensitivity and Python

IDL language used for COM type library is not case sensitive, however Python language is case sensitive.

Type library generators are not handling this issue and the cases in names of functions, fields and properties may not exactly match with cases used in this document.

Python developers are advised to use the case used in generated interop python module when face this issue.

IAxisVMApplication

The main interface of the COM server

Enumerated types

enum	EMessageDialogButton { mdbNone = 0x0, mdbHelp = 0x1, mdbCancel = 0x2, mdbOK = 0x4, mdbNo = 0x8, mdbYes = 0x10, mdbAbort = 0x20, mdbIgnore = 0x40, mdbRetry = 0x80, mdbAll = 0x100, mdbNoToAll = 0x200, mdbAll = 0x400, mdbYesToAll = 0x800, mdbClose = 0x1000} <i>Dialog buttons</i>
enum	EMessageDialogType { mdtWarning = 0, mdtError = 1, mdtInformation = 2, mdtConfirmation = 3, mdtCustom = 4, mdtErrorConfirmation = 5 } <i>Dialog types</i>
enum	EApplicationClose { acEnableShowWarning = 0, Closing AxisVM is enabled, warns if there are COM clients connected acEnableNoWarning =1, Closing AxisVM is enabled, without warning about connected COM clients acDisable = 2 } Closing AxisVM is disabled <i>Dialog types</i>
enum	EClientAliveTest { catNone = 0, Nothing happens if client is not responding catShowWarning = 1, Warning message shows up if client is not responding catExitApplication = 2} AxisVM exits if client is not responding <i>Test if client application is responding</i>
enum	EMainFormTab = { mftGeometry = 0x0, <i>Geometry tab in AxisVM</i> mftElements = 0x1, <i>Elements tab in AxisVM</i> mftLoads = 0x2, <i>Loads tab in AxisVM</i> mftMesh = 0x3, <i>Mesh tab in AxisVM</i> mftStatic = 0x4, <i>Static tab in AxisVM</i> mftVibration = 0x5, <i>Vibration tab in AxisVM</i> mftDynamic = 0x6, <i>Dynamic tab in AxisVM</i> mftBuckling = 0x7, <i>Buckling tab in AxisVM</i> mftRCDesign = 0x8, <i>Design tab in AxisVM</i> mftSteelDesign = 0x9, <i>Steel design tab in AxisVM</i> mftTimberDesign = 0x10} <i>Timber design tab in AxisVM</i>

Error codes

```
enum EApplicationError = {
    appeUnknownUnitSystem = -100001
    appeRCBeamDesignObjectAlreadyCreated = - 100002
}
Error during setting unit system.
```

*Unknown Unit System
RC beam design object is already created, only one instance of this object is allowed to run at a time*

Functions

long **BringToFront**

Bring AxisVM window to front.

Returns 1 if successful, 0 otherwise.

long **ChangeUnitSystem ([in] BSTR Name)**

Name Case insensitive name of unit system(EU units, HUN units, RO units , SI standard units, US units)

Returns index of unit system if successful, otherwise an error code ([appeUnknownUnitSystem](#)).

long **CustomFunction ([in] long CustomID, ([in] BSTR jsonIN,**

[i/o] BSTR jsonOUT)

CustomID Custom function index, default 1

jsonIN Input parameters in JSON format

jsonOUT Output parameters in JSON format

This function was introduced to allow for new functions and features which will be fully implemented in next version of COM server. Read more [here](#).

Returns number of fields in JSON output if successful, otherwise an error code ([errJSONpropertyMissing](#), [errNotImplemented](#)).

void **DisableMainForm**

Disable all controls on AxisVM form including the selection toolbar.

void **EnableMainForm**

Enable all controls on AxisVM form.

void **HandleMessages**

Handles the messages in AxisVM and the COM client while the COM client is blocking the thread being executed. It is used as the body of a loop, that will continue to run till an external event sets a flag. The flag is set in the COM client usually through an event such as

IAxisVMMODELSEvents.SelectionProcessingChanged, when for example it is called with NewStatus = false (signalling the fact that the user has ended the selection process). While the thread being executed is blocked, the COM client will still receive events indirectly through HandleMessages, making it possible to process those events and ultimately to set the flag that will unblock the thread which was blocked through the loop.

[**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 up the [EMessageDialogButton](#) values of the required buttons(e.g. for showing YES and NO buttons; values is: mdbNo + mdbYes = 0x18)

		<i>Shows a message dialog in AxisVM. The result is the value of the button the user clicked.</i>
EMessageDialogButton*		MessageDlg_vb (Visual Basic compatible function of MessageDlg)
long	UnLoadCOMclients	<i>Stops, unloads and releases memory taken by COM clients (addons, plugins and addon-plugins). Returns 1 if successful.</i>
long	Quit	<i>Quit from AxisVM. Stops, unloads and releases memory taken by COM clients (addons, plugins and addonplugins) then AxisVM quits.</i>

Properties

EApplicationClose		ApplicationClose • <i>Determines how AxisVM should close if other clients are connected; if it should show a message or disable AxisVM closing while COM client runs. Read and write property</i>
ELongBoolean		AskCloseOnLastReleased • <i>Determines whether the COM server shutdown displays a query to close the AxisVM application as well. Read and write property</i>
ELongBoolean		AskSaveOnLastReleased• <i>Before COM server shutdown displays a query to save the model. Read and write property Note: the model should be saved after modification or if newer version of AxisVM is used.</i>
EAxisVMPlatform		AxisVMPlatform <i>Get AxisVM version type (32/64bit)</i>
AxisVMCatalog *		Catalog <i>Get AxisVM catalog interface for standard materials and cross-sections.</i>
EClientAliveTest		ClientAliveTest <i>Get or set what happens in AxisVM if client application is not responding</i>
long	ClientAliveTestIntervalSec	<i>Get or set interval (seconds) of how often AxisVM checks if client application is responding or not</i>
ELongBoolean		CloseOnLastReleased • <i>If AskCloseOnLastReleased = False this property determines if AxisVM is closed after completing COM server shutdown. Read and write property</i>
AxisVMCrossSectionEditor		CrossSectionEditor <i>Get an interface for editing cross-sections.</i>
ELongBoolean		COMclientsLoaded <i>Returns lbTrue if AxisVM has loaded any COM clients (addons, plugins or addon-plugins). Read only property</i>
BSTR	FullExePath	<i>Get axisvm.exe path</i>
long	LibraryMajorVersion	<i>Get major version number of the COM-server.</i>
long	LibraryMinorVersion	<i>Get minor version number of the COM-server.</i>
ELongBoolean		Loaded <i>Shows if AxisVM has been loaded or not. Read only property.</i>
EWindowState		MainFormWindowState • <i>Get or set window state of the AxisVM.</i>
RWindowPosition		MainFormWindowPosition • <i>Get or set window position of the AxisVM.</i>

long	MainFormWindowLeft • <i>Get or set left distance of the main window of AxisVM.</i>
long	MainFormWindowTop • <i>Get or set top distance of the main window of AxisVM.</i>
long	MainFormWindowWidth • <i>Get or set width of the main window of AxisVM.</i>
long	MainFormWindowHeight • <i>Get or set height of the main window of AxisVM.</i>
<u>EMainFormTab</u>	MainFormTab • <i>Get or set which main tab in AxisVM should be activated.</i>
long	ModalLevel <i>Get number of opened modal forms in AxisVM. If zero, then no modal forms are open.</i>
<u>AxisVMMODELS*</u>	Models <i>Gets an interface containing the opened models. The number of models is limited to one.</i>
<u>AxisVMOBJECTCREATOR</u>	ObjectCreator <i>Get an interface to ObjectCreator</i>
BSTR	Version <i>Get AxisVM program version string. Not the same as the COM-server version.</i>
<u>ELongBoolean</u>	Visible • <i>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. Read and write property</i>

IAxisVMCatalog

AxisVM catalog interface for materials and cross-sections.

Enumerated types

enum	ENationalDesignCode = {	
	ndcOther = \$00000000	<i>Design code not specified</i>
	ndcHungarian_MSZ = \$00000001	<i>Hungarian design code MSZ</i>
	ndcEuroCode = \$00000002	<i>Eurocode design code</i>
	ndcRomanian_STAS = \$00000004	<i>Romanian design code STAS</i>
	ndcDutch_NEN = \$00000005	<i>Dutch design code NEN</i>
	ndcGerman_DIN1045_1 = \$00000006	<i>German design code DIN 1045-1</i>
	ndcSwiss_SIA26x = \$00000007	<i>Swiss design codes SIA26x</i>
	ndcEuroCode_GER = \$00000008	<i>Eurocode with German national annex</i>
	ndcItalian = \$00000009	<i>Eurocode with Italian national annex</i>
	ndcEuroCode_Austrian = \$0000000A	<i>Eurocode with Austrian national annex</i>
	ndcEuroCode_UK = \$0000000B	<i>Eurocode with UK national annex</i>
	ndcEuroCode_NL = \$0000000C	<i>Eurocode with Netherland national annex</i>
	ndcEuroCode_FIN = \$0000000D	<i>Eurocode with Finish national annex</i>
	ndcEuroCode_RO = \$0000000E	<i>Eurocode with Romanian national annex</i>
	ndcEuroCode_HU = \$0000000F	<i>Eurocode with Hungarian national annex</i>
	ndcEuroCode_CZ = \$00000010	<i>Eurocode with Czech national annex</i>
	ndcEuroCode_B = \$00000011	<i>Eurocode with Belgian national annex</i>
	ndcEuroCode_PL = \$00000012	<i>Eurocode with Polish national annex</i>
	ndcEuroCode_DK = \$00000013	<i>not implemented(Eurocode,Danish national annex)</i>
	ndcEuroCode_S = \$00000014	<i>not implemented(Eurocode, Swedish national annex)</i>
	ndcUS = \$00000015	<i>US not implemented yet</i>
	ndcCA_NBCC = \$00000016	<i>CA_NBCC not implemented yet</i>
	ndcCA_Ontario = \$00000017	<i>CA_ONTARIO not implemented yet</i>
	ndcCA_Bridge = \$00000018	<i>CA_Bridge not implemented yet</i>
	ndcEuroCode_SK = \$00000019	<i>Eurocode with Slovak national annex</i>
	ndcEuroCode_LV = \$00000020	<i>Eurocode with Latvian national annex</i>
	}	

Functions

long **GetCrossSection** ([in] **ECrossSectionShape** *CrossSectionShape*, [in] BSTR **TableName**, [in] BSTR *Name*, [out] AxisVMCrossSection* *CrossSection*)

CrossSectionShape *cross-section shape*

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	GetCrossSection ([in] ECrossSectionShape CrossSectionShape , [in] BSTR TableName , [in] BSTR Name , [out] AxisVMCrossSection * CrossSection)	cross-section
	<p>CrossSectionShape</p> <p style="margin-left: 20px;">TableName <i>name of the cross-section table</i></p> <p style="margin-left: 20px;">Name <i>name of the cross-section in the table</i></p> <p style="margin-left: 20px;">CrossSection <i>an instance of the cross-section object</i></p> <p><i>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).</i></p>	
long	GetCrossSectionEx ([in] ECrossSectionShapeEx CrossSectionShape , [in] BSTR TableName , [in] BSTR Name , [out] AxisVMCrossSection * CrossSection)	cross-section
	<p>CrossSectionShape</p> <p style="margin-left: 20px;">TableName <i>name of the cross-section table</i></p> <p style="margin-left: 20px;">Name <i>name of the cross-section in the table</i></p> <p style="margin-left: 20px;">CrossSection <i>an instance of the cross-section object</i></p> <p><i>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).</i></p>	
long	GetCrossSectionNamesEx ([in] ECrossSectionShapeEx CrossSectionShape , [in] BSTR TableName , [out] SAFEARRAY(BSTR)* Names)	
	<p>CrossSectionShape <i>cross-section shape</i></p> <p style="margin-left: 20px;">TableName <i>name of the cross-section table</i></p> <p style="margin-left: 20px;">Names <i>list of cross-section names</i></p> <p><i>Get a list of cross-section names of the given type from a cross-section table. Returns the number of list items.</i></p>	
long	GetCrossSectionTableNames ([in] ECrossSectionShape CrossSectionShape , [out] SAFEARRAY(BSTR)* TableNames)	
	<p>CrossSectionShape <i>cross-section shape</i></p> <p style="margin-left: 20px;">TableNames <i>list of cross-section table names</i></p> <p><i>Get a list of cross-section table names of the given type. Returns the number of list items.</i></p>	
long	GetCrossSectionTableNamesEx ([in] ECrossSectionShapeEx CrossSectionShape , [out] SAFEARRAY(BSTR)* TableNames)	
	<p>CrossSectionShape <i>cross-section shape</i></p> <p style="margin-left: 20px;">TableNames <i>list of cross-section table names</i></p> <p><i>Get a list of cross-section table names of the given type. Returns the number of list items.</i></p>	
long	GetMaterial ([in] ENationalDesignCode DesignCode , [in] BSTR Name , [out] AxisVMMaterial * Material)	
	<p style="margin-left: 20px;">DesignCode <i>national design code of the material</i></p> <p style="margin-left: 20px;">Name <i>material name</i></p> <p style="margin-left: 20px;">Material <i>an instance of the material object</i></p> <p><i>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).</i></p>	
long	GetMaterialNames ([in] ENationalDesignCode DesignCode , [out] SAFEARRAY(BSTR)* Names)	
	<p style="margin-left: 20px;">DesignCode <i>national design code of the material</i></p> <p style="margin-left: 20px;">Names <i>list of material names</i></p>	

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

long **GetMaterialNamesByType** ([in] [ENationalDesignCode](#) **DesignCode**,
[in] [EMaterialType](#) **MaterialType**, [out] SAFEARRAY(BSTR)* **Names**)

DesignCode *national design code of the material*
MaterialType *Type of the material*
Names *list of material names*

*Get a list of material names of the given design code of specified material type.
Returns number of names.*

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

DesignCode *national design code of the material*
Names *list of names of rebar grades*

Get a list of names of rebar grades of the given design. Returns number of names.

long **GetXLAMmanufacturers** ([out] SAFEARRAY(BSTR)* **Manufacturers**)

Manufacturers *list of XLAM manufacturers,*

Get a list of XLAM manufacturers. Returns the number of list items.

long **GetXLAMnamesByManufacturers** ([in] BSTR **Manufacturer**,
[out] SAFEARRAY(BSTR)* **XLAMnames**)

Manufacturer *name of the manufacturer*
XLAMnames *list of XLAM panels*

Get a list of XLAM panels by manufacturer name. Returns the number of list item, otherwise returns an error code ([errNotFound](#))

IAxisVMCrossSectionEditor

AxisVM interface for editing cross-sections.

Error codes

```
enum ECrossSectionEditorError {
    cseeEditorNotOpened = -100001, Editor dialog window not opened
    cseeEditorModelsNotSolidCrossSection = -100002 } Editor is not in solid cross-section mode
```

Records / structures

```
RCrossSectionUserParams = (
    long IParam User parameter No. 0
    double dParam1 User parameter No. 1
    double dParam2 User parameter No. 2
    double dParam3 User parameter No. 3
    double dParam4 User parameter No. 4
    double dParam5 User parameter No. 5
    double dParam6 User parameter No. 6
    double dParam7 User parameter No. 7
    double dParam8 User parameter No. 8
    double dParam9 User parameter No. 9
    double dParam10 User parameter No. 10
)
```

Functions

```
long AddCustomWithUserParams ([in] BSTR Name, [in] AxisVMPolygon2dList * ShapePolygonList,
    [in] ECrossSectionProcess Process, [in] double b, [in] double h, [in] double tf, [in] double tw, [in]
    double alpha, [in] double Yg, [in] double Zg, [i/o] RCrossSectionUserParams *
    CrossSectionUserParams)
```

Name *Name of the cross-section*

ShapePolygonList *Polygon list with shape of the cross-section*

Process *the manufacturing process*

b *Breadth(width) of the cross-section*

h *Height of the cross section*

tf *cross-section wall thickness at the flange*

tw *cross-section wall thickness at the web*

alpha *angle of rotation of the source shape around its centre of gravity*

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*

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*

CrossSectionUserParams *User parameters*

*Returns 1 if successful, otherwise returns an error code ([errIndexOutOfBoundsException](#),
[cseeEditorModelsNotSolidCrossSection](#), [cseeEditorNotOpened](#), [errDatabaseNotReady](#))*

long **AddCustomWithUserParamsAsArray** ([in] BSTR Name, [in] AxisVMPolygon2dList * ShapePolygonList, [in] ECrossSectionProcess Process, [in] double b, [in] double h, [in] double tf, [in] double tw, [in] double alpha, [in] double Yg, [in] double Zg, [in] long IParam, [in] SAFEARRAY(double) CrossSectionUserParams)

Name *Name of the cross-section*
ShapePolygonList *Polygon list with shape of the cross-section*
Process *the manufacturing process*
b *Breadth(with) of the cross-section*
h *Height of the cross section*
tf *cross-section wall thickness at the flange*
tw *cross-section wall thickness at the web*
alpha *angle of rotation of the source shape around its centre of gravity*
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*
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*
IParam *User parameter*
CrossSectionUserParams *Array with user parameters (only first 10 elements of the array are used)*

Returns 1 if successful, otherwise returns an error code ([errIndexOutOfBoundsException](#), [cseeEditorModelsNotSolidCrossSection](#), [cseeEditorNotOpened](#), [errDatabaseNotReady](#))

long **AddCustomWithUserParamsAsArray_vb** (Visual Basic compatible function of **AddCustomWithUserParamsAsArray**)

long **AddCustomWithUserParamsAsByteArray** ([in] BSTR Name, [in] AxisVMPolygon2dList * ShapePolygonList, [in] ECrossSectionProcess Process, [in] double b, [in] double h, [in] double tf, [in] double tw, [in] double alpha, [in] double Yg, [in] double Zg, [in] SAFEARRAY(double) CrossSectionUserParams)

Name *Name of the cross-section*
ShapePolygonList *Polygon list with shape of the cross-section*
Process *the manufacturing process*
b *Breadth(with) of the cross-section*
h *Height of the cross section*
tf *cross-section wall thickness at the flange*
tw *cross-section wall thickness at the web*
alpha *angle of rotation of the source shape around its centre of gravity*
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*
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*
CrossSectionUserParams *Array with user parameters (first element is 32bit long other 10 elements of the array are 64bit doubles)*

Returns 1 if successful, otherwise returns an error code ([errIndexOutOfBoundsException](#), [cseeEditorModelsNotSolidCrossSection](#), [cseeEditorNotOpened](#), [errDatabaseNotReady](#))

long **AddCustomWithUserParamsAsByteArray_vb** (Visual Basic compatible function of **AddCustomWithUserParamsAsByteArray**)

long ReplaceWithCustomAndUserParamsAsByteArray ([in] long Index, [in] BSTR Name, [in] AxisVMPolygon2dList * ShapePolygonList, [in] ECrossSectionProcess Process, [in] SAFEARRAY(byte) CrossSectionUserParams)

Index cross-section index
Name Name of the cross-section
ShapePolygonList Polygon list with shape of the cross-section
Process the manufacturing process
CrossSectionUserParams Array with user parameters (first element is 32bit long other 10 elements of the array are 64bit doubles)
Returns cross-section index if successful, otherwise returns an error code ([errIndexOutOfBounds](#), [errDatabaseNotReady](#))

long Clear
Clears everything from cross section editor.
Returns 1 if successful, otherwise returns an error code ([cseeEditorNotOpened](#))

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**

Get number of opened models. It is always 1.

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

Get or set model object by its index. Index must be 1.

IAxisVMModel

The opened AxisVM Model

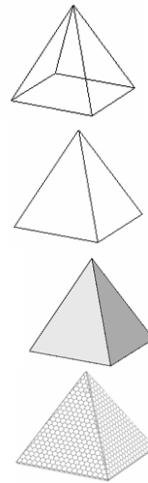
Enumerated types

enum **EDisplay** = {
 dWireframe = 0x0, *Wireframe*

dHidden = 0x1, *wireframe with hidden lines removed*

dRendered = 0x2, *rendered view*

dTextured = 0x3 } *rendered view with textures*



Display modes of the model.

enum **EDXFVersion** = {
 dxFR12 = 0x0, *DXF format of AutoCAD 12*
 dxFR2000 = 0x1 } *DXF format of AutoCAD 2000*
 DXF versions

enum **EAxisVMPlatform** = {
 avmp32 = 0x0, *32 bit AxisVM*
 avmp64 = 0x1 } *64 bit AxisVM*
Version of AxisVM

enum **EFileImportAs** = {
 fiaActualNodes = 0x0, *Create actual nodes and lines to the model*
 fiaBackgroundLayer = 0x1 } *Add it to background layers*
How to import lines and nodes.

enum **EFileImportMode** = {
 fimOverwrite = 0x0, *Overwrite actual nodes and lines to the model*
 fimAdd = 0x1 } *Add to actual nodes and lines to the model*
Import mode of files

enum **EFileImportPlane** = {
 fipPlaneXY = 0x0, *XY plane*
 fipPlaneXZ = 0x1, *XZ plane*
 fipPlaneYZ = 0x2, *YZ plane*
 fipWorkPlane = 0x3 *Work plane*
}

Plane where the 2D elements should be imported

enum	EIfcVersion = {	
	ifc20 = 0x0,	<i>IFC 2.0</i>
	ifc2x = 0x1,	<i>IFC 2x</i>
	ifc2x2 = 0x2,	<i>IFC 2x2</i>
	ifc2x3 = 0x3 }	<i>IFC 2x3</i>
	<i>IFC versions</i>	
enum	ELanguage = {	
	IngEnglish = 0x00,	<i>English</i>
	IngHungarian = 0x01,	<i>Hungarian</i>
	IngGerman = 0x02,	<i>German</i>
	IngRomanian = 0x03,	<i>Romanian</i>
	IngSpanish = 0x04,	<i>Spanish</i>
	IngItalian = 0x05,	<i>Italian</i>
	IngRussian = 0x06,	<i>Not used</i>
	IngPortuguese = 0x07,	<i>Portuguese</i>
	IngSerbian = 0x08,	<i>Serbian</i>
	IngDutch = 0x09,	<i>Dutch</i>
	IngFrench = 0x0A,	<i>French</i>
	IngHebrew = 0x0B,	<i>Not used</i>
	IngArabic = 0x0C,	<i>Not used</i>
	IngCzech = 0x0D,	<i>Czech</i>
	IngSlovakian = 0x0E,	<i>Slovakian</i>
	IngBrazilianPortuguese = 0x0F,	<i>Brazilian Portuguese</i>
	IngGreek = 0x0F,	<i>Greek</i>
	IngCroatian = 0x10,	<i>Croatian</i>
	IngPolish = 0x11	<i>Polish</i>
	IngBulgarian = 0x12 }	<i>Bulgarian</i>
	<i>Program or report languages</i>	
enum	ELengthUnit = {	
	lu_mm = 0x0,	<i>mm</i>
	lu_cm = 0x1,	<i>cm</i>
	lu_dm = 0x2,	<i>dm</i>
	lu_m = 0x3,	<i>m</i>
	lu_inch = 0x4,	<i>inch</i>
	lu_foot = 0x5,	<i>foot</i>
	lu_yard = 0x6 }	<i>yard</i>
	<i>Length units</i>	
enum	ESelectionType = {	
	seltNode = 0x01,	<i>nodes</i>
	selMidPoint = 0x02,	<i>line midpoints</i>
	selAllLines = 0x03,	<i>all line elements(beams, ribs, trusses) and lines</i>
	selTruss = 0x04,	<i>trusses</i>
	selBeam = 0x05,	<i>beams</i>
	selRib = 0x06,	<i>ribs</i>
	selSurfaceEdge = 0x07,	<i>surface edges</i>
	selRigidBody = 0x08,	<i>rigid bodies</i>
	selDiaphragm = 0x09,	<i>diaphragms</i>
	selGap = 0x0A,	<i>gap elements</i>
	selSpring = 0x0B,	<i>springs</i>
	selLink = 0x0C,	<i>link elements</i>
	selAllSurfaces = 0x0D,	<i>surfaces</i>
	selSurfaceMembrane = 0x0E,	<i>membrane elements</i>
	selSurfacePlate = 0x0F,	<i>plate elements</i>
	selSurfaceShell = 0x10,	<i>shell elements</i>

```

seltAllDomains = 0x11,
seltDomainMembrane = 0x12,
seltDomainPlate = 0x13,
seltDomainShell = 0x14,
seltHole = 0x15,
seltAllSupports = 0x16,
seltNodalSupport = 0x17,
seltLineSupport = 0x18,
seltSurfaceSupport = 0x19,
seltReference = 0x1A,
seltAllLoads = 0x1B,
seltLoadDomainConcentrated = 0x1C,
seltLoadDomainPoly = 0x1D,
seltLoadNodalConcentrated = 0x1E,
seltLoadBeamConcentrated = 0x1F,
seltLoadBeamDistributed = 0x20,
seltLoadDomainDistributed = 0x21,
seltLoadDomainFluid = 0x22 }

seltLoadMoving = 0x23,
seltLoadDynamic = 0x24,
seltArchColumn = 0x25,
seltArchBeam = 0x26,
seltArchWall = 0x27,
seltArchSlab = 0x28,
seltArchRamp = 0x29,
seltLoadPanel = 0x2A,
seltLoadPanelEdge = 0x2B
seltVirtualBeam = 0x2C,
seltLinesOnly = 0x2D
}

Selection types

```

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.

enum **EWindowState** = {
wsMaximized = 0x0, *Maximized window*
wsMinimized = 0x1, *Minimized window*
wsNormal = 0x2 } *Normal window*
Main window states.

enum **EIFCDomainReinforcementLinkType** = {
idrltUbar = 0x0, *see AxisVM manual for more info*
idrltOverlap = 0x1, *see AxisVM manual for more info*
idrltLbar = 0x2, *see AxisVM manual for more info*
idrltNone = 0x3 } *see AxisVM manual for more info*
Link type for IFC export

enum **EIFCQuestionableDomainLink** = {
iqdlNone = 0x0, *see AxisVM manual for more info*
iqdlReinforce = 0x1 } *see AxisVM manual for more info*
What to do with questionable link type for IFC export

enum	EIFCDomainOverlap = { idoSeparately = 0x0, idoAnchored = 0x1, idoOverRun = 0x2 }	see AxisVM manual for more info see AxisVM manual for more info see AxisVM manual for more info
<i>How to deal with overlapping domains.</i>		
enum	EIFCimportMethod = { iimStaticModel = 0x0, iimArchitecturalModelObjects = 0x1 }	<i>structural elements, see AxisVM manual for more info</i> <i>architectural elements, see AxisVM manual for more info</i>
<i>What type of elements should be used for IFC import</i>		
enum	EIFCopeningsAlignedToDomainEdge = { ioatdeImportAsOpenings = 0x0, ioatdeAdjustTheDomain = 0x1 }	<i>openings impted as elements</i> <i>contour of domain adjusted for openings</i>
<i>How to import openings</i>		
enum	ECompanyLogoPosition = { clpNoLogo = 0x0, clpLeft = 0x1, clpRight = 0x2, clpTopLeft = 0x3, clpTopCenter = 0x4, clpTopRight = 0x5 }	<i>no logo</i> <i>logo on the left</i> <i>logo on the right</i> <i>logo on the top left</i> <i>logo on the top center</i> <i>logo on the top right</i>
enum	ECompanyLogoSizeOption = { cIsoAuto = 0x0, cIsoWidth = 0x1, cIsoHeight = 0x2 }	<i>automatic</i> <i>width</i> <i>height</i>
enum	EGeneralAlignmentHorizontal = { gahLeft = 0x0, gahRight = 0x1, gahCenter = 0x2 }	<i>justified to the left horizontally</i> <i>justified to the right horizontally</i> <i>justified to the center horizontally</i>
enum	EGeneralAlignmentVertical = { gavTop = 0x0, gavBottom = 0x1, gavCenter = 0x2 }	<i>justified to the top vertically</i> <i>justified to the bottom vertically</i> <i>justified to the center vertically</i>
enum	EaxsImportCustomParts = { aicpToActiveCustomParts = 0x0, aicpPreserveNames = 0x1, aicpNoCustomParts = 0x2 }	<i>all custom parts to active part</i> <i>custom part names of imported model remain same</i> <i>ignore custom parts of imported model</i>

Records / structures

	RWindowPosition = (
long	Top	Top
long	Left	Left
long	Width	Width
long	Height	Height
)	

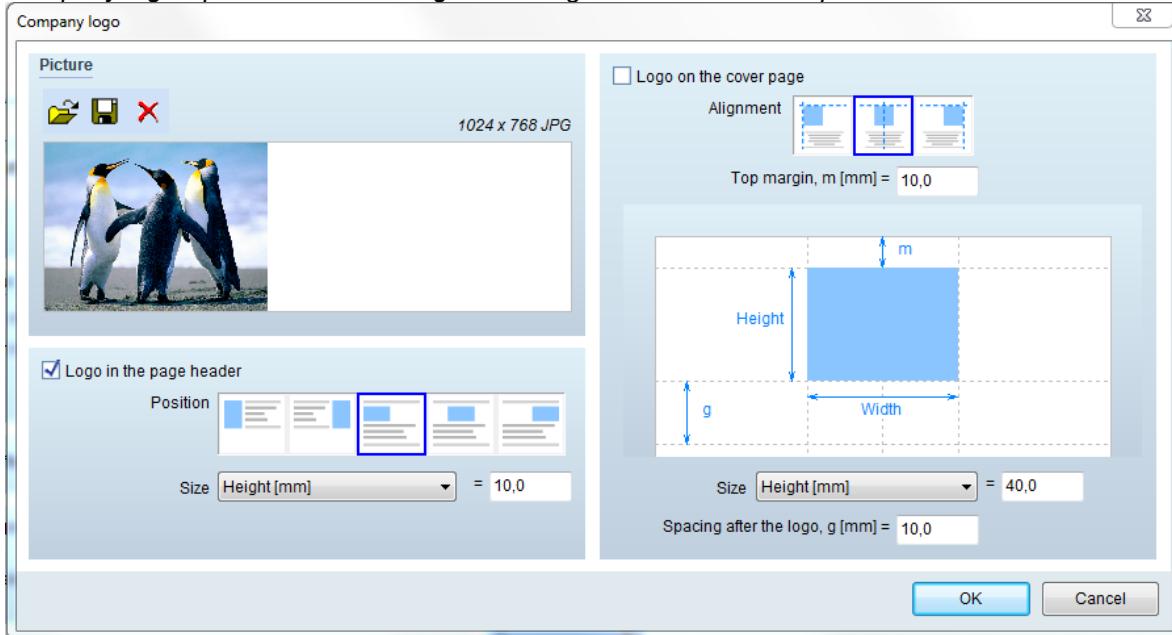
ElongBoolean	RIFCExportReinforcementParams = (see manual
ElongBoolean	EdgeReinforcementNeeded	see manual
ElongBoolean	AdditionalHoleReinforcement	see manual
ElongBoolean	TypicalLapLengthByDesignCode	see manual
ElongBoolean	ConcaveCornerLapLengthByDesignCode	see manual
double	TypicalLapLengthByUser	see manual
double	ConcaveCornerLapLengthByUser	see manual
double	AdditionalHoleRebarDiameter	see manual
	DomainReinforcementLinkType	<i>Link type for IFC export, see manual</i>
EIFCDomainReinforcementLinkType		<i>type of link export, see manual</i>
EIFCQuestionableDomainLink	QuestionableDomainLink	<i>see manual</i>
EIFCDomainOverlap	DomainOverlap_Larger	see manual
EIFCDomainOverlap	DomainOverlap_Smaller	see manual
ElongBoolean	ModifyClosedLinks	see manual
double	ClosedLinkRatio	see manual
)	
EFileImportMode	RIFCimportParameters = (<i>import mode of files</i>
EIFCimportMethod	ImportMode	<i>type of elements</i>
	ImportMethod	<i>used for ifc import</i>
double	MaxDeviation	<i>max. deviation of the arc, ignored if zero</i>
double	ByAngle	<i>angle division of the arc, ignored if zero</i>
double	JoinIfObjectsAreCloserThan	<i>join closer objects</i>
EFileImportAs	ImportAs	<i>how to import lines and nodes.</i>
EIFCopeningsAlignedToDomainEdge	OpeningsAlignedToDomainEdge	<i>how to import openings</i>
)	
ELengthUnit	RDXFimportParameters = (
double	CoordinateUnit	<i>type units used in file</i>
double	MaxDeviation	<i>max. deviation of the arc, ignored if zero</i>
double	GeometryCheckTolerance	<i>used for geometry check</i>
	CoordinateScaleFactor	<i>scale factor</i>
EFileImportAs	ImportAs	<i>how to import lines and nodes.</i>
EFileImportMode	ImportMode	<i>import mode of files</i>
EIFCimportMethod	ImportMethod	<i>type of elements used for ifc import</i>
EFileImportPlane	BasePlane	<i>base plane</i>
Rpoint3D	PlaceOffset	<i>Offset of the origin</i>
ElongBoolean	VisibleLayersOnly	<i>see AxisVM manual</i>
ElongBoolean	ImportHatch	<i>see AxisVM manual</i>
ElongBoolean	ActivateDXFonAllDrawings	<i>see AxisVM manual</i>
long	WorkPlaneIndex	<i>index of the workplane</i>
)	
RPDFimportParameters	RPDFimportParameters = (
long	PageNumber	<i>page number , first is 1</i>
double	MaxDeviation	<i>max. deviation of the arc, ignored if zero</i>
double	GeometryCheckTolerance	<i>used for geometry check</i>
	Scale	<i>scale factor</i>
ElongBoolean	ImportLineWidth	<i>line width from file will used</i>
ElongBoolean	ImportText	<i>if lbTrue, then text will be also imported</i>
EFileImportAs	ImportAs	<i>how to import lines and nodes.</i>
EFileImportMode	ImportMode	<i>import mode of files</i>
EIFCimportMethod	ImportMethod	<i>type of elements used for ifc import</i>
EFileImportPlane	BasePlane	<i>base plane</i>
Rpoint3D	PlaceOffset	<i>Offset of the origin</i>
long	WorkPlaneIndex	<i>index of the workplane</i>
)	

```

RCompanyLogoParameters = (
  ElongBoolean ShowInHeader
  ECompanyLogoPosition HeaderPosition
  ECompanyLogoSizeOption HeaderSizeOption
  double HeaderSize
  ElongBoolean ShowOnCover
  EGeneralAlignment CoverAlignmentHorizontal
  double TopMargin
  double SpacingAfter
  ECompanyLogoSizeOption CoverSizeOption
  double CoverSize
)

```

Company logo's parameters can be given through Preferences -> Report menu



```

RAXImportParameters = (
  RPoint3d Place
  double CheckTolerance
  ELongBoolean MergeLoadCases
  ELongBoolean UserInteraction
  ELongBoolean IgnoreDesignCode
  ELongBoolean MergeLayers
  EaxsImportCustomParts CustomParts
)

```

JSON string properties

BSTR double	jsonFileName rAura	<i>file name of json string</i> <i>In the collision detection of objects there can be uncertainties when two bodies just touch each other. In order to avoid or minimize this effect, all bounding boxes are inflated by a tiny value. This value can be set here. Default: 0,01</i>
double double long ElongBoolean	rTessDistance rEps rTessDegree rTessByDegree	<i>Maximal deviation from the arc for tessellation. Default: 0,05</i> <i>Tolerance used in geometric calculations. Default: 1E-05</i> <i>Tesselation degree Default: 5</i> <i>Certain geometric calculations require arcs to be tesselated. This can happen either by an angle parameter or by maximal deviation from the arc. Default: True</i>

ElongBoolean	rModStatModel	<i>If this property is set true, the program modifies the raw geometric data in order to achieve a proper structural model. In this case, colliding beam are connected with a rigid body, regions with a nearby beam are converted to ribbed region, etc. However there are expensive calculations that may result slow import process.</i> <i>Default: True</i>
ElongBoolean	rClearBefore	<i>If there are elements in the AxisVM database, user can select whether to overwrite them or just refresh modell data with the new items. Default: True</i>
long	rDefMatType	<i>National design code: ENationalDesignCode, default: ndcEuroCode as long</i>
string	rDefMatName	<i>Name of default material, default: 25/30</i>
ElongBoolean	rSuppressDialogs	<i>Hide import dialogs, default: True</i>

Error codes

enum	EModelError = {	
	meCannotExport = -100001;	<i>export failed</i>
	meNoSeismicParams = -100002;	<i>seismic parameters not available</i>
	meErrorSettingSeismicParams = -100003;	<i>seismic parameters incorrect</i>
	meLoadCaseLoadCombinationNotFound = -100004;	<i>SetSeismicParams can return this error when LoadCaseLoadCombination is not specified in RSeismicParams type record</i>
	mePianoCanNotUpdate = -100005;	
	mePianoCanNotFindFile = -100006;	
	mePianoInternalException = -100007;	
	meRCBeamDesignDisabled = -100008;	
	meRCColumnCheckingDisabled = -100009;	<i>When IAxisVMRCBeamDesign interface already created or extension module RC1 is not available</i>
	meSteelDesignMembersDisabled = -100010;	<i>When IAxisVMRCColumnChecking interface already created or extension module RC2 is not available</i>
	meActualReinforcementDisabled = -100011;	<i>SteelDesignMembers interface already created or extension module SD1 not available</i>
	meIFCmoduleNotAvailable = -100012;	<i>extension module IFC is not available</i>
	meDXFmoduleNotAvailable = -100013;	<i>extension module DXF is not available</i>
	meSE1moduleNotAvailable = -100014;	<i>extension module SE1 is not available</i>
	meTD1moduleNotAvailable = -100015;	<i>extension module TD1 is not available</i>
	meSWGmoduleNotAvailable = -100016;	<i>extension module SWG is not available</i>
	meSE2moduleNotAvailable = -100017;	<i>extension module SE2 is not available</i>
	meDYNmoduleNotAvailable = -100018;	<i>extension module DYN is not available</i>
	meCannotSaveGlobalData = -100019;	<i>Internal error during saving data</i>
	meCannotReadGlobalData = -100020;	<i>Internal error during reading data</i>
	meInvalidDataName = -100021;	<i>data name is empty or too long (max 32 chars)</i>
	meInvalidOrEmptyGlobalData = -100022;	<i>global data is empty or inaccessible</i>
	meGlobalDataNotFound = -100023;	<i>global data is not found in the file</i>
	meDataNameAlreadyExists = -100024;	<i>global data with DataName already exists in the axs file</i>
	meFileNotExists = -100025;	<i>file not exists</i>

mePDFmoduleNotAvailable = -100026;	<i>extension module PDF is not available</i>
meReinforcementForExportNotAvailable = -100027;	<i>reinforcement parameters not found error during PDF import, some objects might be missing</i>
mePDFimportCannotReadAllObjects = -100028;	<i>end of file not found in PDF</i>
mePDFimportNoEOF = -100029;	<i>graphicac elements (lines, etc.) not found in PDF</i>
mePDFimportNoGraphicElements = -100030;	<i>Unknown error during PDF import</i>
mePDFimportFailure = -100031;	<i>PDF de-compression error</i>
mePDFimportErrorInCompressedData = -100032;	<i>PDF import object is missing</i>
mePDFimportObjectMissing = -100033;	<i>PDF compression method is unknown</i>
mePDFimportUnknownCompression = -100034;	<i>PDF import data stream not found</i>
mePDFimportStreamDataNotFound = -100035;	<i>PDF import data stream length not found</i>
mePDFimportStreamLengthNotFound = -100036;	<i>graphicac elements (lines, etc.) not found on specified page in the PDF file</i>
mePDFimportNoGraphicElementsOnPage = -100037;	<i>specified page not found in the PDF file</i>
mePDFimportPageNotFound = -100038;	<i>PDF linearization is unknown</i>
mePDFimportUnknownLinearization = -100039;	<i>X ref parsing error</i>
mePDFimportReferenceStreamParsingError = -100040;	<i>unknown embedded object in the PDF file</i>
mePDFimportUnknownEmbeddedObject = -100041;	<i>reference table parsing error</i>
mePDFimportReferenceTableParsingError = -100042;	<i>unknown error during PDF import</i>
mePDFimportUnknownError = -100043;	<i>Glyphlist not found in the PDF file</i>
mePDFimportFileGlyphlistNotFound = -100044;	<i>PDF file is encrypted</i>
mePDFimportPDFfileIsEncrypted = -100045;	<i>specified page not found in the PDF file</i>
mePDFimportPagesCannotBeFound = -100046;	<i>invalid deviation or ByAngle</i>
meIFCInvalidDeviationOrByAngle = -100047;	<i>Structural elements not found</i>
meIFCNoStaticData = -100048;	<i>IFC version not found</i>
meIFCVersionNotFound = -100049;	<i>Other IFC error</i>
meIFCOtherError = -100050;	<i>deviation and/or ByAngle is 0</i>
meIFCMaxDeviationAndByAngleIsZero = -100051;	<i>module SD9 is not available</i>
meSD9moduleNotAvailable = -100052;	<i>module TD9 is not available</i>
meTD9moduleNotAvailable = -100053;	<i>module Revit is not available</i>
meRevitModuleNotAvailable = -100054;	<i>Tess degree is out of the range (acceptable range is from 5 to 15)</i>
meRevitImportTessDegreeOutOfRange = -100055 }	

Error during access to some interfaces, setting parameters and file export

Functions

void **BeginUpdate**

BeginUpdate can be called before modifying the model. When all changes are complete, an EndUpdate must be called to update the model.

long **DeleteAPIGlobalData ([in] BSTR DataName)**

DataName name of the saved API data

Deletes all data saved under name DataName. If successful it returns size of deleted data, otherwise it returns error ([meInvalidDataName](#), [errDatabaseNotReady](#)).

void **EndUpdate**

EndUpdate must be called after completing changes to the model that were begun with a call to the BeginUpdate method. When BeginUpdate is matched by a subsequent call to EndUpdate, the model updates to reflect all changes that occurred.

long	ExportToDXF ([in] BSTR FileName, [in] EDXFVersion DXFVersion, [in] ELengthUnit LengthUnit, [in] double FontFactor)
	<p>FileName file name DXFVersion export format LengthUnit determines the length unit of the DXF coordinates FontFactor font size control factor (FontFactor = 1 means that the default exported font size is used)</p>
	<i>If successful it returns 1, otherwise it returns error (meDXFmoduleNotAvailable, meCannotExport, errDatabaseNotReady).</i>
long	ExportToIFC ([in] BSTR FileName, [in] EIFCVersion IFCVersion, [in] ELongBoolean ArchitectModel, [in] ELongBoolean SelectedOnly, [in] ELongBoolean ExportReinforcement, [in] RIFCExportReinforcementParams ExportReinforcement)
	<p>FileName file name IFCVersion export format ArchitectModel if True only model geometry is exported if False model details (domains, supports, loads, load combinations) are also exported SelectedOnly if True and the selection is not empty only the selected elements are exported, otherwise the entire model is exported ExportReinforcement if True than the reinforcement is also exported, only for IFC version 2x3 ReinforcementParams reinforcement parameters</p>
	<i>If successful it returns 1, otherwise it returns error (meIFCmoduleNotAvailable, meCannotExport, errDatabaseNotReady).</i>
long	ExportToPIA ([in] BSTR FileName, [in] ELongBoolean SelectedOnly)
	<p>FileName file name SelectedOnly if True and the selection is not empty only the selected elements are exported, otherwise the entire model is exported</p>
	<i>Exports the model or a selection to a PIA (PianoCA interface) file. Possible error codes are meCannotExport, errDatabaseNotReady.</i>
<hr/>	
	FitInView
	<i>Scales the model drawing so that it fits in the window.</i>
long	GetAPIGlobalData ([in] BSTR DataName, [out] SAFEARRAY (byte) * APIGlobalData)
	<p>DataName name of the saved API data, recommended to use also the API name (max 32 chars) APIGlobalData custom API data</p>
	<i>Read custom data saved to axs files saved under name DataName. If successful it returns size of saved data, otherwise it returns error (meCannotReadGlobalData, meGlobalDataNotFound, meInvalidDataName, errDatabaseNotReady).</i>
long	GetAPIGlobalAxisSize ([in] BSTR DataName)
	<p>DataName name of the saved API data, recommended to use also the API name (max 32 chars)</p>
	<i>Import DXF file. If successful it returns 1, otherwise an error code.</i>

long	GetCompanyLogoParameters ([i/o] RCompanyLogoParameters Parameters , [out] BSTR LogoFileName)
	<p>Parameters parameters of company's logo LogoFileName logo's filename</p> <p>Get filename and parameters of the company's logo. If succesfull it retruns 1.0, otherwise it returns error (e.g. errDatabaseNotReady).</p> <p>Note: Supported extensions for LogoFileName are: *.png; *.jpg; *.jpeg; *.bmp; *.tif; *.tiff; *.ico; *.emf; *.wmf.</p> <p>For further details see the picture by the description of RCompanyLogoParameters.</p>
long	SetCompanyLogoParameters ([i/o] RCompanyLogoParameters Parameters , [in] BSTR LogoFileName)
	<p>Parameters parameters of company's logo LogoFileName logo's filename</p> <p>Set filename and parameters of the company's logo. If succesfull it retruns 1.0, otherwise it returns error (e.g. errDatabaseNotReady).</p> <p>Note: Supported extensions for LogoFileName are: *.png; *.jpg; *.jpeg; *.bmp; *.tif; *.tiff; *.ico; *.emf; *.wmf.</p> <p>For further details see the picture by the description of RCompanyLogoParameters.</p>
long	GetIFCExportReinfParams ([i/o] RIFCExportReinforcementParams Value)
	<p>Value IFC reinforcement export parameters</p> <p>Get actual reinforcement parametrs for IFC export. If successful it returns 1, otherwise an error code.</p>
long	GetSeismicParams ([i/o] RSeismicParams Value)
	<p>Warning! This function has become obsolete, was superseded by GetSeismicParams_V153</p> <p>Duplicate of function GetSeismicParams. Retained for compatibility but will be removed in later version.</p>
long	ImportDXF ([in] BSTR FileName , [i/o] RDXFimportParameters Parameters)
	<p>FileName file name Parameters import parameters</p> <p>Import DXF file. If successful it returns 1, otherwise an error code.</p>
long	ImportIFC ([in] BSTR FileName , [i/o] RIFCimportParameters Parameters)
	<p>FileName file name Parameters import parameters</p> <p>Import IFC file. If successful it returns 1, otherwise an error code.</p>
long	ImportRAE ([in] BSTR raeFileName , [in] BSTR jsonFileName)
	<p>raeFileName file name jsonFileName file name of json string</p> <p>Import RAE file. If successful it returns 1, otherwise an error code (errDatabaseNotReady, meRevitImportTessDegreeOutOfRange, errJSONpropertyMissing).</p>
long	ImportPDF ([in] BSTR FileName , [i/o] RPDFimportParameters Parameters)
	<p>FileName file name Parameters import parameters</p> <p>Import PDF file. If successful it returns 1, otherwise an error code.</p>
long	ImportAXS ([in] BSTR FileName , [i/o] RAXSimportParameters Parameters)
	<p>FileName file name Parameters import parameters</p> <p>Import AXS file. If successful it returns 1, otherwise an error code.</p>

ELongBoolean	LoadFromFile ([in] BSTR FileName) FileName file name Loads the model from a file. If successful, returns True, otherwise False.
	Redo Reverses the undo or advances the buffer to a more current state of the model when SaveUndo function was called.
long	Refresh Refresh all windows.
long	SelectAll ([in] ELongBoolean Select) Select select all If Select is True, selects all actual components. If Select is False, deselects all actual components. If successful, returns the number of selected actual components, otherwise returns an error code(errDatabaseNotReady) Note: Call Refresh function afterwards if not called between functions BeginUpdate and EndUpdate
long	SetAPIGlobalData ([in] BSTR DataName, [in] SAFEARRAY (byte) * APIGlobalData) DataName name of the saved API data, recommended to use also the API name (max 32 chars) APIGlobalData custom API data Write custom data to the axs file saved under name DataName. If successful it returns size of saved data, otherwise it returns error (meCannotSaveGlobalData , meInvalidOrEmptyGlobalData , meDataNameAlreadyExists , meInvalidDataName , errDatabaseNotReady).
long	SetAPIGlobalData_vb ([in] BSTR DataName, [i/o] SAFEARRAY (byte) * APIGlobalData) Visual Basic compatible version of SetAPIGlobalData .
void	SaveModelBeforeSave The save icon in AxisVM will be enabled and confirmation dialog box about saving the model will be displayed when closing AxisVM.
ELongBoolean	SaveToFile ([in] BSTR FileName, [in] ELongBoolean 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.
	SaveUndo ([in] BSTR UndoText) UndoText Description of the undo step Saves the model for undo with undo step description
ELongBoolean	StartSelection ([in] BSTR SelectionMessage, [in] ELongBoolean DeleteCurrentSelection, [in] BSTR ListOfSelectionTypes) SelectionMessage message to display in the status line during the selection process DeleteCurrentSelection determines if the current selection is removed before selection process ListOfSelectionTypes a string of double-byte characters encoding ESelectionType constants Displays the selection toolbar and lets the user make a selection. Allowed selection types can be set by constructing a double-byte Unicode string of characters encoding ESelectionType constants (0x01-0x22).

ELongBoolean	StartModalSelection ([in] BSTR SelectionMessage , [in] ELongBoolean DeleteCurrentSelection , [in] BSTR ListOfTypeSelectionTypes)
	SelectionMessage <i>message to display in the status line during the selection process</i>
	DeleteCurrentSelection <i>determines if the current selection is removed before selection process</i>
	ListOfTypeSelectionTypes <i>a string of double-byte characters encoding ESelectionType constants</i>
	<i>Displays the selection toolbar and lets the user make a selection. It does not let your program to complete other actions until the end of selection procedure. Allowed selection types can be set by constructing a double-byte Unicode string of characters encoding ESelectionType constants (0x01-0x22).</i>
long	SetSeismicParams ([i/o] RSeismicParams Value)
	Warning! This function has become obsolete, was superseded by SetSeismicParams_V153
	<i>Duplicate of function SetSeismicParams. Retained for compatibility but will be removed in later version.</i>
	Undo
	<i>Revert to an older state of the model when SaveUndo function was called</i>

Properties

IMPORTANT NOTE:

Properties of interface type, should be called only once. Hence, they create the object of that type and only one instance of that object should be created.

AxisVMActualReinforcement *	ActualReinforcement <i>Access to the Actual Reinforcement of surfaces(domains) (extension module RC1 is required)</i>
BSTR*	AnalysisBy <i>Get or set name of the project designer (engineer).</i>
AxisVMCalculation *	Calculation <i>Access to the solver of AxisVM.</i>
ElongBoolean	CallProgress • If set to <i>lbTrue</i> , any progress bar movements on main AxisVM form will call COM event IAxisVMMODELEvents.MainProgress <i>Read and write property.</i>
AxisVMColumnRebars *	ColumnRebars <i>Access to the column rebars in the RC column (extension module RC2 is required)</i>
BSTR*	Comment <i>Get or set comment about (description of) the current model (project).</i>
AxisVMCriticalGroupCombinations *	CriticalGroupCombinations <i>Access to the critical group combinations of the model.</i>
AxisVMCrossSections *	CrossSections <i>Access to the cross-sections of the model.</i>
AxisVMCustomParts *	CustomParts <i>Access to the custom parts of the model.</i>
AxisVMDiaphragm *	Diaphragm <i>Access to the diaphragms of the model</i>
AxisVMDimensions *	Dimensions <i>Access the dimensions</i>

EDisplay	Display • Get or set current display mode of the model.
AxisVMDomains*	Domains Access to the domains of the model.
AxisVMDomainSupports*	DomainSupports Access to the domain supports of the model
AxisVMDomainsSupports*	DomainsSupports Access to the domain supports of the model.
AxisVMDrawingsLibrary*	DrawingsLibrary Access to the library of drawings (Drawings library) of the model
AxisVMDynamicLoadFunctions*	DynamicLoadFunctions Access to the dynamic load functions of the model (extension module DYN is required)
AxisVMEdgeConnections*	EdgeConnections Access to the edge connections of the model
AxisVMEvelopes*	Envelopes Access to the envelopes (of load cases and combinations) of the model
BSTR	FileName Name of the opened model. Read only property.
AxisVMIncrementFunctions*	IncrementFunctions Access to the increment functions of the model
AxisVMLayers*	Layers Access to the layers of the model
AxisVMLines*	Lines Access to the lines of the model.
AxisVMLineSupports*	LineSupports Access to the line supports of the model.
AxisVMLinkElements*	LinkElements Access to the link elements of the model
AxisVMLoadCases*	LoadCases Access to the load cases of the model. Set and get parameters for seismic, pushover, imperfections, wind and snow loads.
AxisVMLoadCombinations*	LoadCombinations Access to the load combinations of the model.
AxisVMLoadGroups*	LoadGroups Access to the load groups of the model.
AxisVMLoadPanels*	LoadPanels Access to the load panels of the model. Used for wind and snow load generation (extension module SWG is required)
AxisVMLoads*	Loads Access to the loads of the model.
AxisVMMaterials*	Materials Access to the materials of the model.
AxisVMMathTexts*	MathTexts Access to the math texts of the model.
ELongBoolean	Modified True if model was changed since last save. Read only property.
AxisVMMovingLoads*	MovingLoads Access to the moving loads of the model
AxisVMMembers*	Members Access to the structural members of the model.
AxisVMMembersSupports*	MembersSupports Access to the member supports of the model.

EBoolean	NeedsSaving <i>True, if the model has been modified so it needs saving. Read only property.</i>
AxisVMNodalSupports*	NodalSupports <i>Access to the node supports of the model.</i>
AxisVMNodesSupports*	NodesSupports <i>Access to the nodal supports of the model.</i>
AxisVMNodes*	Nodes <i>Access to the nodes of the model.</i>
EAxisVMPlatform	Platform <i>Get whether it is 32 or 64 bit version of AxisVM</i>
BSTR*	ProjectName <i>Name (title) of the current project. Read and write property.</i>
AxisVMPushoverHingeFunctions*	PushoverHingeFunctions <i>Access to the hinge functions for pushover (extension module SE2 is required)</i>
AxisVMRCBeamDesign*	RCBeamDesign <i>Access to the RC beam design (extension module RC2 is required)</i>
AxisVMRCColumnChecking*	RCColumnChecking <i>Access to the RC column checking (extension module RC2 is required)</i>
AxisVMReferences*	References <i>Access to the references of the model.</i>
AxisVMReports*	Reports <i>Access to the reports of the model.</i>
AxisVMResults*	Results <i>Access to the results of the model.</i>
AxisVMRebarSteelGrades*	RebarSteelGrades <i>Access to the rebar steel grades</i>
AxisVMRigidBodies*	RigidBodies <i>Access to the rigid bodies of the model.</i>
EBoolean	SelectionProcessing <i>True, if AxisVM waits for the user to complete the selection. Read only property.</i>
EBoolean	SelectionResult <i>True, if the user clicked the OK button of the selection toolbar. Read only property.</i>
AxisVMSettings*	Settings - Get model settings.
AxisVMSpectrum*	SpectrumH Warning! This property has become obsolete, was superseded by SeismicSpectrumH <i>Access to the horizontal spectrum of the first seismic group</i>
AxisVMSpectrum*	SpectrumV Warning! This property has become obsolete, was superseded by SeismicSpectrumV <i>Access to the vertical spectrum of the first seismic group</i>
AxisVMSpectrum*	SpectrumPushOver <i>Access to the pushover spectrum of the model</i>
AxisVMSeismicStoreys*	SeismicStoreys <i>Access to the seismic storeys of the model</i>
AxisVMCrossSectionOptimization*	SteelCrossSectionOptimization <i>Access to the steel cross-section optimization (extension module SD9 is required)</i>
AxisVMSteelDesignMembers*	SteelDesignMembers

	<i>Access to the steel design (extension module SD1 is required)</i>
<u>AxisVMStructuralGrids</u> *	StructuralGrids <i>Access to structural grids</i>
<u>AxisVMSurfaces</u> *	Surfaces <i>Access to the surface elements of the model.</i>
<u>AxisVMSurfaceSupports</u> *	SurfaceSupports <i>Access to the surface supports of the model.</i>
<u>AxisVMSections</u> *	Sections <i>Access to the sections of the model.</i>
<u>AxisVMStoreys</u> *	Storeys <i>Access to storeys of the model.</i>
<u>AxisVMTasks</u> *	Task <i>Run various tasks in AxisVM</i>
<u>AxisVMCrossSectionOptimization</u> *	TimberCrossSectionOptimization <i>Access to the timber cross-section optimization (extension module TD9 is required)</i>
<u>AxisVMTimberDesignMembers</u> *	TimberDesignMembers <i>Access to timber design (extension module TD1 is required)</i>
<u>AxisVMTimeIncrementFunctions</u> *	TimeIncrementFunctions <i>Access to time increment functions of the model (extension module DYN is required)</i>
<u>EView</u>	View • <i>Get or set view of the active window also available in <u>AxisVMWindows</u> and <u>AxisVMWindow</u> interface</i>
<u>AxisVMVirtualBeams</u> *	VirtualBeams <i>Access to virtual beams/strips of the model.</i>
<u>AxisVMWindows</u> *	Windows <i>Access to all windows, their view settings and displayed load case.</i>
<u>AxisVMWorkplanes</u> *	Workplanes <i>Access to workplanes of the model.</i>
<u>AxisVMXLAMpanels</u> *	XLAMpanels <i>Access to XLAM panels of the model.</i>

IAxisVMActualReinforcement

Interface for defining actual reinforcement on surface elements and domains.

If property returning this interface is null (nil) then the extension module RC1 is not available.

Calculated reinforcement can be accessed in interface [IAxisVMCalculatedReinforcement](#) in IAxisVMResults.

Important: Reinforcement design parameters must be set in interfaces [IAxisVMDomain](#) or [IAxisVMSurface](#) prior to defining the actual reinforcement.

Note: If skew reinforcement is set for the domain or surface using RReinforcementParameters, x and y reinforcement directions are interpreted as ξ and η reinforcement directions, respectively.

Enumerated types

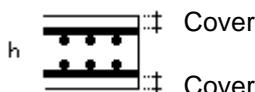
```
enum EActualReinforcementType = {  
    artDomain = 0  
    artPolygon = 1}  
    Type of reinforcement contour
```

Error codes

enum EActualReinforcementError = {	
areCOMError = -100001	COM server error
areDomainIdOutOfBounds = -100002	<i>IAxisVMActualReinforcement.AddDomainReinforcement</i> or <i>IAxisVMActualReinforcement.IndexOfDomainReinforcement</i> can return this error when the DomainId parameter is invalid
areErrorAddingPolygonReinforcement = -100003	<i>IAxisVMActualReinforcement.AddPolygonReinforcement</i> can return this when reinforcement parameters are missing
areDomainReinforcementNotFound = -100004	<i>IAxisVMActualReinforcement.IndexOfDomainReinforcement</i> could not find reinforcement for the domain
areSurfaceIdOutOfBounds = -100005	SurfaceId is invalid.
areSurfaceVertexIndexOutOfBounds = -100006 }	SurfaceVertexIndex is invalid.

Records / structures

	RActualReinforcement = (
double	ds	Rebar diameter [m]
double	spacing	Spacing of rebar [m]
ERebarType	RebarType	Type of the rebar
double	Cover	Cover of the rebar to the edge of the domain/surface [m]
double	Alpha	Only for MSZ (Hungarian design code)
)	



Functions

```
long AddDomainReinforcement ([in] long DomainId, [in] SAFEARRAY(RActualReinforcement)*  
X_Bottom, [in] SAFEARRAY(RActualReinforcement)* X_Top, [in]  
SAFEARRAY(RActualReinforcement)* Y_Bottom, [in] SAFEARRAY(RActualReinforcement)*  
Y_Top)
```

DomainId	index of domain
X_Bottom	Bottom reinforcement in x (or ξ) direction. Lower index of the array must be 1 and it can have more elements (same amount as reinforcement layers).
X_Top	Top reinforcement in x (or ξ) direction. Lower index of the array must be 1 and it can have more elements (same amount as reinforcement layers).
Y_Bottom	Bottom reinforcement in y (or η) direction. Lower index of the array must be 1 and it can have more elements (same amount as reinforcement layers).
Y_Top	Top reinforcement in y (or η) direction. Lower index of the array must be 1 and it can have more elements (same amount as reinforcement layers).

Returns ActualReinforcementID, otherwise returns an error code ([errDatabaseNotReady](#), [areDomainIdOutOfBounds](#)).

long	AddDomainReinforcement_vb (Visual Basic compatible function of AddDomainReinforcement)
long	AddPolygonReinforcement ([in] IAxisVMLines3d* Polygon , [in] SAFEARRAY(RActualReinforcement)* X_Bottom , [in] SAFEARRAY(RActualReinforcement)* X_Top , [in] SAFEARRAY(RActualReinforcement)* Y_Bottom , [in] SAFEARRAY(RActualReinforcement)* Y_Top)
	Polygon IAxisVMLines3d object. Polygon must be closed (endpoint of the last line must be same as start point of the first line)
	X_Bottom Bottom reinforcement in x (or ξ) direction. Lower index of the array must be 1 and it can have more elements (same amount as reinforcement layers).
	X_Top Top reinforcement in x (or ξ) direction. Lower index of the array must be 1 and it can have more elements (same amount as reinforcement layers).
	Y_Bottom Bottom reinforcement in y (or η) direction. Lower index of the array must be 1 and it can have more elements (same amount as reinforcement layers).
	Y_Top Top reinforcement in y (or η) direction. Lower index of the array must be 1 and it can have more elements (same amount as reinforcement layers).
	Returns ActualReinforcementID, otherwise returns an error code (errDatabaseNotReady , areErrorAddingPolygonReinforcement).
long	AddPolygonReinforcement_vb (Visual Basic compatible function of AddPolygonReinforcement)
long	Clear Returns number of deleted actual reinforcement, otherwise returns an error code (errDatabaseNotReady)
long	Delete ([in] long Index) Index index of the actual reinforcement, $1 \leq Index \leq Count$ If successful returns ActualReinforcementID, otherwise returns an error code (errDatabaseNotReady , errIndexOutOfBoundsException)
long	DeleteSelected Returns number of deleted actual reinforcement, otherwise returns an error code(errDatabaseNotReady)
long	GetSelectedItemIds ([out] SAFEARRAY (long) * ItemIds) ItemIds Index list of selected actual reinforcement Returns the number of selected actual reinforcement, otherwise returns an error code(errDatabaseNotReady)

long **GetReinforcement** ([in] long **Index**, [out] [EActualReinforcementType](#)* **ActualReinforcementType**, [out] long* **DomainId**, [out] [IAxisVMLines3d](#)* **Polygon**, [out] [SAFEARRAY\(RActualReinforcement\)](#)* **X_Bottom**, [out] [SAFEARRAY\(RActualReinforcement\)](#)* **X_Top**, [out] [SAFEARRAY\(RActualReinforcement\)](#)* **Y_Bottom**, [out] [SAFEARRAY\(RActualReinforcement\)](#)* **Y_Top**)

Index index of the actual reinforcement, $1 \leq Index \leq Count$

ActualReinforcementType artDomain: Polygon variable won't be filled ; artPolygon:

DomainId DomainId variable won't be filled

Polygon [IAxisVMLines3d](#) object

X_Bottom Bottom reinforcement in x (or ξ) direction. Lower index of the array must be 1 and it can have more elements (same amount as reinforcement layers).

X_Top Top reinforcement in x (or ξ) direction. Lower index of the array must be 1 and it can have more elements (same amount as reinforcement layers).

Y_Bottom Bottom reinforcement in y (or η) direction. Lower index of the array must be 1 and it can have more elements (same amount as reinforcement layers).

Y_Top Top reinforcement in y (or η) direction. Lower index of the array must be 1 and it can have more elements (same amount as reinforcement layers).

If successful returns Index, otherwise returns an error code ([errInternalException](#)).

long **GetSurfaceReinforcement** ([in] long **Surfaceld**, [in] [ESurfaceVertexIndex](#) **SurfaceVertexIndex**, [out] [SAFEARRAY\(RActualReinforcement\)](#)* **X_Bottom**, [out] [SAFEARRAY\(RActualReinforcement\)](#)* **X_Top**, [out] [SAFEARRAY\(RActualReinforcement\)](#)* **Y_Bottom**, [out] [SAFEARRAY\(RActualReinforcement\)](#)* **Y_Top**)

Surfaceld index of the surface, $1 \leq Index \leq Count$

SurfaceVertexIndex artDomain: Polygon variable won't be filled ; artPolygon: DomainId variable won't be filled

X_Bottom Bottom reinforcement in x (or ξ) direction. Lower index of the array must be 1 and it can have more elements (same amount as reinforcement layers).

X_Top Top reinforcement in x (or ξ) direction. Lower index of the array must be 1 and it can have more elements (same amount as reinforcement layers).

Y_Bottom Bottom reinforcement in y (or η) direction. Lower index of the array must be 1 and it can have more elements (same amount as reinforcement layers).

Y_Top Top reinforcement in y (or η) direction. Lower index of the array must be 1 and it can have more elements (same amount as reinforcement layers).

If successful returns Surfaceld, otherwise returns an error code ([areSurfaceldOutOfBounds](#)).

long **GetSurfaceReinforcements** ([in] long **Surfaceld**, [out] SAFEARRAY(VARIANT)* **X_Bottom**,
[out] SAFEARRAY(VARIANT)* **X_Top**, [out] SAFEARRAY(VARIANT)* **Y_Bottom**, [out]
SAFEARRAY(VARIANT)* **Y_Top**)

Surfaceld index of the surface, $1 \leq Index \leq Count$
X_Bottom Bottom reinforcement in x (or ξ) direction. See note below.
X_Top Top reinforcement in x (or ξ) direction. See note below.
Y_Bottom Bottom reinforcement in y (or η) direction. See note below.
Y_Top Top reinforcement in y (or η) direction. See note below.

If successful returns **Surfaceld**, otherwise returns an error code ([areSurfaceldOutOfBounds](#)).

NOTE:

VARIANTs are VarArrays/SafeArrays/ so be aware of destroying them by calling VariantClear!
So the out parameters are SAFEARRAY(SAFEARRAY(RActualReinforcement)).

EXAMPLE:

Dim id As long

Dim Xbot() As Variant

Dim Xtop() As Variant

Dim Ybot() As Variant

Dim Ytop() As Variant

nResult = AxActualReinf.GetSurfaceReinforcements(id, Xbot, Xtop, Ybot, Ytop)

Xbot, Xtop, Ybot, Ytop are two dimensional arrays with $9 \times N$ elements for surface of quad mesh and surface of triangular mesh will have also $9 \times N$ elements, but element No.5 and 9 will be empty or null.

N is number of actual reinforcement layers in one direction (usually 1)

Each quad surface element has 4 corner and 4 mid-line values and 1 central value (total 9No. RActualReinforcement records)

long **IndexOfDomainReinforcement** ([in] long **DomainId**)

DomainId Index of domain

Returns ActualReinforcementID, otherwise returns an error code
([areDomainReinforcementNotFound](#), [areDomainIdOutOfBounds](#))

long **SelectAll** ([in] [EBoolean](#) **Select**)

Select selection state

If Select is True, selects all actual reinforcement.

If Select is False, deselects all actual reinforcement.

If successful, returns the number of selected actual reinforcement, otherwise returns an error code([errDatabaseNotReady](#))

Note: Call [Refresh](#) function afterwards if not called between functions [BeginUpdate](#) and [EndUpdate](#)

Properties

long **Count**

Get number of actual reinforcement in the model

[EBoolean](#) **Selected** [long **Index**] ***

Index index of the actual reinforcement, $1 \leq Index \leq Count$

Get or set the selection status of the actual reinforcement

*** Call [Refresh](#) function afterwards if not called between functions

[BeginUpdate](#) and [EndUpdate](#)

long **SelCount**

Get number of selected actual reinforcements in the model

[EActualReinforcementType](#) **ActualReinforcementType** [long **Index**]

Index index of the actual reinforcement, $1 \leq Index \leq Count$

IAxisVMAAttachments

Attachments are custom data with various size saved to elements (nodes, lines, members, etc.) with index.

Error codes

enum	EAttachmentsError = {	
	attaeCannotAddAttachment = -100001	<i>Internal error during adding the attachment data</i>
	attaeCannotGetAttachment = -100002	<i>Internal error during reading attachment data</i>
	attaeInvalidName = -100003	<i>Length of attachment name data must be less than 32 chars</i>
	attaeInvalidOrEmptyItemData = -100004	<i>Attachment data (buffer) is empty or not valid</i>
	atteAttributeNotFound = -100005	<i>Attachment data not found</i>
	attaeItemIndexOutOfBounds = -100006	<i>Item index is not valid</i>
	attaeCannotDeleteAttachment = -100007	<i>Internal error during deleting attachment data</i>
	attaeItemHasNoAttachment = -100008	<i>Internal error during seting the attachment data</i>
	attaeInvalidOrEmptyItemIndexes = -100009 }	<i>Array with item indexes is empty or not valid</i>

Functions

long	AddData ([in] BSTR Name, [in] long ItemIndex, [in] SAFEARRAY (byte) * ItemData)	
	Name	<i>Name of the saved attachment data, recommended to include also the API name (max. 32 chars) e.g "attachment A"</i>
	ItemIndex	<i>Index of the element/item of the parent interface (Nodes, lines, etc.)</i>
	ItemData	<i>attachment data, this data will be assigned to an item/elemnt with index ItemIndex</i>
		<i>Add attachment data to an element with index of the parent interface (Node index, Line index, etc.). If successful it returns attachment index, otherwise it returns error.</i>
long	AddData_vb ([in] BSTR Name, [in] long ItemIndex, [i/o] SAFEARRAY (byte) * ItemData)	
		<i>Visual Basic compatible function of AddData.</i>
long	GetData ([in] BSTR Name, [in] long ItemIndex, [out] SAFEARRAY (byte) * ItemData)	
	Name	<i>Name of the saved attachment data</i>
	ItemIndex	<i>Index of the element/item of the parent interface (Nodes, lines, etc.)</i>
	ItemData	<i>attachment data</i>
		<i>Get attachment data from the axs file. If successful it returns attachment size, otherwise it returns error.</i>
long	GetSize ([in] BSTR Name, [in] long ItemIndex)	
	Name	<i>Name of the saved attachment data</i>
	ItemIndex	<i>Index of the element/item of the parent interface (Nodes, lines, etc.)</i>
		<i>Get size of attachment data from the axs file. If successful it returns attachment data size, otherwise it returns error.</i>
long	Delete ([in] BSTR Name, [in] long ItemIndex)	
	Name	<i>Name of the saved attachment data</i>
	ItemIndex	<i>Index of the element/item of the parent interface (Nodes, lines, etc.)</i>
		<i>Delete attachment data from the axs file. If successful it returns ItemIndex, otherwise it returns error.</i>

Properties

long	ItemCount Get number of attachments of the parent interface (IAxisVMNodes, IAxisVMLines, etc.)
------	--

IAxisVMAtributes

Attributes are custom fixed size data saved to each elements (nodes, lines, members, etc.) Length of data depends on length of ItemData safearray added with AddDefault function.

Default attribute data is assigned to all items and this default data will be also used for all newly created elements of the parent interface.

Error codes

enum	EAttributesError = {	
	atteCannotAddAttribute = -100001	<i>Internal error during adding the attribute data</i>
	atteCannotGetAttribute = -100002	<i>Internal error during reading attribute data</i>
	atteInvalidName = -100003	<i>Length of attribute name data must be less than 32 chars</i>
	atteInvalidOrEmptyItemData = -100004	<i>Attribute data (buffer) is empty or not valid</i>
	atteAttributeNotFound = -100005	<i>Attribute data not found</i>
	atteNameAlreadyExists = -100006	<i>Attribute with same name already exists</i>
	atteCannotDeleteAttribute = -100007	<i>Internal error during deleting attribute data</i>
	atteCannotSetAttribute = -100008	<i>Internal error during setting the attribute data</i>
	atteInvalidOrEmptyItemIndexes = -100009	<i>Array with item indexes is empty or not valid</i>
	atteItemIndexOutOfBounds = -100010	<i>Index in array with item indexes is out of bounds</i>
	atteAttributeSizeMustMatch = -100011	<i>Size of new and default attribute data must match</i>
	atteItemsDataMustContainAllItems = -100012 }	<i>Attribute data must contain data of all items (n * attribute size)</i>

Functions

long	AddDefault ([in] BSTR Name, [in] SAFEARRAY (byte) * ItemData)	
	Name	<i>Attribute name of the saved attribute data, recommended to include also the API name (max. 32 chars) e.g "AttributeA"</i>
	ItemData	<i>default attribute data, this data will be assigned to all items and also to new ones created from now on.</i>
		<i>Add default attribute data to all new elements of the parent interface (Nodes, Lines, etc.). This default attribute data will be also used for all new items created from now on. If successful it returns attribute index, otherwise it returns error (atteCannotAddAttribute, atteInvalidOrEmptyItemData, atteNameAlreadyExists, atteInvalidName errDatabaseNotReady).</i>
long	AddDefault_vb ([in] BSTR Name, [i/o] SAFEARRAY (byte) * ItemData)	Visual Basic compatible function of AddDefault.
long	DeleteByName ([in] BSTR Name)	
	Name	<i>Attribute name of the saved attribute data</i>
		<i>Delete attribute data by name. If successful it returns 1, otherwise it returns error (atteCannotDeleteAttribute, atteAttributeNotFound, atteInvalidName errDatabaseNotReady).</i>
long	DeleteByIndex ([in] long AttributeIndex)	
	AttributeIndex	<i>Attribute index of the saved attribute data</i>
		<i>Delete attribute data by attribute index. If successful it returns 1, otherwise it returns error (atteCannotDeleteAttribute, atteAttributeNotFound, errIndexOutOfBoundsException, errDatabaseNotReady).</i>
long	GetDefault ([in] BSTR Name, [out] SAFEARRAY (byte) * ItemData)	
	Name	<i>Attribute name of the saved attribute data</i>
	ItemData	<i>default attribute data</i>
		<i>Get default attribute data from the axs file. If successful it returns attribute size, otherwise it returns error (atteAttributeNotFound, errIndexOutOfBoundsException, errDatabaseNotReady).</i>
long	GetName ([in] long AttributeIndex , [out] BSTR Name)	
	AttributeIndex	<i>Attribute index of the saved attribute data</i>
	Name	<i>Attribute name of the saved attribute data</i>
		<i>Get attribute name. If successful it returns attribute size, otherwise it returns error (atteAttributeNotFound, errIndexOutOfBoundsException, errDatabaseNotReady).</i>

long	GetItemByIndex ([in] long AttributeIndex , [in] long ItemIndex , [out] SAFEARRAY (byte) * ItemData)
	<p>AttributeIndex Attribute index of the saved attribute data ItemIndex item (element) index , 1 to ParentInterface.count ItemData custom attribute data of the item</p> <p>Get attribute data of the item (element) from the axs file. If successful it returns attribute size, otherwise it returns error (atteCannotGetAttribute, atteAttributeNotFound, errIndexOutOfBounds, errDatabaseNotReady).</p>
long	GetItemByName ([in] BSTR Name , [in] long ItemIndex , [out] SAFEARRAY (byte) * ItemData)
	<p>Name Attribute name of the saved attribute data ItemIndex item (element) index , 1 to ParentInterface.count ItemData custom attribute data of the item</p> <p>Get attribute data of the item (element) from the axs file. If successful it returns attribute size, otherwise it returns error (atteCannotGetAttribute, atteAttributeNotFound, atteInvalidName, errDatabaseNotReady).</p>
long	GetSizeByName ([in] BSTR Name)
	<p>Name Attribute name of the saved attribute data</p> <p>Get attribute data size by name. If successful it returns attribute size, otherwise it returns error (atteAttributeNotFound, atteInvalidName, errDatabaseNotReady).</p>
long	GetSizeByIndex ([in] long AttributeIndex)
	<p>AttributeIndex Attribute index of the saved attribute data</p> <p>Get attribute data size by index. If successful it returns attribute size, otherwise it returns error (atteAttributeNotFound, errIndexOutOfBounds, errDatabaseNotReady).</p>
long	FillItemByIndex ([in] long AttributeIndex , [in] long ItemIndex , [in] SAFEARRAY (byte) * ItemData)
	<p>AttributeIndex Attribute index of the saved attribute data ItemIndex item (element) index , 1 to ParentInterface.count ItemData custom attribute data, length must match length of default attribute data</p> <p>Owerwrite attribute data of one item (element) with new data. If successful it returns attribute index, otherwise it returns error (atteCannotSetAttribute, atteInvalidOrEmptyItemData, atteAttributeSizeMustMatch, attItemIndexOutOfBounds, errIndexOutOfBounds, errDatabaseNotReady).</p>
long	FillItemByName ([in] BSTR Name , [in] long ItemIndex , [in] SAFEARRAY (byte) * ItemData)
	<p>Name Attribute name of the saved attribute data ItemIndex item (element) index , 1 to ParentInterface.count ItemData custom attribute data, length must match length of default attribute data</p> <p>Owerwrite attribute data of one item (element) with new data. If successful it returns attribute index, otherwise it returns error (atteCannotSetAttribute, atteInvalidOrEmptyItemData, atteAttributeSizeMustMatch, attItemIndexOutOfBounds, atteAttributeNotFound, atteInvalidName, errDatabaseNotReady).</p>
long	FillItemByName_vb ([in] BSTR Name , [in] long ItemIndex , [i/o] SAFEARRAY (byte) * ItemData)
	Visual Basic compatible function of FillItemByName.
long	FillAllItemsByIndex ([in] long AttributeIndex , [in] SAFEARRAY (byte) * ItemData)
	<p>AttributeIndex Attribute index of the saved attribute data ItemData custom attribute data, length must match length of default attribute data</p> <p>Owerwrite attribute data of all items (elements) with new data. If successful it returns attribute index, otherwise it returns error (atteCannotSetAttribute, atteInvalidOrEmptyItemData, atteAttributeSizeMustMatch, errIndexOutOfBounds, errDatabaseNotReady).</p>

long	FillAllItemsByIndex_vb ([in] long AttributeIndex , [i/o] SAFEARRAY (byte) * ItemData) Visual Basic compatible function of FillAllItemsByIndex.
long	FillAllItemsByName ([in] BSTR Name , [in] SAFEARRAY (byte) * ItemData) Name Attribute name of the saved attribute data ItemData custom attribute data, length must match length of default attribute data Overwrite attribute data of all items (elements) with new data. If successful it returns attribute index, otherwise it returns error (atteCannotSetAttribute , atteInvalidOrEmptyItemData , atteAttributeSizeMustMatch , atteAttributeNotFound , atteInvalidName , errDatabaseNotReady).
long	FillAllItemsByName_vb ([in] BSTR Name , [i/o] SAFEARRAY (byte) * ItemData) Visual Basic compatible function of FillAllItemsByName.
long	FillItemsByIndex ([in] long AttributeIndex , [in] SAFEARRAY (long) * ItemIndexes , [in] SAFEARRAY (byte) * ItemData) AttributeIndex Attribute index of the saved attribute data ItemIndexes Array of item (element) indexes, item index = 1 to ParentInterface.count ItemData custom attribute data, length must match length of default attribute data Overwrite attribute data of items (elements) in ItemIndexes array with new data. If successful it returns attribute index, otherwise it returns error (atteCannotSetAttribute , atteInvalidOrEmptyItemData , atteAttributeSizeMustMatch , atteItemIndexOutOfBounds , errIndexOutOfBounds , errDatabaseNotReady).
long	FillItemsByIndex_vb ([in] long AttributeIndex , [i/o] SAFEARRAY (long) * ItemIndexes , [in] SAFEARRAY (byte) * ItemData) Visual Basic compatible function of FillItemsByIndex.
long	FillItemsByName ([in] BSTR Name , [in] SAFEARRAY (long) * ItemIndexes , [in] SAFEARRAY (byte) * ItemData) Name Attribute name of the saved attribute data ItemIndexes Array of item (element) indexes, item index = 1 to ParentInterface.count ItemData custom attribute data, length must match length of default attribute data Overwrite attribute data of items (elements) in ItemIndexes array with new data. If successful it returns attribute index, otherwise it returns error (atteCannotSetAttribute , atteInvalidOrEmptyItemData , atteAttributeSizeMustMatch , atteItemIndexOutOfBounds , atteAttributeNotFound , atteInvalidName , errIndexOutOfBounds , errDatabaseNotReady).
long	FillItemsByName_vb ([in] BSTR Name , [i/o] SAFEARRAY (long) * ItemIndexes , [in] SAFEARRAY (byte) * ItemData) Visual Basic compatible function of FillItemsByName.
long	IndexOf ([in] BSTR Name) Name Attribute name of the saved attribute data Get attribute index by name. If successful it returns attribute size, otherwise it returns error (atteAttributeNotFound , atteInvalidName , errDatabaseNotReady).
<u>ElongBoolean</u>	IsDefaultItemByIndex ([in] long AttributeIndex , [in] long ItemIndex) AttributeIndex Attribute index of the saved attribute data ItemIndex item (element) index , 1 to ParentInterface.count If returns IbTrue then it is the default attribute.
<u>ElongBoolean</u>	IsDefaultItemByName ([in] BSTR Name , [in] long ItemIndex) Name Attribute name of the saved attribute data ItemIndex item (element) index , 1 to ParentInterface.count If returns IbTrue then it is the default attribute.

long	SetAllItemsByIndex ([in] long AttributeIndex , [in] SAFEARRAY (byte) * ItemsData)
	AttributeIndex Attribute index of the saved attribute data ItemsData custom data of all elements (length = n * default attribute data size) Overwrite attribute data of all items (elements) with new data. If successful it returns attribute index, otherwise it returns error (atteCannotSetAttribute , atteInvalidOrEmptyItemData , atteItemsDataMustContainAllItems , errIndexOutOfBoundsException , errDatabaseNotReady).
long	SetAllItemsByIndex_vb ([in] long AttributeIndex , [in] SAFEARRAY (byte) * ItemsData , [in] SAFEARRAY (byte) * ItemData)
	Visual Basic compatible function of SetAllItemsByIndex.
long	SetAllItemsByName ([in] BSTR Name , [in] SAFEARRAY (byte) * ItemsData)
	Name Attribute name of the saved attribute data ItemsData custom data of all elements (length = n * default attribute data size) Overwrite attribute data of all items (elements) with new data. If successful it returns attribute index, otherwise it returns error (atteCannotSetAttribute , atteInvalidOrEmptyItemData , atteItemsDataMustContainAllItems , atteAttributeNotFound , atteInvalidName , errDatabaseNotReady).
long	SetAllItemsByName_vb ([in] BSTR Name , [i/o] SAFEARRAY (byte) * ItemsData)
	Visual Basic compatible function of SetAllItemsByName.

Properties

long	Count
	Get number of attributes of the parent interface elements
ELongBoolean	InheritCopiedElements [long AttributeIndex] • Get or set whether copied elements should copy their custom data to the new copied elements or use default data
ELongBoolean	InheritDividedElements [long AttributeIndex] • Get or set whether divided elements should copy their custom data to the new copied elements or use default data

IAxisVMCalculation

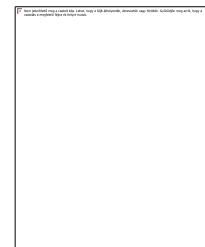
Solver interface of AxisVM.

Error codes

enum ECalculationError = {	
ceSE2moduleNotAvailable = -100001	extension module SE2 is not available
ceNLpackageNotAvailable = -100002	package NL (non-linear) is not available
ceDYNmoduleNotAvailable = -100003	extension module DYN is not available
ceStoryIDOutOfBounds = -100004	Story index is out of range
}	

Enumerated types

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



enum ECalculationUserInteraction = {	
cuiUserInteraction = 0x0,	

user has to interact with the program to answer questions arising during the analysis (e.g. domains are not meshed, degrees of freedom not set correctly)

AxisVM tries to correct problems automatically during the analysis

AxisVM stops if a problem is detected during the analysis

same as

cuiNoUserInteractionWithAutoCorrect, but the calculation window won't be visible

same as

cuiNoUserInteractionWithoutAutoCorrect, but the calculation window won't be visible

cuiNoUserInteractionWithAutoCorrect = 0x1,

cuiNoUserInteractionWithoutAutoCorrect = 0x2,

cuiNoUserInteractionWithAutoCorrectNoShow = 0x3,

cuiNoUserInteractionWithoutAutoCorrectNoShow = 0x4,

Set the way of interaction between the user and the solver.

Note:

Set visible property of IAxisVMAplication interface to lbFalse to skip error messages.

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

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,	<i>force control (equal load steps)</i>
scDisplacement = 0x1,	<i>displacement control (equal displacement steps)</i>
scArcLength = 0x2,	<i>arc length control</i>
scPushOver = 0x3 }	<i>pushover analysis</i>

Way of solution control for nonlinear analysis.

```
enum EMassMatrixType = {  
    mtDiagonal = 0x0,  
    mtConsistent = 0x1}  
    Concentrated mass matrix  
    Distributed mass matrix
```

enum **EMassesTakenIntoAccount** = {
mtiaAll = 0x0,
mtiaAboveHeightZ = 0x1
mtiaAboveSelectedStory = 0x2 }
Setting used in vibration analysis regarding considered masses.

Records / structures

<p>long RNonLinearAnalysis = (</p> <p> LoadCase</p> <p> long Iterations</p> <p> ESolutionControl</p> <p> long NodeId</p> <p> EAxis</p> <p> EBoolean</p> <p> double MaxDisplacement</p> <p> long Increments</p> <p> double DisplacementConvergenceValue</p> <p> double ForceConvergenceValue</p> <p> double WorkConvergenceValue</p> <p> EBoolean EnableDisplacementConvergence</p> <p> EBoolean EnableForceConvergence</p> <p> EBoolean EnableWorkConvergence</p> <p> EBoolean GeometricNonlinearity</p> <p> EBoolean ReinforcementCalculation</p> <p> EBoolean StoreLastIncrementOnly</p> <p> EBoolean ReinforcementCalculation</p> <p> EBoolean ContinueWithoutConvergence</p> <p> long IncrementFunctionId</p> <p> EBoolean ConsiderCreep</p> <p> EBoolean ConsiderShrinkage</p> <p>)</p>	<p><i>load case or load combination to analyse</i></p> <p>IMPORTANT NOTE: If it is the index of the load combination then:</p> <p><i>Load combination index = LoadCase – IAxisVMLoadcases.count</i></p> <p><i>maximum number of iterations</i></p> <p><i>way of solution control</i></p> <p><i>index of the node for displacement control</i></p> <p><i>direction of displacement for displacement control</i></p> <p><i>Enable/disable material non-linearity</i></p> <p><i>maximum displacement for displacement control [m]</i></p> <p><i>number of increments</i></p> <p><i>convergence criterion for displacement</i></p> <p><i>convergence criterion for force</i></p> <p><i>convergence criterion for work</i></p> <p><i>DisplacementConvergenceValue enabled</i></p> <p><i>ForceConvergenceValue enabled</i></p> <p><i>WorkConvergenceValue enabled</i></p> <p><i>taking into account the geometric nonlinearity</i></p> <p><i>taking into account the reinforcement</i></p> <p><i>only the last increment is stored</i></p> <p><i>use actual or calculated reinforcement</i></p> <p><i>if ReinforcementCalculation = True</i></p> <p><i>If true then analysis continues without convergence</i></p> <p><i>Index of the increment function, see here</i></p> <p><i>consider the effect of creep on the stiffness of RC structures</i></p> <p><i>More here...</i></p> <p><i>consider shrinkage strain by the calculation of RC structures</i></p>
--	---

These settings correspond to settings on Nonlinear Static Analysis form.

<p>long RVibration = (</p> <p> LoadCase</p> <p> long Iterations</p> <p> long ModeShapes</p> <p> double EigenValueConvergence</p> <p> double EigenVectorConvergence</p> <p> EMassControl</p> <p> EBoolean ConvertLoadsToMasses</p> <p> EBoolean ConcentratedMasses</p> <p> EBoolean ConvertConcentratedMassesToLoads</p> <p> EBoolean ElementMasses</p> <p> EBoolean MassComponentX</p> <p> EBoolean MassComponentY</p> <p> EBoolean MassComponentZ</p> <p> EBoolean ConvertSlabsToDiaphragms</p> <p> EMassMatrixType</p> <p> EBoolean IncreasedSupportStiffness</p> <p> EMassesTakenIntoAccount</p> <p> double HeightZ</p> <p> long StoryID</p> <p>)</p>	<p><i>load case or load combination to analyse</i></p> <p>IMPORTANT NOTE: If it is the index of the load combination then:</p> <p><i>Load combination index = LoadCase – IAxisVMLoadcases.count</i></p> <p><i>maximum number of iterations</i></p> <p><i>number of vibration shapes to determine</i></p> <p><i>eigenvalue convergence criterion</i></p> <p><i>eigenvector convergence criterion</i></p> <p><i>type of mass control</i></p> <p>NOT USED, see ElementMasses</p> <p><i>if MassControl = mcConvertLoadsToMasses enables user-defined concentrated masses</i></p> <p><i>used only for nonlinear(2nd order) vibration analysis</i></p> <p><i>if MassControl = mcConvertLoadsToMasses enables the conversion of concentrated masses to loads</i></p> <p><i>else if MassControl = mcMassesOnly enables the conversion of masses to loads (Convert Masses to Loads checkbox)</i></p> <p><i>taking into account the element masses</i></p> <p><i>include the mX mass component</i></p> <p><i>include the mY mass component</i></p> <p><i>include the mZ mass component</i></p> <p><i>temporary conversion of slabs to diaphragms</i></p> <p><i>mass matrix type</i></p> <p><i>increased support stiffness</i></p> <p><i>what masses are taken into account, see pic. below</i></p> <p><i>above height Z, see pic. below</i></p> <p><i>index of the storey</i></p>
--	--

These settings correspond to settings on Vibration Analysis form.

<pre> double SavingInterval ELongBoolean LoadsAndNodalMassesForDamping) These settings correspond to settings on dynamic analysis form. </pre>	<p><i>interval of saving results [s]</i> <i>both loads and masses are taken into account for damping (used with Rayleigh damping coefficient a)</i></p>
<pre> long RPushOverAnalysis = (LoadCase </pre>	<p><i>load case or load combination to analyse</i></p>
<pre> long ConstantLoadCase long NodeId EAxis Direction ELongBoolean MaterialNonLinearity double MaxDisplacement long Increments long Iterations double DisplacementConvergenceValue double ForceConvergenceValue double WorkConvergenceValue ELongBoolean EnableDisplacementConvergence ELongBoolean EnableForceConvergence ELongBoolean EnableWorkConvergence ELongBoolean GeometricNonlinearity ELongBoolean ContinueWithoutConvergence) </pre>	<p>IMPORTANT NOTE: If it is the index of the load combination then: <i>Load combination index = LoadCase -IAxisVMLoadcases.count</i> <i>optional static load case ID which can be enabled in analysis</i> <i>index of the node for displacement control</i> <i>direction of displacement for displacement control</i> <i>Enable/disable material non-linearity</i> <i>maximum displacement for displacement control [m]</i> <i>number of increments</i> <i>Number of iterations</i> <i>convergence criterion for displacement</i> <i>convergence criterion for force</i> <i>convergence criterion for work</i> <i>DisplacementConvergenceValue enabled</i> <i>ForceConvergenceValue enabled</i> <i>WorkConvergenceValue enabled</i> <i>taking into account the geometric nonlinearity</i> <i>If true then analysis continues without convergence</i></p>
<p><i>These settings correspond to settings on Nonlinear Static Analysis form.</i></p>	

Functions

<code>ELongBoolean Buckling ([i/o] RBuckling AnalysisParameters, [in] ECalculationUserInteraction UserInteraction)</code>	<p>AnalysisParameters analysis parameters UserInteraction type of interaction during calculation</p>
--	---

Buckling analysis of the model. Returns True if the analysis was completed.

<code>ELongBoolean Buckling2 ([i/o] RBuckling AnalysisParameters, [in] ECalculationUserInteraction UserInteraction, [in] ELongBoolean ShowError, [out] BSTR* ErrorList)</code>	<p>AnalysisParameters analysis parameters UserInteraction type of interaction during calculation ShowError show error message in AxisVM, ErrorList is always returned ErrorList message with errors after analysis</p>
--	---

Buckling analysis of the model. Returns True if the analysis was completed.

<code>ELongBoolean DynamicAnalysis ([i/o] RDynamicAnalysis* AnalysisParameters, [in] ECalculationUserInteraction UserInteraction, [in] ELongBoolean ShowError, [out] BSTR* ErrorList)</code>	<p>AnalysisParameters analysis parameters UserInteraction type of interaction with user ShowError show error message in AxisVM, ErrorList is always returned ErrorList message with errors after analysis</p>
--	--

Dynamic analysis of the model. Returns True if the analysis was completed.

<code>ELongBoolean LinearAnalysis ([in] ECalculationUserInteraction UserInteraction)</code>	<p>UserInteraction type of interaction during calculation</p>
---	--

Linear analysis of the model. Returns True if the analysis was completed.

EBoolean	LinearAnalysis2 ([in] ECalculationUserInteraction UserInteraction, [in] EBoolean ShowError, [out] BSTR* ErrorList)
	<p>UserInteraction type of interaction during calculation</p> <p>ShowError show error message in AxisVM, ErrorList is always returned</p> <p>ErrorList message with errors after analysis</p>
	<i>Linear analysis of the model. Returns True if the analysis was completed.</i>
EBoolean	NonLinearAnalysis ([i/o] RNonLinearAnalysis AnalysisParameters, [in] ECalculationUserInteraction UserInteraction)
	<p>AnalysisParameters analysis parameters</p> <p>UserInteraction type of interaction during calculation</p>
	<i>Nonlinear analysis of the model. Returns True if the analysis was completed.</i>
EBoolean	NonLinearAnalysis 2 ([i/o] RNonLinearAnalysis AnalysisParameters, [in] ECalculationUserInteraction UserInteraction, [in] EBoolean ShowError, [out] BSTR* ErrorList)
	<p>AnalysisParameters analysis parameters</p> <p>UserInteraction type of interaction during calculation</p> <p>ShowError show error message in AxisVM, ErrorList is always returned</p> <p>ErrorList message with errors after analysis</p>
	<i>Nonlinear analysis of the model. Returns True if the analysis was completed.</i>
EBoolean	NonLinearVibration ([i/o] RVibration AnalysisParameters, [in] ECalculationUserInteraction UserInteraction)
	<p>AnalysisParameters analysis parameters</p> <p>UserInteraction type of interaction during calculation</p>
	<i>Second order vibration analysis of the model. Returns True if the analysis was completed.</i>
EBoolean	NonLinearVibration2 ([i/o] RVibration AnalysisParameters, [in] ECalculationUserInteraction* UserInteraction, [in] EBoolean ShowError, [out] BSTR* ErrorList)
	<p>AnalysisParameters analysis parameters</p> <p>UserInteraction type of interaction during calculation</p> <p>ShowError show error message in AxisVM, ErrorList is always returned</p> <p>ErrorList message with errors after analysis</p>
	<i>Second order vibration analysis of the model. Returns True if the analysis was completed.</i>
EBoolean	Vibration ([i/o] RVibration AnalysisParameters, [in] ECalculationUserInteraction UserInteraction)
	<p>AnalysisParameters analysis parameters</p> <p>UserInteraction type of interaction during calculation</p>
	<i>First order vibration analysis of the model. Returns True if the analysis was completed.</i>

ELongBoolean	Vibration2 ([i/o] RVibration AnalysisParameters , [in] ECalculationUserInteraction UserInteraction , [in] ELongBoolean ShowError , [out] BSTR* ErrorList)
	AnalysisParameters <i>analysis parameters</i>
	UserInteraction <i>type of interaction during calculation</i>
	ShowError <i>show error message in AxisVM, ErrorList is always returned</i>
	ErrorList <i>message with errors after analysis</i>
	<i>First order vibration analysis of the model. Returns True if the analysis was completed.</i>
ELongBoolean	PushOverAnalysis ([i/o] RPushOverAnalysis AnalysisParameters , [in] ECalculationUserInteraction UserInteraction , [in] ELongBoolean ShowError , [out] BSTR* ErrorList)
	AnalysisParameters <i>analysis parameters</i>
	UserInteraction <i>type of interaction during calculation</i>
	ShowError <i>show error message in AxisVM, ErrorList is always returned</i>
	ErrorList <i>message with errors after analysis</i>
	<i>Pushover analysis of the model. Returns True if the analysis was completed.</i>

Properties

ELongBoolean	CallMainProgress • If set to True, calculation progress COM events are regularly called. See IAxisVMCalculationEvents . Read and write property.
ELanguage	ReportLanguage • Get or set report language

IAxisVMColumnRebars

Interface used for defining rebar (reinforcement) in concrete columns. If property returning this interface is null (nil) then the extension module RC2 is not available.
Only one instance of his interface is allowed.

Error codes

```
enum EColumnRebarsError = {
    crbelInvalidCrossSectionId = -100001 }      invalid cross section index
```

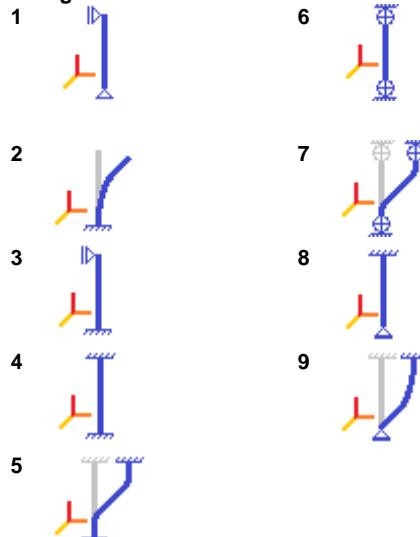
Enumerated types

enum	ESeismicDuctilityClass = { sdc_DCM = 0x0, sdc_DCH = 0x1 }	enum <i>medium ductility class</i> <i>high ductility class</i>
------	--	--

Records / structures

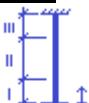
RColumnRebarPos = (double x double y double d)	<i>local x coordinate of the rebar from bottom left corner of the cross section</i> <i>local y coordinate of the rebar from bottom left corner of the cross section</i> <i>diameter of the rebar</i>
RColumnCheckingParameters = (<u>EBoolean</u> CalcEccIncz <u>EBoolean</u> CalcEccIncy <u>EBoolean</u> CalcEcc2Y <u>EBoolean</u> CalcEcc2Z double Ky double Kz <u>EBoolean</u> IsCustomLength double Columnlength double fieff_EC_ITA	<i>calculate eccentricity in local z direction</i> <i>calculate eccentricity in local y direction</i> <i>calculate second order eccentricity in local y direction</i> <i>calculate second order eccentricity in local z direction</i> <i>buckling length factor in local x-z plane of the column</i> <i>buckling length factor in local x-y plane of the column</i> <i>over write length of the column</i> <i>length of the column (only for custom length)</i> φ_{eff} <i>effective creep ratio (only for EuroCode and Italian DesignCode),</i> <i>see 5.8.4 in EN 1992-1-1:2004(E)</i>
<u>EBoolean</u> SwayImp long BucklingModeY long BucklingModeZ <u>EBoolean</u> ApproximateCurvatureSIA	<i>consider sway geometric imperfections (see 5.2 (7) in EN 1992-1-1)</i> <i>buckling mode in local x-z plane of the column</i> <i>buckling mode in local x-y plane of the column</i> <i>consider approximate curvature in calculation of second order eccentricities (used only if SIA standard is selected)</i>
double ShrinkCreepEpsSIA <u>EBoolean</u> Sum2ndEccentricities	<i>shrinkage + creep strain (used only if SIA standard is selected)</i> <i>consider both 2nd eccentricities calculated in local y and z directions simultaneously or separately</i>
)	

Buckling modes used in fields **BucklingModeY** and **BucklingModeZ** :



RColumnStirrupSpacing = (
double ss_bottom	spacing between stirrups in the bottom zone
double ss_middle	spacing between stirrups in the middle zone
double ss_top	spacing between stirrups in the top zone
)	
RColumnStirrupDiameters = (
double sd_bottom	diameter of stirrups in the bottom zone
double sd_middle	diameter of stirrups in the middle zone
double sd_top	diameter of stirrups in the top zone
)	
RColumnStirrupZones = (
double sz_bottom	the position of boundary between bottom and middle zones (relative to the column length)
double sz_middle	the position of boundary between middle and top zones (relative to the column length)
)	

Notes: Column's stirrup zones



RC column divided into three zones, namely bottom (I), middle (II) and top (III) zones. In each zones, different stirrup spacing and diameters can be defined.

RColumnReinforcementParameters = (
long ColumnRebarsId	column rebar index
long ConcreteMaterialId	concrete material index
long RebarSteelGradeId	rebar steel grade index
RColumnCheckingParameters	column checking parameters
double fse	load factor for seismic forces (default is 1 =>no change)
EBoolean TakeConcTensileStrength	take tensile strength of the concrete into account in nonlinear analysis
EBoolean Usefcfmfl	consider flexural tensile strength in nonlinear analysis instead of tensile strength (only for EC and ITA)
double ShrinkageEps	shrinkage strain considered in nonlinear analysis
EBoolean SpiralStirrup	use spiral stirrup (only for circular sections)
RColumnStirrupSpacing	spacing between stirrups
RColumnStirrupDiameters	diameter of stirrups
RColumnStirrupZones	boundary of stirrup zones
long StirrupLegNumY	number of stirrup's legs parallel to local y axis
long StirrupLegNumZ	number of stirrup's legs parallel to local z axis
long ShearCrackAngle	angle of shear crack considered in the calculation of shear and torsion resistance
RRCColumnCapacityDesignParams	parameters for capacity design
EBoolean CBDetailingRules	consider detailing rules regarding to the diameter of compression rebars
long SteelMaterialId	steel material index (composite section)
double ShearRhoFactor	shear utilization of steel section's web (composite section)
)	
RRCColumnCapacityDesignParams = (
EBoolean PlasticHinges	consider plastic hinges in the column
ESeismicDuctilityClass	ductility class
EBoolean PlasticHingeYY_Top	plastic hinge can be formed due to yy bending moment at the top of the column
EBoolean PlasticHingeYY_Bottom	plastic hinge can be formed due to yy bending moment at the bottom of the column
EBoolean PlasticHingeZZ_Top	plastic hinge can be formed due to zz bending moment at the top of the column
EBoolean PlasticHingeZZ_Bottom	plastic hinge can be formed due to zz bending moment at the bottom of the column
double MRdB_MRdC_RatioYY_Top	ratios between the sum of bending moment capacity of beams and columns connected to the same joint
double MRdB_MRdC_RatioYY_Bottom	
double MRdB_MRdC_RatioZZ_Top	
double MRdB_MRdC_RatioZZ_Bottom	
double RelativeClearLengthYY	distances between end sections where plastic hinges can be formed
double RelativeClearLengthZZ	
double GammaRd	safety factor
)	

Functions

long **Add** ([in] BSTR **Name**, [in] long **CrossSectionId**, [in] SAFEARRAY([RColumnRebarPos](#)) **Rebars**)

Name name of the column rebar

CrossSectionId index of the cross section for column rebar

Rebars rebar parameters

Returns column rebar index if successful, otherwise returns an error code ([errDatabaseNotReady](#), [errCOMServerInternalError](#), [crbelInvalidCrossSectionId](#)).

long **Add_vb** (Visual Basic compatible function of [Add](#))

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

Index column rebar index

Delete column rebar. Returns column rebar index if successful, otherwise returns an error code ([errDatabaseNotReady](#), [errIndexOutOfBounds](#)).

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

Index column rebar index

Delete column rebar parameters. Returns column rebar index if successful, otherwise returns an error code ([errDatabaseNotReady](#), [errIndexOutOfBounds](#)).

long **GetRebars** ([in] long **Index**, [out] SAFEARRAY([RColumnRebarPos](#)) **Rebars**)

Index column rebar index

Rebars rebar parameters

Returns number of column rebars, if successful, otherwise returns an error code ([errDatabaseNotReady](#), [errIndexOutOfBounds](#), [errCOMServerInternalError](#)).

long **SetRebars** ([in] long **Index**, [in] SAFEARRAY([RColumnRebarPos](#)) **Rebars**)

Index column rebar index

Rebars rebar parameters

Returns index, if successful, otherwise returns an error code ([errDatabaseNotReady](#), [errIndexOutOfBounds](#), [errCOMServerInternalError](#)).

long **SetRebars_vb** (Visual Basic compatible function of [SetRebars](#))

Properties

long **Count** Get number of column rebars

long **CrossSectionId** [long **index**] • Get or set section index of the column rebar with rebar index

Index rebar index, $1 \leq \text{Index} \leq \text{Count}$

BSTR **Name** • Get or set name of column rebars

long **RebarArea** [long **index**] Get area of the rebar with rebar index

Index rebar index, $1 \leq \text{Index} \leq \text{Count}$

long **RebarCount** Get number of rebar in the column

IAxisVMCriticalGroupCombinations

Adding and editing critical load group combinations considered in critical results of the model.

Error codes

```
enum ECriticalGroupCombinationsError = {
    cgceLoadGroupIdOutOfBounds = -100001,           index of load group combination is invalid
    cgceNotEditable = -100002 }                      load group combination can not be edited
```

Functions

long **AddDefaultCombinations**

Returns critical group combination index if successful, otherwise returns an error code ([errDatabaseNotReady](#)).

long **Clear**

Delete all critical group combinations. Returns number of deleted critical group combination if successful, otherwise returns an error code ([errDatabaseNotReady](#)).

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

Index critical group combination index

Delete critical group combination. Returns critical group combination index if successful, otherwise returns an error code ([errDatabaseNotReady](#), [errIndexOutOfBoundsException](#)).

Properties

[**EBoolean**](#) **Editable** [[long LoadGroupId](#)] Returns `IbTrue` if can be switched on/off for consideration in critical results

LoadGroupId index of the load group, $1 \leq \text{LoadGroupId} \leq \text{IAxisVMLoadGroups.Count}$

long **Count** Get number of critical group combinations

[**EBoolean**](#) **Used** [[long Index](#), [long LoadGroupId](#)] • Get or set if load group with index `LoadGroupId` is considered in critical results.

Index index of critical group combination, $1 \leq \text{index} \leq \text{Count}$

LoadGroupId index of the load group, $1 \leq \text{LoadGroupId} \leq \text{IAxisVMLoadGroups.Count}$

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	$b \leq 0$
	cseNonPositive_h = -100004	$h \leq 0$
	cseNonPositive_tw = -100005	$tw \leq 0$
	cseNonPositive_tf = -100006	$tf \leq 0$
	cseNonPositive_d = -100007	$d \leq 0$
	cseNonPositive_v = -100008	$v \leq 0$
	cseNonPositive_b1 = -100009	$b1 \leq 0$
	cseNonPositive_h1 = -100010	$h1 \leq 0$
	cseNonPositive_b2 = -100011	$b2 \leq 0$
	cseNonPositive_h2 = -100012	$h2 \leq 0$
	cseNonPositive_tw1 = -100013	$tw1 \leq 0$
	cseNonPositive_tf1 = -100014	$tf1 \leq 0$
	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	<i>h too large</i>
	cseTooHigh_tw = -100024	<i>tw too large</i>
	cseTooHigh_tf = -100025	<i>tf too large</i>
	cseTooHigh_r1 = -100026	<i>r1 too large</i>
	cseTooHigh_r2 = -100027	<i>r2 too large</i>
	cseTooHigh_r3 = -100028	<i>r3 too large</i>
	cseTooHigh_v = -100029	<i>v too large</i>
	cseTooLow_e = -100030	<i>e too small</i>
	cseTooHigh_tw1 = -100031	<i>tw1 too large</i>
	cseTooHigh_tf1 = -100032	<i>tf1 too large</i>
	cseTooHigh_tw2 = -100033	<i>tw2 too large</i>
	cseTooHigh_tf2 = -100034	<i>tf2 too large</i>
	cseTooLow_h = -100035	<i>h too small</i>
	cseTooLow_r1 = -100036	<i>r1 too small</i>
	cseTooLow_N = -100037	<i>N too small</i>
	cseDifferentThicknesses = -100038	$v \neq t$
	cseDifferentWidthAndHeight = -100039	$b \neq h$
	cseIncompatibleWidthAndHeight = -100040	$b \neq 2h$
	cseNonPositiveAx = -100041	$Ax \leq 0$
	cseNegativeAy = -100042	$Ay < 0$
	cseAyIsHigherThanAx = -100043	$Ay > Ax$
	cseNegativeAz = -100044	$Az < 0$
	cseAzIsHigherThanAx = -100045	$Ay > Ax$
	cseNonPositiveIx = -100046	$Ix \leq 0$
	cseNonPositively = -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	<i>cross-section name is empty</i>
	cseNameAlreadyExists = -100059	<i>cross-section name already exists</i>
	cseExtParams = -100060	<i>error in extended parameters</i>
	cseErrorAdding = -100061	<i>error while adding or Cancel pressed</i>
	cseErrorEditing = -100062	<i>error while editing, invalid cross-section or Cancel pressed</i>
	cseInvalidCrossSectionType = -100063	<i>invalid cross section type (solid sections accepted)</i>
	cseDifferentCrossSectionShape = -100064	<i>cross section shape does not match the section's shape</i>
	cseTooHigh_e = -100065	<i>e too large</i>
	cseTooLargeInnerCrossSection = -100066	<i>does not fit into boundaries</i>
	cseInvalidMaterials = -100067	<i>inner section should be metal</i>
	cseException = -101000 }	<i>other error</i>

Cross-section error codes.

Enumerated types

enum	ECrossSectionShape = {	
	cssAll = 0x00000000,	<i>all shapes</i>
	cssCustom = 0x00000001,	<i>custom shape</i>
	cssRectangular = 0x00000002,	<i>rectangular shape</i>
	cssI = 0x00000004,	<i>I shape</i>
	cssDoubleI = 0x00000008,	<i>double I shape</i>
	cssWedgedI = 0x00000010,	<i>wedged I shape</i>
	cssAsymmetricI = 0x00000020,	<i>asymmetric I shape</i>
	cssPipe = 0x00000040	<i>pipe</i>
	cssRegularPolygon = 0x00000080,	<i>regular polygon</i>
	cssBox = 0x00000100,	<i>box shape</i>
	cssDoubleBox = 0x00000200,	<i>double I box shape</i>
	cssU = 0x00000400,	<i>U shape</i>
	cssDoubleUOpened = 0x00000800,	<i>opened double U shape</i>
	cssDoubleUClosed = 0x00001000,	<i>closed double U shape</i>
	cssL = 0x00002000,	<i>unequal angle</i>
	cssDoubleL = 0x00004000,	<i>double angle</i>
	cssDoubleLFlange = 0x00008000,	<i>double angles connected at their flanges</i>
	cssT = 0x00100000,	<i>T shape</i>
	cssZ = 0x00200000,	<i>Z shape</i>
	cssC = 0x00400000,	<i>C shape</i>
	cssS = 0x00800000,	<i>S shape</i>
	cssJ = 0x01000000,	<i>J shape</i>
	cssCircle = 0x02000000 }	<i>circle shape</i>
	cssRectangleRounded = 0x00400000,	<i>rectangle shape with rounded corners</i>
	cssRectangleHollow = 0x00800000,	<i>rectangle shape with hollow inside</i>
	cssHaunched = 0x01000000,	<i>I shape with haunched flanges</i>
	cssTWallHaunched = 0x02000000,	<i>T shape with haunched web</i>
	cssTTopHaunched = 0x04000000,	<i>I shape with haunched top flange</i>
	cssCircleHollow = 0x08000000,	<i>circle shape with hollow inside</i>
	cssTrapezoid = 0x10000000,	<i>Symmetric trapezoid shape</i>
	css2LX = 0x20000000,	<i>2 No. L-shapes</i>
	css4L = 0x40000000,	<i>4 No. L-shapes (X cross)</i>

Cross-section shapes, superseded with ECrossSectionShapeEx, will be deprecated

enum	ECrossSectionShapeEx = {	
	cssAll = 0x00000000,	<i>all shapes</i>
	cssCustom = 0x00000001,	<i>custom shape</i>
	cssRectangular = 0x00000002,	<i>rectangular shape</i>
	cssI = 0x00000003,	<i>I shape</i>
	cssDoubleI = 0x00000004,	<i>double I shape</i>
	cssWedgedI = 0x00000005,	<i>wedged I shape</i>
	cssAsymmetricI = 0x00000006,	<i>asymmetric I shape</i>
	cssPipe = 0x00000007	<i>pipe</i>
	cssRegularPolygon = 0x00000008,	<i>regular polygon</i>
	cssBox = 0x00000009,	<i>box shape</i>
	cssDoubleBox = 0x00000010,	<i>double I box shape</i>
	cssU = 0x00000011,	<i>U shape</i>
	cssDoubleUOpened = 0x00000012,	<i>opened double U shape</i>
	cssDoubleUClosed = 0x00000013,	<i>closed double U shape</i>
	cssL = 0x00000014,	<i>unequal angle</i>
	cssDoubleL = 0x00000015,	<i>double angle</i>
	cssDoubleLFlange = 0x00000016,	<i>double angles connected at their flanges</i>
	cssT = 0x00000017,	<i>T shape</i>
	cssZ = 0x00000018,	<i>Z shape</i>
	cssC = 0x00000019,	<i>C shape</i>
	cssS = 0x00000020,	<i>S shape</i>
	cssJ = 0x00000021,	<i>J shape</i>

```

cssCircle = 0x00000022,
cssRectangleRounded = 0x00000023,
cssRectangleHollow = 0x00000024,
cssIHaunched = 0x00000025,
cssTWallHaunched = 0x00000026,
cssTTopHaunched = 0x00000027,
cssCircleHollow = 0x00000028,
cssTrapezoid = 0x00000029,
csse2LX = 0x00000030,
csse4L = 0x00000031,
csseCross = 0x00000032,
csseCompositePipe = 0x00000033,
csseCompositeBox = 0x00000034,
csseCompositeRound = 0x00000035,
csseCompositeRectangle = 0x00000036
}

Cross-section shapes, replaces ECrossSectionShape in the future versions

```

enum **ECrossSectionBasicType** = {
csbt_All = 0x0,
csbt_Solid = 0x1,
csbt_ThinWalled = 0x2,
csbtCompositeThinWalled = 0x4,
csbtCompositeSolid = 0x8 }
Basic type of cross section

all basic types
solid cross sections
thin walled cross sections
composite with inner and outer cross section
composite with inner cross section

enum **ECrossSectionProcess** = {
cspOther = 0x0,
cspRolled = 0x1,
cspWelded = 0x2,
cspColdFormed = 0x3 }
How the cross-section was manufactured.

other
rolled
welded
cold-formed

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

enum **ECrossSectionImageExportOptions**= {
csieoNone= 0x0,
csieoFilled= 0x1,
csieoStressPointsMarks= 0x2,
csieoStressPointsLabels= 0x4
csieoMainAxis= 0x8,
csieoMainAxisLabels= 0x16}

none
filled
marks of stress points
labels of stress points
main axis
main axis labels

enum **ECompositInnerCSalign** = { **cicsa_CentreGravity** = 0x0, **cicsa_Centre** = 0x1}
Not used yet, actual default: cicsa_CentreGravity

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 [EGeneralErrors](#) 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.

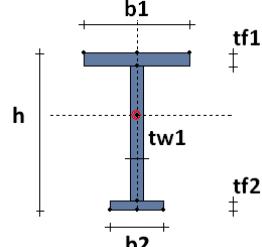
long	AddFromDialog ([in] long CrossSectionBasicTypes , [in] long CrossSectionShapes , [in] SAFEARRAY(BSTR) CatalogFileNames)
	CrossSectionBasicTypes cross-section types (ECrossSectionBasicType value or sum of values) CrossSectionShapes cross-section shapes (ECrossSectionShape value or sum of values) CatalogFileNames name of the catalog files, when nil all files are shown
	<i>Opens Dialog box with cross sections. Adds selected cross-section(s) from selected catalog file(s) or default catalog files if CatalogFileNames is empty (nil). If successful, returns index of the last added cross section, otherwise an error (cseErrorAdding).</i>

long	AddFromDialog_vb (Visual Basic compatible function of AddFromDialog)
------	---

long	AddFromDialogEx ([in] long CrossSectionBasicTypes , [in] SAFEARRAY(long) CrossSectionShapes , [in] SAFEARRAY(BSTR) CatalogFileNames)
	CrossSectionBasicTypes cross-section types (ECrossSectionBasicType value or sum of values) CrossSectionShapes array with enabled cross-section shapes, integer array of type casted (ECrossSectionShapeEx) CatalogFileNames name of the catalog files, when nil all files are shown
	<i>Opens Dialog box with cross sections. Adds enabled cross-section(s) from selected catalog file(s) or default catalog files if CatalogFileNames is empty (nil). If successful, returns index of the last added cross section, otherwise an error (cseErrorAdding).</i>

long	AddFromEditor ([in] long CrossSectionBasicTypes , [in] long CrossSectionShapes)
	CrossSectionBasicTypes cross-section types (ECrossSectionBasicType value or sum of values) CrossSectionShapes cross-section shapes (ECrossSectionShape value or sum of values)
	<i>Adds a cross-section from a cross section editor. If successful, returns index of the cross section, otherwise an error (cseErrorAdding).</i>

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

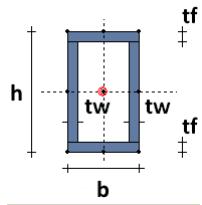


Adds an asymmetric I-shape to the model.

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

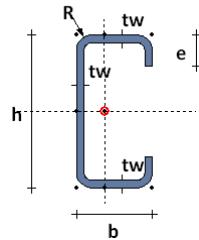
Process the manufacturing process



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)

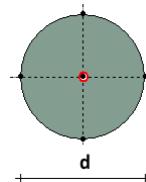
Name name of the new cross-section
h cross-section height
b cross-section width
e leg height
tw wall thickness
R fillet radius
Process the manufacturing process



Adds a C-shape to the model.

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

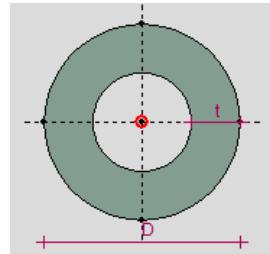
Name name of the new cross-section
d circle diameter



Adds a circle shape to the model.

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

Name name of the new cross-section
d circle diameter
t thickness



Adds a hollow circle shape to the model.

long **AddCompositePipe** ([in] BSTR Name, [in] double Diameter, [in] double t, [in] long OuterMaterial_ID, [in] long InnerMaterial_ID, [in] long CS_MaterialID, [in] long CS_ID, [in] ECompositeInnerCSAlign InnerCSAlign)

Name name of the new cross-section
Diameter outer diameter of the pipe cross-section
t thickness of the pipe cross-section
OuterMaterial_ID index of the material of the pipe cross-section
InnerMaterial_ID index of inner (fill) material
CS_MaterialID index of the inner cross-section
CS_ID index of the material of the cross-section inside
InnerCSAlign not used

Adds a composite pipe section with inner cross-section filled with inner material. Returns index of the new composite cross-section.

long **AddCompositeBox** ([in] BSTR **Name**, [in] double **b**, [in] double **h**, [in] double **t_flange**,
[in] double **t_web**, [in] double **r**, [in] long **OuterMaterial_ID**, [in] long **InnerMaterial_ID**, [in] long
CS_MaterialID, [in] long **CS_ID**, [in] [ECompositeInnerCSalign](#) **InnerCSalign**)

Name *name of the new cross-section*
b *width of the hollow rectangular cross-section*
h *height of the hollow rectangular cross-section*
t_flange *flange thickness of the hollow rectangular cross-section*
t_web *web thickness of the hollow rectangular cross-section*
r *radius of the hollow rectangular cross-section*
OuterMaterial_ID *index of the material of the pipe cross-section*
InnerMaterial_ID *index of inner (fill) material*
CS_MaterialID *index of the material of the cross-section inside*
CS_ID *index of the material of the cross-section inside*
InnerCSalign *not used*

*Adds a composite rectangular hollow section with cross-section inside and filled with inner material.
Returns index of the new composite cross-section.*

long **AddCompositeRound** ([in] BSTR **Name**, [in] double **d**, [in] long **InnerMaterial_ID**,
[in] long **CS_MaterialID**, [in] long **CS_ID**, [in] [ECompositeInnerCSalign](#) **InnerCSalign**)

Name *name of the new cross-section*
d *outer diameter of the pipe cross-section*
InnerMaterial_ID *index of inner (fill) material*
CS_MaterialID *index of the inner cross-section*
CS_ID *index of the material of the cross-section inside*
InnerCSalign *not used*

*Adds a composite round cross-section with inner cross-section. Returns index of the new
composite cross-section.*

long **AddCompositeRectangle** ([in] BSTR **Name**, [in] double **b**, [in] double **h**,
[in] long **InnerMaterial_ID**, [in] long **CS_MaterialID**, [in] long **CS_ID**,
[in] [ECompositeInnerCSalign](#) **InnerCSalign**)

Name *name of the new cross-section*
b *width of the rectangular cross-section*
h *height of the rectangular cross-section*
InnerMaterial_ID *index of inner (fill) material*
CS_MaterialID *index of the inner cross-section*
CS_ID *index of the material of the cross-section inside*
InnerCSalign *not used*

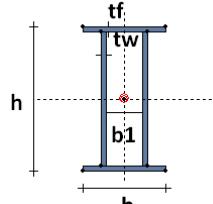
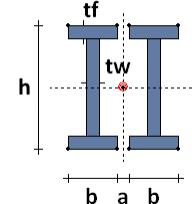
*Adds a composite rectangular cross-section with inner cross-section. Returns index of the new
composite cross-section.*

long **AddCustomWithUserParams** ([in] BSTR **Name**, [in] [AxisVMPolygon2dList](#) * **ShapePolygonList**,
[in] [ECrossSectionProcess](#) **Process**, [i/o] [RCrossSectionUserParams](#)* **CrossSectionUserParams**)

Name *Name of the cross-section*
ShapePolygonList *Polygon list with shape of the cross-section*
Process *the manufacturing process*
CrossSectionUserPara *User parameters*
ms

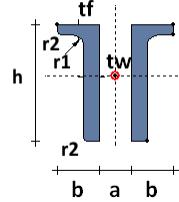
*Returns cross-section index if successful. If not successful, then returns an error code
([errIndexOutOfBoundsException](#))*

long	AddCustomWithUserParamsAsArray ([in] BSTR Name, [in] AxisVMPolygon2dList* ShapePolygonList, [in] ECrossSectionProcess Process, [in] long IParam, [in] SAFEARRAY(double) CrossSectionUserParams)
	<p style="margin-left: 20px;">Name <i>Name of the cross-section</i></p> <p style="margin-left: 20px;">ShapePolygonList <i>Polygon list with shape of the cross-section</i></p> <p style="margin-left: 20px;">Process <i>the manufacturing process</i></p> <p style="margin-left: 20px;">IParam <i>User parameter</i></p> <p style="margin-left: 20px;">CrossSectionUserParams <i>Array with user parameters (only first 10 elements of the array are used)</i></p> <p style="margin-left: 20px;"><i>Returns cross-section index if successful. If not successful, then returns an error code (errIndexOutOfBoundsException)</i></p>
long	AddCustomWithUserParamsAsArray_vb (Visual Basic compatible function of AddCustomWithUserParamsAsArray)
long	AddCustomWithUserParamsAsByteArray ([in] BSTR Name, [in] AxisVMPolygon2dList* ShapePolygonList, [in] ECrossSectionProcess Process, [in] SAFEARRAY(double) CrossSectionUserParams)
	<p style="margin-left: 20px;">Name <i>Name of the cross-section</i></p> <p style="margin-left: 20px;">ShapePolygonList <i>Polygon list with shape of the cross-section</i></p> <p style="margin-left: 20px;">IParam <i>User parameter</i></p> <p style="margin-left: 20px;">Process <i>the manufacturing process</i></p> <p style="margin-left: 20px;">CrossSectionUserParams <i>Array with user parameters (first element is 32bit long other 10 elements of the array are 64bit doubles)</i></p> <p style="margin-left: 20px;"><i>Returns cross-section index if successful. If not successful, then returns an error code (errIndexOutOfBoundsException)</i></p>
long	AddCustomWithUserParamsAsByteArray_vb (Visual Basic compatible function of AddCustomWithUserParamsAsByteArray)
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)
	<p style="margin-left: 20px;">Name <i>name of the new cross-section</i></p> <p style="margin-left: 20px;">h <i>cross-section height</i></p> <p style="margin-left: 20px;">b <i>cross-section width</i></p> <p style="margin-left: 20px;">tw <i>cross-section web thickness</i></p> <p style="margin-left: 20px;">tf <i>cross-section flange thickness</i></p> <p style="margin-left: 20px;">R <i>fillet radius</i></p> <p style="margin-left: 20px;">a <i>distance of the I-shapes (zero for touching shapes)</i></p> <p style="margin-left: 20px;">Process <i>the manufacturing process</i></p> <p style="margin-left: 20px;"><i>Adds a double I-shape to the model.</i></p>
long	AddDoubleIBox ([in] BSTR Name, [in] double h, [in] double b, [in] double b1, [in] double tw, [in] double tf)
	<p style="margin-left: 20px;">Name <i>name of the new cross-section</i></p> <p style="margin-left: 20px;">h <i>cross-section height</i></p> <p style="margin-left: 20px;">b <i>cross-section width</i></p> <p style="margin-left: 20px;">b1 <i>width of the box opening</i></p> <p style="margin-left: 20px;">tw <i>cross-section web thickness</i></p> <p style="margin-left: 20px;">tf <i>cross-section flange thickness</i></p> <p style="margin-left: 20px;"><i>Adds a double I box shape to the model.</i></p>



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)

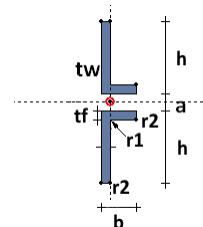
Name name of the new cross-section
h cross-section height
b cross-section width
tw cross-section web thickness
tf cross-section flange thickness
r1 fillet radius at the web/flange connection
r2 fillet radius at the ends
a the distance of angle shapes (zero for touching shapes)
Process the manufacturing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

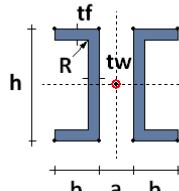


Adds a double angle shape connected at the flanges to the model.

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] ECrossSectionShape CrossSectionShape,
[in] BSTR CrossSectionName)

CrossSectionShape cross-section shape
CrossSectionName cross-section name

Adds a cross-section from the catalog to the model.

Returns the same value as the other Add... functions.

long **AddFromCatalogFile** ([in] BSTR CatalogFileName, [in] BSTR CrossSectionName)

CatalogFileName name of the catalog file
(e.g.: 'c:\Program Files\Axis VM9\EU_LEQ.SEC')

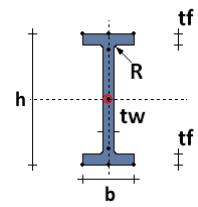
CrossSectionName name of the cross-section

Adds a cross-section from the catalog file 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)

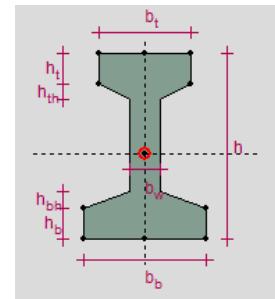
Name name of the new cross-section
h cross-section height
b cross-section width
tw cross-section web thickness
tf cross-section flange thickness
R fillet radius
Process the manufacturing process



Adds an I-shape to the model.

long **AddIHaunched** ([in] BSTR Name, [in] double bt, [in] double bw, [in] double bb, [in] double h, [in] double ht, [in] double hth, [in] double hbh, [in] double hb)

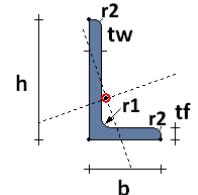
Name name of the new cross-section
bt width at top
bw thickness of the wall
bb width at bottom
h height
ht top flange thickness
hth height of the top haunch
hbh height of the bottom haunch
hb bottom flange thickness



Adds a haunched 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)

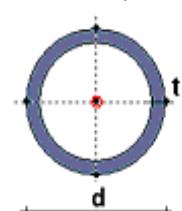
Name name of the new cross-section
h cross-section height
b cross-section width
tw cross-section web thickness
tf cross-section flange thickness
r1 fillet radius at the web/flange connection
r2 fillet radius at the ends
Process the manufacturing process



Adds an unequal angle shape to the model.

long **AddPipe** ([in] BSTR Name, [in] double d, [in] double t, [in] ECrossSectionProcess Process)

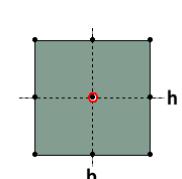
Name name of the new cross-section
d external diameter of the pipe
t pipe wall thickness
Process the manufacturing process



Adds a pipe to the model.

long **AddRectangular** ([in] BSTR Name, [in] double b, [in] double h, [in] ECrossSectionProcess Process)

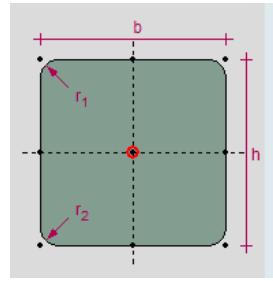
Name name of the new cross-section
b cross-section width
h cross-section height
Process the manufacturing process



Adds a rectangular shape to the model.

long **AddRectangularRounded** ([in] BSTR Name, [in] double b, [in] double h, [in] double r1, [in] double r2)

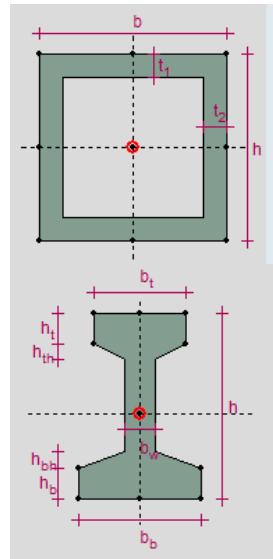
Name name of the new cross-section
b cross-section width
h cross-section height
r1 fillet radius at top
r2 fillet radius at bottom



Adds a rectangular shape with rounded corners to the model.

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

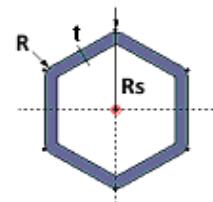
Name name of the new cross-section
b cross-section width
h cross-section height
tf cross-section flange thickness
tw cross-section web thickness



Adds a hollow rectangular shape to the model.

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

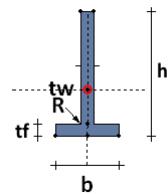
Name name of the new cross-section
N number of polygon sides
Rshape diameter of the circumcircle
t cross-section wall thickness
R fillet radius
Process the manufacturing process



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

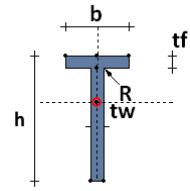


Adds a reversed T-shape to the model.

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

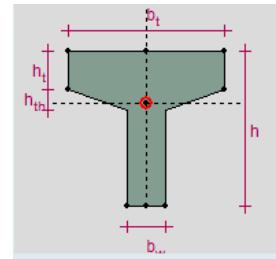
Process the manufacturing process



Adds a T-shape to the model.

long **AddTTopHaunched** ([in] BSTR **Name**, [in] double **bt**, [in] double **bw**, [in] double **h**, [in] double **ht**,
[in] double **hth**)

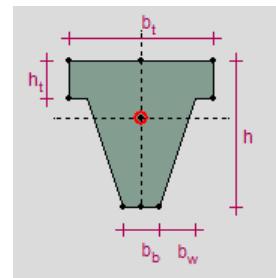
Name name of the new cross-section
bt width at top
bw wall thickness
h height
ht top flange thickness
hth height of the top haunch



Adds a T-shape with haunched top flange to the model.

long **AddTWallHaunched** ([in] BSTR **Name**, [in] double **bt**, [in] double **bb**, [in] double **bw**, [in] double **h**,
[in] double **ht**)

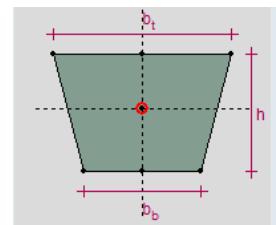
Name name of the new cross-section
bt width at top
bb width at bottom
bw wall haunch
h height
ht top flange thickness



Adds a T-shape with haunched wall to the model.

long **AddTrapezoid** ([in] BSTR **Name**, [in] double **h**, [in] double **bt**, [in] double **bb**)

Name name of the new cross-section
h height
bt width at top
bb width at bottom

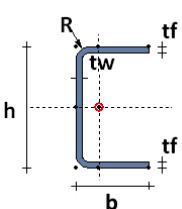


Adds a trapezoid shape to the model.

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

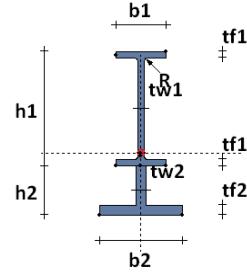
Process the manufacturing process



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)

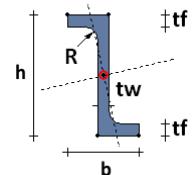
Name name of the new cross-section
h1 upper I-shape height
b1 upper I-shape width
tw1 upper I-shape web thickness
tf1 upper I-shape flange thickness
h2 lower I-shape height
b2 lower I-shape width
tw2 lower I-shape web thickness
tf2 lower I-shape flange thickness
R fillet radius
Process the manufacturing 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 **Edit** ([in] long Index, [in] long CrossSectionBasicType, [in] long CrossSectionShapes)

Index cross-section index

CrossSectionBasicTypes cross-section basic types ([ECrossSectionBasicType](#) value or sum of values)

CrossSectionShapes cross-section shapes ([ECrossSectionShape](#) value or sum of values)

Edits a cross-section in a cross section editor. If successful, returns index of the edited cross-section, otherwise an error ([cseErrorEditing](#), [errDatabaseNotReady](#) or [errIndexOutOfBounds](#)).

long **EditEx** ([in] long Index [in] long CrossSectionBasicTypes,

[in] SAFEARRAY(long) CrossSectionShapes)

Index cross-section index

CrossSectionBasicTypes cross-section types ([ECrossSectionBasicType](#) value or sum of values)

CrossSectionShapes array with enabled cross-section shapes, integer array of type casted ([ECrossSectionShapeEx](#))

Opens Dialog box with cross sections. Adds enabled cross-section(s) from selected catalog file(s) or default catalog files if CatalogFileNames is empty (nil). If successful, returns index of the last added cross section, otherwise an error ([cseErrorAdding](#)).

long **GetDimensionsOfC** ([in] long **Index**, [out] double **h**, [out] double **b**, [out] double **e**,
[out] double **tw**, [out] double **R**),

Index cross-section index
h cross-section height
b cross-section width
e leg height
tw wall thickness
R fillet radius

Get dimensions of a C-shape cross-section.

long **GetDimensionsOfIHaunched** ([in] long **Index**, [out] double **bt**, [out] double **bw**, [out] double **bb**,
[out] double **h**, [out] double **ht**, [out] double **hth**, [out] double **hbh**, [out] double **hb**)

Index cross-section index
bt width at top
bw thickness of the wall
bb width at bottom
h height
ht top flange thickness
hth height of the top haunch
hbh height of the bottom haunch
hb bottom flange thickness

Get dimensions of a haunched I-shape cross-section.

long **GetDimensionsOfTTopHaunched** ([in] long **Index**, [out] double **bt**, [out] double **bw**,
[out] double **h**, [out] double **ht**, [out] double **hth**)

Index cross-section index
bt width at top
bw wall thickness
h height
ht top flange thickness
hth height of the top haunch

Get dimensions of a T-shape with haunched top flange cross-section.

long **GetDimensionsOfTWallHaunched** ([in] long **Index**, [out] double **bt**, [out] double **bb**,
[out] double **bw**, [out] double **h**, [out] double **ht**)

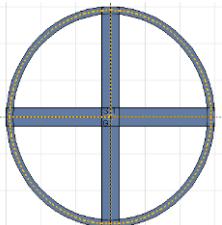
Index cross-section index
bt width at top
bb width at bottom
bw wall haunch
h height
ht top flange thickness

Get dimensions of a T-shape with haunched wall cross-section.

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](#)).

<p>long Merge ([in] long IntoItem, [in] long FromItem, [in] double OffsetY, [in] double OffsetZ, [in] double Rotation, [in] ELongBoolean MergeIntoNew)</p> <p>IntoItem 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.</p>	<p><i>Merges two cross-sections of same cross-section basic type (csbt_Solid or csbt_ThinWalled). If MergeIntoNew is True returns the index of the new cross-section. If MergeIntoNew is False, returns the destination index (IntoItem). In case of an error the result is the error code.</i></p> 
<p>long ReplaceFromCatalog ([in] long Index ,[in] ECrossSectionShape CrossSectionShape, [in] BSTR CrossSectionName)</p> <p>Index cross-section index to replace CrossSectionShape shape of the new cross-section CrossSectionName name of the new cross-section</p>	<p><i>If successful, returns index, otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBounds, errNotFound, errInternalException).</i></p>
<p>long ReplaceFromCatalogEx ([in] long Index ,[in] EcrossSectionShapeEx CrossSectionShape, [in] BSTR CrossSectionName)</p> <p>Index cross-section index to replace CrossSectionShape shape of the new cross-section CrossSectionName name of the new cross-section</p>	<p><i>If successful, returns index, otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBounds, errNotFound, errInternalException).</i></p>
<p>long ReplaceFromCatalogFile ([in] long Index, [in] BSTR CatalogFileName, [in] BSTR CrossSectionName)</p> <p>Index cross-section index to replace CatalogFileName name of the catalog file (e.g.: 'c:\Program Files\Axis VM9\EU_LEQ.SEC') CrossSectionName name of the new cross-section</p>	<p><i>If successful, returns index, otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBounds, errNotFound, errInternalException).</i></p>
<p>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)</p> <p>Index cross-section index 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</p>	<p><i>Replaces the specified cross-section with an asymmetric I-shape.</i></p>

long	ReplaceWithBox ([in] long Index , [in] BSTR Name , [in] double h , [in] double b , [in] double tw , [in] double tf , [in] double R , [in] ECrossSectionProcess Process)
	<p style="margin-left: 20px;">Index cross-section index 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</p> <p style="margin-left: 20px;">Process the manufacturing process</p>
	<i>Replaces the specified cross-section with a box shape.</i>
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)
	<p style="margin-left: 20px;">Index cross-section index 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</p> <p style="margin-left: 20px;">Process the manufacturing process</p>
	<i>Replaces the specified cross-section with a C-shape.</i>
long	ReplaceWithCircle ([in] long Index , [in] BSTR Name , [in] double d)
	<p style="margin-left: 20px;">Index cross-section index to replace Name name of the new cross-section d diameter of the circle</p>
	<i>Replaces the specified cross-section with a circle shape.</i>
long	ReplaceWithCustom ([in] long Index , [in] BSTR Name , [in] AxisVMPolygon2dList* ShapePolygonList , [in] ECrossSectionProcess Process)
	<p style="margin-left: 20px;">Index cross-section index to replace Name name of the new cross-section ShapePolygonList polygon list describing the shape Process the manufacturing process</p>
	<i>Replaces the cross-section with a custom shape.</i>
long	ReplaceWithCircleHollow ([in] long Index , [in] BSTR Name , [in] double d , [in] double t)
	<p style="margin-left: 20px;">Index cross-section index to replace Name name of the new cross-section d circle diameter t thickness</p>
	<i>Replaces the cross-section with a hollow circle shape to the model.</i>
long	ReplaceWithCustomAndUserParams ([in] long Index , [in] BSTR Name , [in] AxisVMPolygon2dList * ShapePolygonList , [in] ECrossSectionProcess Process , [i/o] RCrossSectionUserParams * CrossSectionUserParams)
	<p style="margin-left: 20px;">Index cross-section index to replace Name Name of the cross-section ShapePolygonList Polygon list with shape of the cross-section Process the manufacturing process CrossSectionUserParams User parameters</p>
	<i>Returns index if successful. If not successful, then returns an error code (errIndexOutOfBounds)</i>

long	ReplaceWithCustomAndUserParamsAsArray ([in] long Index , [in] BSTR Name , [in] AxisVMPolygon2dList * ShapePolygonList , [in] ECrossSectionProcess Process , [in] long IParam , [in] SAFEARRAY(double) CrossSectionUserParams)
	Index cross-section index to replace Name Name of the cross-section ShapePolygonList Polygon list with shape of the cross-section Process the manufacturing process IParam User parameter CrossSectionUserPara Array with user parameters (only first 10 elements of the array are used) ms
	<i>Returns index if successful. If not successful, then returns an error code (errIndexOutOfBounds)</i>
long	ReplaceWithCustomAndUserParamsAsArray_vb (Visual Basic compatible function of ReplaceWithCustomAndUserParamsAsArray)
long	ReplaceWithCustomAndUserParamsAsByteArray ([in] long Index , [in] BSTR Name , [in] AxisVMPolygon2dList * ShapePolygonList , [in] ECrossSectionProcess Process , [in] SAFEARRAY(byte) CrossSectionUserParams)
	Index cross-section index to replace Name Name of the cross-section ShapePolygonList Polygon list with shape of the cross-section Process the manufacturing process CrossSectionUserPara Array with user parameters (first element is 32bit long other 10 elements of the array are 64bit doubles) ms
	<i>Returns index if successful. If not successful, then returns an error code (errIndexOutOfBounds)</i>
long	ReplaceWithCustomAndUserParamsAsByteArray_vb (Visual Basic compatible function of ReplaceWithCustomAndUserParamsAsByteArray)
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 index 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
	<i>Replaces the specified cross-section with a double I-shape.</i>
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 index 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
	<i>Replaces the specified cross-section with a double I box shape.</i>

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 index 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 **ReplaceWithDoubleLFlange** ([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 index 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 index 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 index 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 **ReplaceWithIHaunched** ([in] long **Index**, [in] BSTR **Name**, [in] double **bt**, [in] double **bw**,
[in] double **bb**, [in] double **h**, [in] double **ht**, [in] double **hth**, [in] double **hbb**, [in] double **hb**)

Index cross-section index to replace
Name name of the new cross-section
bt width at top
bw thickness of the wall
bb width at bottom
h height
ht top flange thickness
hth height of the top haunch
hbb height of the bottom haunch
hb bottom flange thickness

Adds a haunched I-shape to the model.

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 index 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 **t**,
[in] ECrossSectionProcess **Process**)

Index cross-section index to replace
Name name of the new cross-section
d external diameter of the pipe
t pipe wall thickness
Process the manufacturing process

Replaces the specified cross-section with a pipe.

long **ReplaceWithRectangular** ([in] long **Index**, [in] BSTR **Name**, [in] double **b**, [in] double **h**,
[in] ECrossSectionProcess **Process**)

Index cross-section index to replace
Name name of the new cross-section
b cross-section width
h cross-section height
Process the manufacturing process

Replaces the specified cross-section with a rectangular shape.

long **ReplaceWithRectangularRounded** ([in] long **Index**, [in] BSTR **Name**, [in] double **b**, [in] double **h**,
[in] double **r1**, [in] double **r2**)

Index cross-section index to replace
Name name of the new cross-section
b cross-section width
h cross-section height
r1 fillet radius at top
r2 fillet radius at bottom

Replaces the specified cross-section with a rectangular shape with rounded corners in the model.

long **ReplaceWithRectangularHollow** ([in] long **Index**, [in] BSTR **Name**, [in] double **b**, [in] double **h**,
[in] double **r1**, [in] double **r2**)

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

Replaces the cross-section with a hollow rectangular shape in the model.

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

Index cross-section index to replace
Name name of the new cross-section
N number of polygon sides
Rshape diameter of the circumcircle
t 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 index 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 index 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 cross-section with a T-shape.

long **ReplaceWithTHaunched** ([in] long **Index**, [in] BSTR **Name**, [in] double **bt**, [in] double **bb**,
[in] double **bw**, [in] double **h**, [in] double **ht**)

Index cross-section index to replace
Name name of the new cross-section
bt width at top
bb width at bottom
bw wall haunch
h height
ht top flange thickness

Replaces the cross-section with a T-shape with haunched wall in the model.

long **ReplaceWithTTopHaunched** ([in] long **Index**, [in] BSTR **Name**, [in] double **bt**, [in] double **bw**,
[in] double **h**, [in] double **ht**, [in] double **hth**)

Index cross-section index to replace
Name name of the new cross-section
bt width at top
bw wall thickness
h height
ht top flange thickness
hth height of the top haunch

Replaces the cross-section with a T-shape with haunched top flange to the model.

long **ReplaceWithTrapezoid** ([in] long **Index**, [in] BSTR **Name**, [in] double **h**, [in] double **bt**,
[in] double **bb**)

Index cross-section index to replace
Name name of the new cross-section
h height
bt width at top
bb width at bottom

Replaces the cross-section with a trapezoid shape to the model.

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 index 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 index 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 index 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.

Save results to MetaFile functions

long **SaveToMetaFile** ([in] long **Index**, [in] BSTR **FileName**, [in] BSTR **CreatedBy**,
[in] BSTR **Description**, [in] long **Width**, [in] long **Height**,
[in] ECrossSectionImageExportOptions **Options**)

Index cross-section index
FileName filename with extension emf
CreatedBy the file has been created by
Description descriptions
Width picture's size in pixel
Height picture's size in pixel
Options options

It creates a metafile from the cross-section. It returns CrossSectionId or an error code
(EgeneralError).

long **SaveToBitmapFile** ([in] long **Index**, [in] BSTR **FileName**, [in] long **Width**, [in] long **Height**,
[in] ECrossSectionImageExportOptions **Options**)

Index cross-section index
FileName filename with extension bmp
Width picture's size in pixel
Height picture's size in pixel
Options options

It creates a bitmap file from the cross-section. It returns CrossSectionId or an error code
(EgeneralError).

Properties

long **Count**

Get number of cross-sections in the model.

long **IndexOfUID** [long **UID**] Get index of the cross-section

UID unique index of the cross-section

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

A cross-section of the model by Index. $1 \leq \text{Index} \leq \text{Count}$.

long **UID** [long **Index**] Get unique index of the cross-section which remains the same while exists in the model

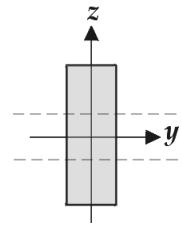
Index index of the cross-section

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
)
```



```
RStressPointParams = (
    double Tx_y   CTx,y stress calculation factor for shear stress along y axis due to Saint-Venant torsion
    double Tx_z   CTx,z stress calculation factor for shear stress along z axis due to Saint-Venant torsion
    double Vy_y   CVy,y stress calculation factor for shear stress along y axis due to shear force along y axis
    double Vy_z   CVy,z stress calculation factor for shear stress along z axis due to shear force along y axis
    double Vz_y   CVz,y stress calculation factor for shear stress along y axis due to shear force along z axis
    double Vz_z   CVz,z stress calculation factor for shear stress along z axis due to shear force along z axis
    double w      Cw stress calculation factor for normal stress due to restrained warping
    double Tw_y   CTw,y stress calculation factor for shear stress along y axis due to restrained warping
    double Tw_z   CTw,z stress calculation factor for shear stress along z axis due to restrained warping
)
```

Functions

long **AddStressPoint** ([i/o] [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 **GetStressPoint** ([in] long Index, [i/o] [RStressPoint](#) Point)

Index stress point index ($0 < \text{Index} \leq \text{StressPointCount}$)
 Point stress point coordinates

Get stress point coordinates by index ($0 < \text{Index} \leq \text{StressPointCount}$). If successful, returns cross-section index, otherwise an error code ([errIndexOutOfBounds](#)).

long **GetStressPointParams** ([in] long Index, [i/o] [RStressPointParams](#) Value)

Index stress point index ($0 < \text{Index} \leq \text{StressPointCount}$)
 Value stress point parameters

Get a stress point parameters by index. If successful, returns cross-section index, otherwise an error code ([errDatabaseNotReady](#), [errIndexOutOfBounds](#), [cseInvalidCrossSectionType](#)).

long	 GetUserParams ([i/o] RCrossSectionUserParams Value)
	Value user parameters
	<i>If successful, returns cross-section index, otherwise an error code (errDatabaseNotReady, errIndexOutOfBounds).</i>
long	 GetUserParamsAsArray ([out] long IParam, [out] SAFEARRAY(double) * Value)
	IParam User parameter (long)
	Value Array with user parameters (only first 10 elements are used)
	<i>If successful, returns cross-section index , otherwise an error code (errDatabaseNotReady, errIndexOutOfBounds).</i>
long	 GetUserParamsAsByteArray ([out] SAFEARRAY(byte) * Value)
	Value Array with user parameters (first element is 32bit integer and another 10 elements are 64bit double)
	<i>If successful, returns cross-section index , otherwise an error code (errDatabaseNotReady, errIndexOutOfBounds).</i>
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
	<i>Mirrors the cross-section to an axis specified by its two points.</i>
	<i>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).</i>
long	 Move ([in] double dy, [in] double dz)
	dy shift in y direction
	dz shift in z direction
	<i>Shifts the cross-section.</i>
	<i>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).</i>
long	 Rotate ([in] double oy, [in] double oz, [in] double alfa)
	oy y distance of the rotation center from center of gravity
	oz z distance of the rotation center from center of gravity
	alfa angle of rotation
	<i>Rotates the cross-section around a rotation center with the specified angle.</i>
	<i>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).</i>
long	 SetStressPoint ([in] long Index, [i/o] RStressPoint Point)
	Index stress point index ($0 < \text{Index} \leq \text{StressPointCount}$)
	Point stress point coordinates
	<i>Set stress point coordinates by index ($0 < \text{Index} \leq \text{StressPointCount}$). If successful, returns cross-section index, otherwise an error code (errIndexOutOfBounds).</i>

long	SetStressPointParams ([in] long Index, [i/o] RStressPointParams Value)
	<p>Index stress point index ($0 < Index \leq StressPointCount$)</p> <p>Value stress point parameters</p> <p>Get a stress point parameters by index. If successful, returns cross-section index, otherwise an error code (errDatabaseNotReady, errIndexOutOfBounds, cselInvalidCrossSectionType, errReadOnly).</p>
long	SetUserParams ([i/o] RCrossSectionUserParams Value)
	<p>Value user parameters</p> <p>If successful, returns cross-section index, otherwise an error code (errReadOnly).</p>
long	SetUserParamsAsArray ([in] long IParam , [in] SAFEARRAY(double)* Value)
	<p>IParam User parameter (long)</p> <p>Value Array with user parameters (only first 10 elements are used)</p> <p>If successful, returns cross-section index, otherwise an error code (errReadOnly).</p>
long	SetUserParamsAsArray_vb (Visual Basic compatible function of SetUserParamsAsArray)
long	SetUserParamsAsByteArray ([in] SAFEARRAY(byte) * Value)
	<p>Value Array with user parameters (first element is 32bit integer and another 10 elements are 64bit double)</p> <p>If successful, returns cross-section index, otherwise an error code (errReadOnly).</p>
long	SetUserParamsAsByteArray_vb (Visual Basic compatible function of SetUserParamsAsByteArray)
long	VerifyProperties
	<p>Verifies cross-section properties.</p> <p>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).</p>

Properties

double	a • (for double shapes) distance of the two shapes [m]
double	Alpha Get rotation angle of parametric shapes. Positive angles are counter-clockwise. [rad]
double	Ax • cross-section area [m^2]
double	Ay • shear area associated with shear forces in local y direction [m^2]
double	Az • shear area associated with shear forces in local z direction [m^2]
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]
ECrossSectionShape	CrossSectionShape cross-section shape (will be deprecated)
ECrossSectionShapeEx	CrossSectionShapeEx cross-section shape (introduced in x5)
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 [m^4]
double	I2 principal inertia about 2nd local axis [m^4]

double	Ialpha Get angle between the local 1st axis and the local y axis [rad]
double	Ix • torsional inertia [m^4]
double	Iy • flexural inertia about local y axis [m^4]
double	Iw • warping modulus [m^6]
double	Iyz • centrifugal inertia [m^4]
double	Iz • flexural inertia about local z axis [m^4]
<u>ELongBoolean</u>	Mirrored • mirrored cross-section
long	N • (for regular polygon shapes) number of polygon sides
BSTR	Name • cross-section name
double	OuterPerimeter Get outer perimeter (boundary perimeter) relative to the lower-left corner
<u>ECrossSectionProcess</u>	Process • manufacturing process
double	r1 • 1. fillet radius [m]
double	r2 • 2. fillet radius [m]
double	r3 • 3. fillet radius [m]
<u>AxisVMPolygon2dList</u>	ShapePolygonList • polygon list describing the shape relative to the lower-left corner of the bounding rectangle
long	StressPointCount • number of stress calculation points
double	tf • cross-section flange thickness [m]
double	tf2 • (for wedged I-shapes) flange thickness for the lower I-shape [m]
double	tw • cross-section web thickness (as t for pipes and regular polygons) [m]
double	tw2 • (for wedged I-shapes) web thickness for the lower I-shape [m]
double	W1b • bottom elastic cross-section modulus for the 1st axis [m^3]
double	W1pl • plastic cross-section modulus for the 1st axis [m^3]
double	W1t • top elastic cross-section modulus for the 1st axis [m^3]
double	W2b • bottom elastic cross-section modulus for the 2st axis [m^3]
double	W2pl • plastic cross-section modulus for the 2nd axis [m^3]
double	W2t • top elastic cross-section modulus for the 2nd axis [m^3]
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	Ys • y coordinate of the shear center relative to the centre of gravity [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]
double	Zs • z coordinate of the shear center relative to the centre of gravity [m]
long	UID Get unique index of the cross-section which remains the same while exists in the model

IAxisVMCrossSectionOptimization

Interface used for optimizing cross-sections both steel and timber.

If property returning this interface is null (nil) then the extension module SD9 or TD9 is not available.

Error codes

```
enum ECrossSectionOptimizationError = {  
    csoGroupNameAlreadyExists = -100001  
    csoGroupsForPredefinedShapes = -100002  
    csoGroupsForParametricOptimization = -100003  
    csoOptimizationChecksNotValid = -100004  
    csoGroupNameInvalid = -100005  
    csoSteelMemberDesignIDsEmpty = -100006  
    csoVariousCrossSectionsAreNotSupported = -100007  
    csoCrossSectionTypeIsNotSupported = -100008  
    csoUsedMaterialIsNotSupported = -100009  
    csoInvalid_b = -100010  
    csoInvalid_h = -100011  
    csoInvalid_tw = -100012  
    csoInvalid_tf = -100013  
    csoInvalid_b2 = -100014  
    csoInvalid_tf2 = -100015  
    csoInvalid_a = -100016  
    csoInvalidManufacturingProcess = -100017  
    csoSteelDesignMemberIDOutOfBounds = -100018  
    csoGroupCrossSectionTypeIsDifferent = -100019  
    csoOptimizationCheckCombinationIsNotValid = -100020  
    csoGroupCrossSectionIndexOutOfBounds = -100021  
    csoOutOfConstraintLimits = -100022  
    csoMaterialVariesInTheGroup = -100023  
    csoCrossSectionVariesInTheGroup = -100024  
    csoManufacturingProcessIsNotSupported = -100025  
    csoCrossSectionsInTheGroupAreNotUsable = -100026  
    csoLoadCaseIDIndexOutOfBounds = -100027  
    csoLoadCombinationIDIndexOutOfBounds = -100028  
    csoInvalidAnalysisType = -100029  
    csoCombinationTypeNotValidForCurrentNationalDesignCode = -100030  
    csoInvalidMaterial = -100031  
    csoInvalidOptimizationType = -100032  
    csoNoneOfTheCrossSectionsIsValid = -100033  
}
```

Enumerated types

```
enum ECrossSectionObjectiveOfOptimization = {  
    cs ooMinimumWeight = 0           optimize for min. weight  
    cs ooMinimumHeight = 1          optimize for min. height  
    cs ooMinimumWidth = 2}         optimize for min. width  
    Objective of optimization  
  
enum ECrossSectionOptimizationType = {  
    cs otPreDefinedShapes = 0       optimize using cross sections in the group  
    cs otParametric = 1            parametric optimization  
}  
    Type of optimization
```

Records / structures

ECrossSectionObjectiveOfOptimization	RCSOptimizationParamsGeneral = (
double	ObjectiveOfOptimization	objective of optimization
double	Constraint_h_min	min. height constraint
double	Constraint_h_max	max. height constraint
double	Constraint_b_min	min. width constraint
double	Constraint_b_max	max. width constraint
	MaximumEfficiency	max. efficiency
ElongBoolean	Custom	if <i>lbTrue</i> then max. number of iterations is considered
long	NumberOflterations	max. number of iterations
long	Beams	Item index in Beams combobox, see AxisVM manual
)	

```

RCSParametricOptimizationParams = (
    General           general parameters of optimization
    fixed_h          if lbTrue then height is fixed
    delta_h          increments in height of the cross-section
    fixed_b          if lbTrue then width is fixed
    delta_b          increments in width of the cross-section
    fixed_tw         if lbTrue then wall thickness is fixed
    delta_tw         increments in wall thickness of the cross-section
    fixed_tf          if lbTrue then flange thickness is fixed
    delta_tf         increments in flange thickness of the cross-section
    fixed_b2         if lbTrue then width b2 is fixed , only for non-symetric I
    b2_min           min. width b2 constraint
    b2_max           max. width b2 constraint
    delta_b2         increments in width b2 of the cross-section, only for non-symetric I
    fixed_tf2        if lbTrue then flange thickness tf2 is fixed , only for non-symetric I
    tf2_min          min. flange thickness tf2 constraint
    tf2_max          max. flange thickness tf2 constraint
    delta_tf2        increments in flange thickness tf2, only for non-symetric I
    fixed_a          if lbTrue then distance of two cross-sections is fixed only for double U
    a_min            min. distance of two cross-sections constraint only for double U
    a_max            max. distance of two cross-sections constraint only for double U
    delta_a          increments in distance of two cross-sections, only for double U
)
)

RCSOptimizationResultsParametric = (
    OptimizationEfficiency   see Optimization in AxisVM manual
    Efficiency               see Optimization in AxisVM manual
    M                        see Optimization in AxisVM manual
    DeltaM                  see Optimization in AxisVM manual
    b                        width, see Optimization in AxisVM manual
    h                        height, see Optimization in AxisVM manual
    tw                       wall thickness, see Optimization in AxisVM manual
    tf                       flange thickness, see Optimization in AxisVM manual
    b2                      width b2, see Optimization in AxisVM manual
    tf2                     flange thickness tf2, see Optimization in AxisVM manual
    a                        distance, see Optimization in AxisVM manual
)
)

RCSOptimizationResultsPredefinedShapes = (
    OptimizationEfficiency   see Optimization in AxisVM manual
    Efficiency               see Optimization in AxisVM manual
    M                        see Optimization in AxisVM manual
    DeltaM                  see Optimization in AxisVM manual
    GroupCrossSection        index of the cross-section in the optimization group
)
)

```

Functions

long **AddGroup** ([in] BSTR Name, [in] ECrossSectionOptimizationType OptimizationType,
[in] SAFEARRAY(long) * MemberDesignIDs)

Name name of the group
OptimizationType optimization type
MemberDesignIDs indexes of [SteelDesignMembers](#) or [TimberDesignMembers](#)
depends on the used property of [IAxisVMModel](#)

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

long **AddGroup_vb** ([in] BSTR Name, [in] ECrossSectionOptimizationType OptimizationType,
[in] SAFEARRAY(long) * MemberDesignIDs)

Visual basic compatible version of function **AddGroup**.

long **AddCrossSectionFromModel** ([in] long GroupIndex, [in] long CrossSectionIndex)

GroupIndex index of the group
CrossSectionIndex index of cross-section in the model,
 $1 \leq \text{CrossSectionIndex} \leq \text{IAxisVMCrossSections.count}$

Adds a cross-section to an optimization group added to the model. If successful, returns index of the cross-section in the group (GroupCrossSectionIndex), otherwise an error code.

long	AddCrossSectionFromCatalog ([in] long GroupIndex , [in] BSTR CrossSectionName)	
	GroupIndex	<i>index of the group</i>
	CrossSectionName	<i>name of the cross-section used in catalog (must be the same shape as shape used in the group)</i>
<i>Adds a cross-section to an optimization group from a catalog. If successful, returns index of the cross-section in the group (GroupCrossSectionIndex), otherwise an error code.</i>		
long	AddCrossSectionFromDialog ([in] long GroupIndex)	
	GroupIndex	<i>index of the group</i>
<i>Adds a cross-section to an optimization group a dialog. If successful, returns index of the cross-section in the group (GroupCrossSectionIndex), otherwise an error code.</i>		
long	DeleteGroup ([in] long GroupIndex)	
	GroupIndex	<i>index of the group</i>
<i>Delete a optimization group. If successful, returns GroupIndex, otherwise an error code.</i>		
long	DeleteGroupCrossSection ([in] long GroupIndex , [in] long GroupCrossSectionIndex)	
	GroupIndex	<i>index of the group</i>
	GroupCrossSectionIndex	<i>index of the cross-section in the group</i>
<i>Delete a group cross-section from optimization group. If successful, returns GroupCrossSectionIndex, otherwise an error code.</i>		
long	GetGroupCrossSections ([in] long GroupIndex ,	
	[out] SAFEARRAY(long) * ModelCrossSectionIDs , [out] SAFEARRAY(BSTR) * TableNames ,	
	[out] SAFEARRAY(BSTR) * CrossSectionNames)	
	GroupIndex	<i>index of the group</i>
	ModelCrossSectionIDs	<i>index of cross-section in the model</i>
	TableNames	<i>table name of cross-section in the group</i>
	CrossSectionNames	<i>name of the cross-section in the group</i>
<i>Get group cross sections of the optimization group. If successful, returns length of each array, otherwise an error code.</i>		
long	GetOptimizationChecks ([in] long GroupIndex , [out] ElongBoolean Strength ,	
	[out] ElongBoolean FlexuralBuckling , [out] ElongBoolean LateralTorsionalBuckling ,	
	[out] ElongBoolean WebBuckling)	
	GroupIndex	<i>index of the group</i>
	Strength	<i>strength considered in optimization</i>
	FlexuralBuckling	<i>flexural buckling considered in optimization</i>
	LateralTorsionalBuckling	<i>lateral torsional buckling considered in optimization</i>
	WebBuckling	<i>web buckling considered in optimization</i>
<i>Get optimization checks of the optimization group. If successful, returns GroupIndex, otherwise an error code.</i>		
long	GetParametersForPredefinedShapes ([in] long GroupIndex ,	
	[i/o] RCSOptimizationParamsGeneral GeneralParams)	
	GroupIndex	<i>index of the group</i>
	GeneralParams	<i>general parameters for optimization of pre-defined shapes</i>
<i>Get optimization parameters of the optimization group. If successful, returns GroupIndex, otherwise an error code.</i>		

long	GetParametersForParametricOptimization ([in] long GroupIndex , [i/o] RCSParametricOptimizationParams ParametricParams)
	GroupIndex <i>index of the group</i>
	ParametricParams parameters for parametric optimization
	<i>Get optimization parameters of the optimization group. If successful, returns GroupIndex, otherwise an error code.</i>
long	GetMemberDesignIDs ([in] long GroupIndex , [out] SAFEARRAY(long) * MemberDesignIDs)
	GroupIndex <i>index of the group</i>
	MemberDesignIDs indexes of SteelDesignMembers or TimberDesignMembers <i>depends on the used property of IAxisVMModel</i>
	<i>Get design members indexes of the optimization group. If successful, returns GroupIndex, otherwise an error code.</i>
long	GetParametricShapeOptimizationResultsByLoadCaseId ([in] long GroupIndex , [in] long LoadCaseID , [in] long LoadLevel , [in] EAnalysisType AnalysisType , [i/o] RCSOptimizationResultsParametric ParametricResults)
	GroupIndex <i>index of the group</i>
	LoadCaseID <i>index of the load case</i>
	LoadLevel <i>load level</i>
	AnalysisType <i>type of analysis</i>
	ParametricResults <i>results of parametric optimization</i>
	<i>Get parametric optimization results of the optimization group. If successful, returns GroupIndex, otherwise an error code.</i>
long	GetParametricShapeOptimizationResultsByLoadCombinationId ([in] long GroupIndex , [in] long LoadCombinationId , [in] long LoadLevel , [in] EAnalysisType AnalysisType , [i/o] RCSOptimizationResultsParametric ParametricResults)
	GroupIndex <i>index of the group</i>
	LoadCombinationId <i>index of the load combination</i>
	LoadLevel <i>load level</i>
	AnalysisType <i>type of analysis</i>
	ParametricResults <i>results of parametric optimization</i>
	<i>Get parametric optimization results of the optimization group. If successful, returns GroupIndex, otherwise an error code.</i>
long	GetEnvelopeParametricShapeOptimizationResults ([in] long GroupIndex , [in] long EnvelopeUID , [in] EAnalysisType AnalysisType , [i/o] RCSOptimizationResultsParametric ParametricResults)
	GroupIndex <i>index of the group</i>
	EnvelopeUID <i>unique index of the envelope</i>
	AnalysisType <i>type of analysis</i>
	ParametricResults <i>results of parametric optimization</i>
	<i>Get parametric optimization results of the optimization group. If successful, returns GroupIndex, otherwise an error code.</i>

long	GetCriticalParametricShapeOptimizationResults ([in] long GroupIndex , [in] ECombinationType CombinationType , [in] EAnalysisType AnalysisType , [i/o] RCSOptimizationResultsParametric ParametricResults)
	<p style="margin-left: 20px;">GroupIndex <i>index of the group</i></p> <p style="margin-left: 20px;">CombinationType <i>combination type</i></p> <p style="margin-left: 20px;">AnalysisType <i>type of analysis</i></p> <p style="margin-left: 20px;">ParametricResults <i>results of parametric optimization</i></p> <p><i>Get parametric optimization results of the optimization group. If successful, returns GroupIndex, otherwise an error code.</i></p>
long	GetPredefinedShapesOptimizationResultsByLoadCaseId ([in] long GroupIndex , [in] long LoadCaseID , [in] long LoadLevel , [in] EAnalysisType AnalysisType , [out] SAFEARRAY(RCSOptimizationResultsPredefinedShapes) * PredefinedShapesResults)
	<p style="margin-left: 20px;">GroupIndex <i>index of the group</i></p> <p style="margin-left: 20px;">LoadCaseID <i>index of the load case</i></p> <p style="margin-left: 20px;">LoadLevel <i>load level</i></p> <p style="margin-left: 20px;">AnalysisType <i>type of analysis</i></p> <p style="margin-left: 20px;">PredefinedShapesResults <i>array with optimization results of pre-defined shapes</i></p> <p><i>Get optimization results of pre-defined shapes. If successful, returns PredefinedShapesResults array index of most optimal cross-section, otherwise an error code.</i></p>
long	GetPredefinedShapesOptimizationResultsByLoadCombinationId ([in] long GroupIndex , [in] long LoadCombinationId , [in] long LoadLevel , [in] EAnalysisType AnalysisType , [out] SAFEARRAY(RCSOptimizationResultsPredefinedShapes) * PredefinedShapesResults)
	<p style="margin-left: 20px;">GroupIndex <i>index of the group</i></p> <p style="margin-left: 20px;">LoadCombinationId <i>index of the load combination</i></p> <p style="margin-left: 20px;">LoadLevel <i>load level</i></p> <p style="margin-left: 20px;">AnalysisType <i>type of analysis</i></p> <p style="margin-left: 20px;">PredefinedShapesResults <i>array with optimization results of pre-defined shapes</i></p> <p><i>Get optimization results of pre-defined shapes. If successful, returns PredefinedShapesResults array index of most optimal cross-section, otherwise an error code.</i></p>
long	GetEnvelopePredefinedShapesOptimizationResults ([in] long GroupIndex , [in] long EnvelopeUID , [in] EAnalysisType AnalysisType , [out] SAFEARRAY(RCSOptimizationResultsPredefinedShapes) * PredefinedShapesResults)
	<p style="margin-left: 20px;">GroupIndex <i>index of the group</i></p> <p style="margin-left: 20px;">EnvelopeUID <i>unique index of the envelope</i></p> <p style="margin-left: 20px;">AnalysisType <i>type of analysis</i></p> <p style="margin-left: 20px;">PredefinedShapesResults <i>array with optimization results of pre-defined shapes</i></p> <p><i>Get optimization results of pre-defined shapes. If successful, returns PredefinedShapesResults array index of most optimal cross-section, otherwise an error code.</i></p>
long	GetCriticalPredefinedShapesOptimizationResults ([in] long GroupIndex , [in] ECombinationType CombinationType , [in] EAnalysisType AnalysisType , [out] SAFEARRAY(RCSOptimizationResultsPredefinedShapes) * PredefinedShapesResults)
	<p style="margin-left: 20px;">GroupIndex <i>index of the group</i></p> <p style="margin-left: 20px;">CombinationType <i>combination type</i></p> <p style="margin-left: 20px;">AnalysisType <i>type of analysis</i></p> <p style="margin-left: 20px;">PredefinedShapesResults <i>array with optimization results of pre-defined shapes</i></p> <p><i>Get optimization results of pre-defined shapes. If successful, returns PredefinedShapesResults array index of most optimal cross-section, otherwise an error code.</i></p>

long	PredefinedShapesOptimizationResultsByLoadCaseId ([in] long GroupIndex , [out] SAFEARRAY(RCSOptimizationResultsPredefinedShapes) * PredefinedShapesResults)
	GroupIndex index of the group PredefinedShapesResults array with optimization results of pre-defined shapes Get optimization results of pre-defined shapes. If successful, returns PredefinedShapesResults array index of most optimal cross-section, otherwise an error code.
long	PredefinedShapesOptimizationResultsByLoadCombinationId ([in] long GroupIndex , [out] SAFEARRAY(RCSOptimizationResultsPredefinedShapes) * PredefinedShapesResults)
	GroupIndex index of the group PredefinedShapesResults array with optimization results of pre-defined shapes Get optimization results of pre-defined shapes. If successful, returns PredefinedShapesResults array index of most optimal cross-section, otherwise an error code.
long	EnvelopePredefinedShapesOptimizationResults ([in] long GroupIndex , [out] SAFEARRAY(RCSOptimizationResultsPredefinedShapes) * PredefinedShapesResults)
	GroupIndex index of the group PredefinedShapesResults array with optimization results of pre-defined shapes Get optimization results of pre-defined shapes. If successful, returns PredefinedShapesResults array index of most optimal cross-section, otherwise an error code.
long	CriticalPredefinedShapesOptimizationResults ([in] long GroupIndex , [out] SAFEARRAY(RCSOptimizationResultsPredefinedShapes) * PredefinedShapesResults)
	GroupIndex index of the group PredefinedShapesResults array with optimization results of pre-defined shapes Get optimization results of pre-defined shapes. If successful, returns PredefinedShapesResults array index of most optimal cross-section, otherwise an error code.
long	ParametricShapeOptimizationResultsByLoadCaseId ([in] long GroupIndex , [i/o] RCSOptimizationResultsParametric ParametricResults)
	GroupIndex index of the group ParametricResults results of parametric optimization Get parametric optimization results of the optimization group. If successful, returns GroupIndex , otherwise an error code.
long	ParametricShapeOptimizationResultsByLoadCombinationId ([in] long GroupIndex , [i/o] RCSOptimizationResultsParametric ParametricResults)
	GroupIndex index of the group ParametricResults results of parametric optimization Get parametric optimization results of the optimization group. If successful, returns GroupIndex , otherwise an error code.
long	EnvelopeParametricShapeOptimizationResults ([in] long GroupIndex , [i/o] RCSOptimizationResultsParametric ParametricResults)
	GroupIndex index of the group ParametricResults results of parametric optimization Get parametric optimization results of the optimization group. If successful, returns GroupIndex , otherwise an error code.
long	CriticalParametricShapeOptimizationResults ([in] long GroupIndex , [i/o] RCSOptimizationResultsParametric ParametricResults)
	GroupIndex index of the group ParametricResults results of parametric optimization Get parametric optimization results of the optimization group. If successful, returns GroupIndex , otherwise an error code.

long	ReplaceCrossSectionToGroupCrossSection ([in] long GroupIndex , [in] long GroupCrossSectionIndex ,)
	GroupIndex <i>index of the group</i>
	GroupCrossSectionIndex <i>index of the cross-section in the group</i>
	<i>Replace assigned cross-section(s) of design members in the group with a cross-section from the group. If successful, returns GroupCrossSectionIndex, otherwise an error code.</i>
long	ReplaceCrossSectionToModelCrossSection ([in] long GroupIndex , [in] long ModelCrossSectionIndex)
	GroupIndex <i>index of the group</i>
	ModelCrossSectionIndex <i>index of the cross-section in the model</i>
	<i>Replace assigned cross-section(s) of design members in the group with cross-section in the model. If successful, returns GroupCrossSectionIndex, otherwise an error code.</i>
long	SetOptimizationChecks ([in] long GroupIndex , [in] <u>ElongBoolean</u> Strength , [in] <u>ElongBoolean</u> FlexuralBuckling , [in] <u>ElongBoolean</u> LateralTorsionalBuckling , [in] <u>ElongBoolean</u> WebBuckling)
	GroupIndex <i>index of the group</i>
	Strength <i>strength considered in optimization</i>
	FlexuralBuckling <i>flexural buckling considered in optimization</i>
	LateralTorsionalBuckling <i>lateral torsional buckling considered in optimization</i>
	WebBuckling <i>web buckling considered in optimization</i>
	<i>Set optimization checks of the optimization group. If successful, returns GroupIndex, otherwise an error code.</i>
long	SetParametersForPredefinedShapes ([in] long GroupIndex , [i/o] <u>RCSOptimizationParamsGeneral</u> GeneralParams)
	GroupIndex <i>index of the group</i>
	GeneralParams <i>general parameters for optimization of pre-defined shapes</i>
	<i>Set optimization parameters of the optimization group. If successful, returns GroupIndex, otherwise an error code.</i>
long	SetParametersForParametricOptimization ([in] long GroupIndex , [i/o] <u>RCSParametricOptimizationParams</u> ParametricParams)
	GroupIndex <i>index of the group</i>
	ParametricParams <i>parameters for parametric optimization</i>
	<i>Get optimization parameters of the optimization group. If successful, returns GroupIndex, otherwise an error code.</i>
long	SetMemberDesignIDs ([in] long GroupIndex , [in] SAFEARRAY(long) * MemberDesignIDs)
	GroupIndex <i>index of the group</i>
	SteelMemberDesignIDs <i>indexes of <u>SteelDesignMembers</u> or <u>TimberDesignMembers</u> depends on the used property of <u>IAxisVMMModel</u></i>
	<i>Set steel design members indexes of the optimization group. If successful, returns GroupIndex, otherwise an error code.</i>
long	SetMemberDesignIDs_vb ([in] long GroupIndex , [in] SAFEARRAY(long) * MemberDesignIDs)
	<i>Visual basic compatible version of function SetMemberDesignIDs.</i>

Properties

<u>EAnalysisType</u>	AnalysisType • Get or set the type of analysis
<u>ELongBoolean</u>	CallMainProgress • Get or set whether MainProgress event should be called
<u>ECombinationType</u>	CombinationType • Get or set the type of combination for critical results
long	EnvelopeUID • Get or set the unique index of the envelope used in functions for reading envelope results
long	GroupCount Get number of optimization groups
long	GroupCrossSectionCount [long Index] Get number of cross-sections in the group
<u>ECrossSectionShape</u>	GroupCrossSectionShape [long Index] Get cross-section shape of the group (will be deprecated with CrossSectionShapeEx)
<u>ECrossSectionShapeEx</u>	CrossSectionShapeEx [long Index] Get cross-section shape of the group
BSTR	GroupName [long Index] Get name of the group
long	LoadCaseId • Get or set the load case index $(0 < LoadCaseId \leq \text{AxisVMLoadCases.Count})$
long	LoadCombinationId • Get or set the load combination index $(0 < LoadCombinationId \leq \text{AxisVMLoadCombinations.Count})$
long	LoadLevel • Get or set the load level (increment) index
<u>ECrossSectionOptimizationType</u>	OptimizationType [long Index] Get optimization type of the group

IAxisVMCustomParts



Interface used for editing existing user-defined (custom) parts in the model. See Parts in AxisVM.

Important note:

New user-defined (custom) parts can be created in [IAxisVMCustomPartFolder](#)

Enumerated types

```
enum ESelectMode = {
    smSelect = 0x0,      select elements of the part
    smDeselect =        unselect elements of the part
    0x1,
    smInvert = 0x2 }    invert selection status of the part
    Selection mode of the part
```

```
enum EPartItemType = {
    pitNode = 0x0,      node element
    pitLine = 0x1,      line element
    pitSurface = 0x2,   surface element
    pitDomain = 0x3 }   domain element
}
Type of the part item
```

Error codes

enum ECustomePartsError = {	
cpeErrorRenamingPartFolder = -100001	Error whole renaming part folder, folder with same name exists
cpePartNameAlreadyExists = -100002	Error whole renaming part, part with same name exists
cpeInvalidPath = -100003	Path is invalid or doesn't exist
}	

Records / structures

<u>EPartItemType</u>	<u>RPartItem</u> = (
	<u>ItemTye</u>	Type of the element in the part
long	<u>Id</u>	Index of the element in the part
)	

Functions

long **AddPartItemsFromSelectedItemsToPart** ([in] long **PartID**)

PartID index of the custom part

Add selected elements to the existing custom part. Returns PartID if successful, otherwise returns an error code ([errDatabaseNotReady](#), [errIndexOutOfBoundsException](#))

long **AddPartItemsToPart** ([in] long **PartID**, [in] SAFEARRAY(RPartItem)* **PartItems**)

PartID index of the custom part

PartItems custom part items

Add part items to the existing custom part. Returns number of added part items if successful, otherwise returns an error code ([errDatabaseNotReady](#), [errIndexOutOfBoundsException](#))

long **DeletePart** ([in] long **PartID**)

PartID index of the custom part

Delete custom part. Returns PartID if successful, otherwise returns an error code ([errDatabaseNotReady](#), [errIndexOutOfBoundsException](#))

IAxisVMCustomPartFolder	GetCustomPartFolderByPath ([in] BSTR FullPath) FullPath <i>Full path of the custom part folder</i> <i>Get custom part folder from the full path. Returns custom part folder if successful, otherwise null.</i>
long	GetPart ([in] long PartID, [out] SAFEARRAY(RPartItem)* PartItems) PartID <i>index of the custom part</i> PartItems <i>custom part items</i> <i>Get part items of the existing custom part. Returns number of part items if successful, otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBoundsException)</i>
long	GetPartItemsByUID ([in] long PartUID, [i/o] SAFEARRAY(RPartItem)* PartItems) PartUID <i>unique index of the custom part</i> PartItems <i>custom part items</i> <i>Get part items of the existing custom part. Returns number of part items if successful, otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBoundsException)</i>
long	ModifyPart ([in] long PartID, [in] SAFEARRAY(RPartItem)* PartItems) PartID <i>index of the custom part</i> PartItems <i>custom part items</i> <i>Modify part items in the existing custom part. Returns number of part items if successful, otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBoundsException)</i>
long	ModifyPartFromSelectedItems ([in] long PartID) PartID <i>index of the custom part</i> <i>Replace part items in the existing custom part with selected element. Returns PartID if successful, otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBoundsException)</i>
long	SelectPartItems ([in] long PartID, [in] ESelectMode SelectMode) PartID <i>index of the custom part</i> SelectMode <i>Selection mode</i> <i>Select part items of the existing custom part based on selection mode. Returns PartID if successful, otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBoundsException)</i> <i>*** Call Refresh function afterwards if not called between functions BeginUpdate and EndUpdate</i>

Properties

AxisVMAssignments*	Attachments <i>Get the attachments interface</i>
long	Count <i>Get total number of custom parts in the model.</i>
ELongBoolean	IsCustomPart [long PartUID] • <i>Is lbTrue if it is a custom part</i> PartUID <i>Unique index of the custom part</i>
long	IndexOfUID [long PartUID] • <i>Get index of the custom part</i> PartUID <i>Unique index of the custom part</i>
BSTR	Name [long PartID] • <i>Get or set name of the custom part</i> PartID <i>index of the custom part</i>
BSTR	FullName [long PartID] • <i>Get or set full name of the custom part</i> PartID <i>index of the custom part</i>
IAxisVMCustomPartFolder	RootFolder <i>Name of the custom part root folder</i>

long **PartUID** [|long **PartID**] *Get unique index of the custom part which remains static in the mode, same as PartUID in IAxisVMCustomPartFolder*
PartID *index of the custom part*

IAxisVMCustomPartFolder



Interface used for defining user-defined (custom) part folders for user-defined (custom) parts and creating new user-defined in the model. See Parts in AxisVM

Enumerated types

enum	EPartControl = {
	pcDeleteParts = 0x0,
	pcMoveToUpperFolder = 0x1,
	pcMoveToRootFolder = 0x2 }
<i>Part control after subfolder delete</i>	

Functions

long **AddPart** ([in] BSTR Name, [in] SAFEARRAY([RPartItem](#))* PartItems)

Name *name of the custom part*

PartItems *custom part items*

Create new custom part with part items in the current custom part folder. Returns PartIndex if successful, otherwise returns an error code ([errDatabaseNotReady](#))

long **AddPartFromSelectedItems** ([in] BSTR Name)

Name *name of the custom part*

Create new custom part from selected elements in the current custom part folder. Returns PartIndex if successful, otherwise returns an error code ([errDatabaseNotReady](#))

long **AddPartItemsFromSelectedItemsToPart** ([in] long PartIndex)

PartIndex *index of the custom part in the current custom part folder
(0 < PartIndex ≤ [PartCount](#))*

Add selected elements to the existing custom part. Returns PartIndex if successful, otherwise returns an error code ([errDatabaseNotReady](#), [errIndexOutOfBounds](#))

long **AddPartItemsToPart** ([in] long PartIndex, [in] SAFEARRAY([RPartItem](#))* PartItems)

PartIndex *index of the custom part in the current custom part folder
(0 < PartIndex ≤ [PartCount](#))*

PartItems *custom part items*

Add part items to the existing custom part. Returns number of added part items if successful, otherwise returns an error code ([errDatabaseNotReady](#), [errIndexOutOfBounds](#))

long **AddSubFolder** ([in] BSTR Name)

Name *name of the custom part subfolder*

Create new custom part subfolder in the current custom part folder. Returns index of the custom part subfolder if successful, otherwise returns an error code ([errDatabaseNotReady](#), [errCOMServerInternalError](#))

long **GetPart** ([in] long PartIndex, [out] SAFEARRAY([RPartItem](#))* PartItems)

PartIndex *index of the custom part in the current custom part folder
(0 < PartIndex ≤ [PartCount](#))*

PartItems *custom part items*

Get part items of the existing custom part. Returns number of part items if successful, otherwise returns an error code ([errDatabaseNotReady](#), [errIndexOutOfBounds](#))

long **DeletePart** ([in] long PartIndex)

PartIndex *index of the custom part in the current custom part folder
(0 < PartIndex ≤ [PartCount](#))*

Deletes custom part. Returns custom part index if successful, otherwise returns an error code ([errDatabaseNotReady](#), [errIndexOutOfBounds](#))

long	DeleteSubFolder ([in] long SubFolderIndex , [in] EPartControl PartControl)
	SubFolderIndex <i>index of the custom part subfolder ($0 < SubFolderIndex \leq SubFolderCount$)</i>
	PartControl <i>Part control after subfolder delete</i>
	<i>Deletes custom part subfolder. Returns SubFolderIndex if successful, otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBounds)</i>
long	ModifyPart ([in] long PartIndex , [in] SAFEARRAY(RPartItem)* PartItems)
	PartIndex <i>index of the custom part in the current custom part folder ($0 < PartIndex \leq PartCount$)</i>
	PartItems <i>custom part items</i>
	<i>Replace part items of the existing custom part with new part items. Returns custom part index if successful, otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBounds)</i>
long	ModifyPartFromSelectedItems ([in] long PartIndex)
	PartIndex <i>index of the custom part in the current custom part folder ($0 < PartIndex \leq PartCount$)</i>
	<i>Replace part items of the existing custom part with selected elements. Returns custom part index if successful, otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBounds)</i>
long	RenamePart ([in] long PartIndex , [in] BSTR Name)
	PartIndex <i>index of the custom part in the current custom part folder</i>
	Name <i>name of the custom part</i>
	<i>Rename custom part. Returns custom part index if successful, otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBounds)</i>
long	RenameSubFolder ([in] long SubFolderIndex , [in] BSTR Name)
	SubFolderIndex <i>index of the custom part subfolder ($0 < SubFolderIndex \leq SubFolderCount$)</i>
	Name <i>name of the custom part subfolder</i>
	<i>Rename custom part subfolder. Returns SubFolderIndex if successful, otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBounds)</i>
long	SelectPartItems ([in] long PartIndex , [in] ESelectMode SelectMode)
	PartIndex <i>index of the custom part in the current custom part folder ($0 < PartIndex \leq PartCount$)</i>
	SelectMode <i>Selection mode</i>
	<i>Select part items of the existing custom part based on selection mode. Returns index of the custom part if successful, otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBounds)</i>
	<i>NOTE: Call Refresh function afterwards if not called between functions BeginUpdate and EndUpdate</i>

Properties

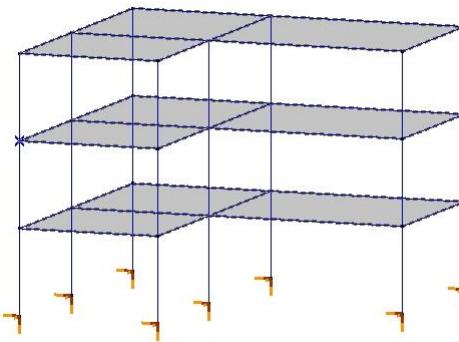
IAxisVMCustomPartFolder	EBoolean IsRootFolder <i>Is lbTrue if the custom part folder is the root folder</i>
	BSTR Name • <i>Get or set name of the custom part folder</i>
	IAxisVMCustomPartFolder ParentFolder <i>Get the custom part folder's parent folder</i>
	long PartCount <i>Get number of custom parts in the current custom part folder</i>
	long PartId [long PartIndex] <i>Get PartID of the custom part, used in IAxisVMCustomParts</i>
	PartIndex <i>index of the custom part in the current custom part folder</i>
	long PartUID [long PartIndex] <i>Get unique index of the custom part which remains the same while exists in the model</i>
	PartIndex <i>index of the custom part in the current custom part folder</i>
	BSTR Path <i>Path of the custom part folder, folders separated with TAB character</i>
IAxisVMCustomPartFolder	SubFolder [long SubFolderIndex] <i>Get the custom part subfolder</i>
	SubFolderIndex <i>index of the custom part subfolder ($0 < SubFolderIndex \leq SubFolderCount$)</i>

long **SubFolderCount**

Get number of custom part folders in the current custom part folder

IAxisVMDiaphragm

Interface used for defining diaphragms in the model.



Error codes

enum	EDiaphragmError = {	
	dpheLineListIsEmpty = -100001	array with LineIDs is empty
	dpheNoLinesAreSelected = -100002	when none of the lines is selected
	dpheLineIndexOutOfBounds = -100003	when LineID is out bounds
	dpheIllegalDOFValue = -100004}	DOF value is not valid

Functions

long	Add ([in] long Dof , [in] SAFEARRAY(long) LineIDs)	
	Dof Degree of freedom see: EDegreeOfFreedom	
	LineIDs Array with LineIDs	
	<i>If successful returns number of diaphragms, otherwise returns an error code (errDatabaseNotReady, dpheIllegalDOFValue, dpheLineListIsEmpty, dpheLineIndexOutOfBounds)</i>	
long	Add_vb (Visual Basic compatible function of Add)	
long	AddSelectedLines ([in] long Dof)	
	Dof Degree of freedom see: EDegreeOfFreedom	
	<i>If successful returns number of diaphragms, otherwise returns an error code (errDatabaseNotReady, dpheIllegalDOFValue, dpheNoLinesAreSelected)</i>	
long	Clear	
	<i>If successful returns number of diaphragms before delete, otherwise returns an error code (errDatabaseNotReady)</i>	
long	Delete ([in] long Index)	
	Index index of the diaphragm, $1 \leq \text{Index} \leq \text{Count}$	
	<i>If successful returns Index, otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBounds)</i>	
long	GetLines ([in] long Index , [out] SAFEARRAY(long) LineIDs)	
	Index index of the diaphragm, $1 \leq \text{Index} \leq \text{Count}$	
	LineIDs Array with LineIDs	
	<i>If successful returns number of lines in diaphragm with index Index, otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBounds)</i>	
long	GetDOF ([in] long Index , [out] long Dof)	
	Index index of the diaphragm, $1 \leq \text{Index} \leq \text{Count}$	
	Dof Degree of freedom see: EDegreeOfFreedom	
	<i>If successful returns Index, otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBounds)</i>	

long **RemoveLinesFromDiaphragm** ([in] SAFEARRAY(long) **LineIDs**)
LineIDs Array with LineIDs
*If successful returns number of diaphragms, otherwise returns an error code
(errDatabaseNotReady, dpheLineIndexOutOfBounds, dpheLineListIsEmpty)*

long **RemoveLinesFromDiaphragm_vb** (Visual Basic compatible function of
RemoveLinesFromDiaphragm)

Properties

long **Count**
Get number of diaphragms in the model.

IAxisVMDimensions

AxisVM dimensions

Enumerated types

enum	EDimensionType = {	
	dtOrtho = 0x0,	orthogonal dimension line
	dtAligned = 0x1,	aligned dimension line
	dtAngle = 0x2,	angle dimension (NOT SUPPORTED YET)
	dtArcLength = 0x3,	arc length dimension (NOT SUPPORTED YET)
	dtCoordinate = 0x4,	coordinate dimension (NOT SUPPORTED YET)
	dtLevel = 0x5,	level marks (NOT SUPPORTED YET)
	dtElevation = 0x6,	elevation marks (NOT SUPPORTED YET)
	dtTextBox = 0x7,	text box
	dtNodeAssoc = 0x8,	information associated with a node (NOT SUPPORTED YET)
	dtLineAssoc = 0x9,	information associated with a line (NOT SUPPORTED YET)
	dtDomainAssoc = 0x10,	information associated with a domain (NOT SUPPORTED YET)
	dtLineIntegrated = 0x11,	integrated value on a line (NOT SUPPORTED YET)
	dtLowLevel = 0x12,	dimension given by four points (NOT SUPPORTED YET)
	dtIsoLabel = 0x13,	dimension's label (NOT SUPPORTED YET)
	dtNone = 0x14,	only logical (NOT SUPPORTED YET)
	dtInfoHint = 0x15,	element info hints (NOT SUPPORTED YET)
	dtRadius = 0x16},	radius dimension (NOT SUPPORTED YET)

NOTE. The not supported types cannot be added and their parameters cannot be read.

enum	EDimensionLabelOrientation = {	
	dloAutomatic = 0x0,	automatic
	dloHorizontal = 0x1,	horizontal
	dloVertical = 0x2,	vertical
	dloAligned = 0x3,	aligned
	dloRadial = 0x4,	radial
	dloTangential = 0x5,	tangential
	dloLeft = 0x6,	positioning to left
	dloCentre = 0x7,	positioning to centre
	dloRight = 0x8}	positioning to right

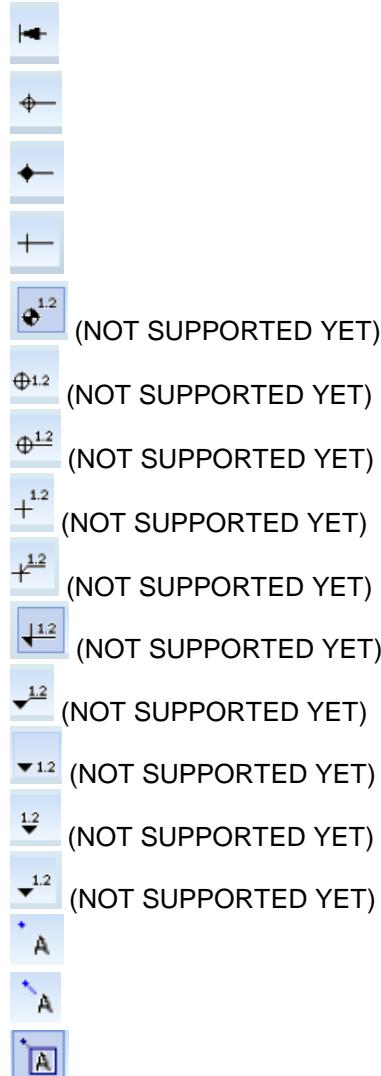
enum	EDimensionStyle = {	
	dsNormalTick = 0x00,	
	dsBoldTick = 0x01,	
	dsArrow = 0x02,	
	dsTriangle = 0x03,	
	dsTriangleStr = 0x04,	



```

dsTriangleFilled= 0x05,
dsCircleStr= 0x06,
dsCircleFilled= 0x07,
dsPlus= 0x08,
dsLevelCircleCheck= 0x09,
dsLevelCircle= 0x10,
dsLevelCircleExt= 0x11,
dsLevelCross= 0x12,
dsLevelCrossExt= 0x13,
dsElevCorner= 0x14,
dsElevTriangleExt= 0x15,
dsElevTriangleRight= 0x16,
dsElevTriangleTop= 0x17,
dsElevTriangleTopRight= 0x18,
dsNothing= 0x19,
dsExtLine= 0x20,
dsInBox= 0x21}

```



NOTE. The not supported types cannot be added and their parameters cannot be read.

Error codes

```

enum EDimensionsError = {
    deINVALIDText= -100001 , text is not valid
    deINVALIDType= -100002 , type is not valid
    deINVALIDMarkType= -100003 , Marktype is not valid
    deINVALIDLayerID= -100004, Layer Id is not valid
    deINVALIDNodeID= -100005} Node Id is not valid

```

Records / structures

ELongBoolean	RDimensionLineParameters = (
EGlobalWorkplaneType	<i>Orthogonal</i>
EAxis	<i>orthogonal</i>
	Plane
	<i>plane</i>
	OrthoDirection
	<i>orthogonal direction</i>
NodelD1	start node's index
NodelD2	endnode's index
LayerID	layer's index
LabelDistance	distance
LabelInside	label is inside/outside
LabelOrientation	label's orientation
DimensionStyle	dimension's style
ELongBoolean	DisplayUnit
	<i>display unit</i>
)
	RTextBoxParameters = (
RPoint3d	NodelD
	<i>node's index</i>
DimensionLabelOrientation	LayerID
	<i>layer's index</i>
DimensionStyle	Position
	<i>position</i>
	LabelOrientation
	<i>orientation</i>
	TextBoxStyle
	<i>style</i>
)

Functions

long	AddDimensionLine ([in] BSTR Text , [i/o] RDimensionLineParameters DimensionLineParameters)
	Text <i>text</i> <i>Note: the text must be started with MeasurementCharacter (#).</i> <i>Example: “# wall” → it gives e.g. “4.0 wall”</i>
	DimensionLineParameters <i>dimension line’s parameters</i>
	<i>Adds a new dimension as dimension line. If successful, returns dimension’s index, otherwise an error code (EDimensionsError).</i>
long	AddTextBox ([in] BSTR Text , [i/o] RTextBoxParameters TextBoxParameters)
	Text <i>text</i> TextBoxParameters <i>text box’s parameters</i>
	<i>Adds a new dimension as a text box. If successful, returns dimension’s index, otherwise an error code (EDimensionsError).</i>
long	GetDimensionLine ([in] long Index , [out] BSTR Text , [i/o] RDimensionLineParameters DimensionLineParameters)
	Index <i>dimension line’s index</i> Text <i>text</i> DimensionLineParameters <i>dimension line’s parameters</i>
	<i>Gets dimension line’s text and parameters. If successful, returns dimension’s index, otherwise an error code (EDimensionsError).</i>
long	GetParams ([in] long Index , [out] SAFEARRAY(byte) Params)
	Index <i>dimension’s index</i> Params <i>parameters</i>
	<i>Gets dimension’s parameters. If successful, returns dimension’s index, otherwise an error code (EDimensionsError). NOTE: The Parameters must be type casted into the appropriate dimension record type!</i>
long	SetParams ([in] long Index , [in] SAFEARRAY(byte) Params)
	Index <i>dimension line’s index</i> Params <i>parameters</i>
	<i>Sets dimension’s parameters. If successful, returns dimension’s index, otherwise an error code (EDimensionsError). NOTE: The Parameters must be type casted into the appropriate dimension record type!</i>
long	GetText ([in] long Index , [out] BSTR Text)
	Index <i>dimension’s index</i> Text <i>text</i>
	<i>Sets dimension’s text. If successful, returns dimension’s index, otherwise an error code (EDimensionsError).</i>
long	SetText ([in] long Index , [in] BSTR Text)
	Index <i>dimension’s index</i> Params <i>text</i>
	<i>Sets dimension’s text. If successful, returns dimension’s index, otherwise an error code (EDimensionsError).</i>
long	Delete ([in] long Index)
	Index <i>dimension’s index</i>
	<i>Deletes dimension. If successful, returns dimension’s index, otherwise an error code (EDimensionsError).</i>

Properties

long	Count <i>Get number of dimensions</i>
EDimensionType	DimensionType [long Index] <i>Get dimension's type by index</i>
BSTR	Index <i>dimension's index</i> MeasurementCharacter (#).

IAxisVMDomains

Domains of the model

Enumerated types

enum	ELineNonlinearity = {	
	InTensionAndCompression = 0x0,	<i>linear behaviour</i>
	InTensionOnly = 0x1,	<i>tension only</i>
	InCompressionOnly = 0x2 }	<i>compression only</i>
<i>Types of nonlinear behaviour.</i>		
enum	EMeshType = {	
	mtAdaptive = 0x0,	<i>adaptive mesh</i>
	mtUniform = 0x1 }	<i>unofrm mesh</i>
<i>Types of finite element mesh.</i>		
enum	EMeshGeometryType = {	
	mgtTriangle = 0,	<i>triangle mesh</i>
	mgtQuad = 1,	<i>quad mesh</i>
	mgtMixedQuadTriangle = 2 }	<i>mixed triang and quad mesh</i>
<i>Geometry type of finite element mesh.</i>		
enum	ESurfaceCharacteristics = {	
	schLinear = 0x0,	<i>linear behaviour</i>
	schTensionOnly = 0x1,	<i>tension only</i>
	schCompressionOnly = 0x2,	<i>compression only</i>
	schBilinear = 0x3 }	<i>bilinear behaviour</i>
<i>Nonlinear surface characteristics (not used).</i>		
enum	ESurfaceType = {	
	stHole = 0x0,	<i>hole</i>
	stMembraneStress = 0x1,	<i>membrane (plane stress)</i>
	stMembraneStrain = 0x2,	<i>membrane (plane strain)</i>
	stPlate = 0x3,	<i>plate</i>
	stShell = 0x4 }	<i>shell</i>
<i>Types of surface elements.</i>		
enum	EDomainVariableThicknessType = {	
	dvtvNone = 0x0,	<i>constant thickness</i>
	dvtvOneDirection = 0x1,	<i>varies in one direction</i>
	dvtvTwoDirections = 0x2 }	<i>varies in two directions</i>
<i>Type of variable thickness</i>		
enum	EDomainExcentricityType = {	
	detNone = 0x0,	<i>no eccentricity</i>
	detConstant = 0x1,	<i>constant eccentricity</i>
	detOneDirection = 0x2,	<i>eccentricity in one direction</i>
	detTwoDirections = 0x3,	<i>eccentricity in two directions</i>
	detTopSurface = 0x4,	<i>eccentricity by alignment to top surfacce</i>
	detBottomSurface = 0x5 }	<i>eccentricity by alignment to bottom surfacce</i>
<i>Types of eccentricity</i>		

```

enum EXLAMTopLayerDirection = {
    xtIdLocalX = 0x0,           in local x direction od the domain
    xtIdLocalY = 0x1,           in local y direction od the domain
    Direction of the XLM panel's top layer

```

Error codes

enum EDomainsError = {	
deEmptyContour = -100001,	<i>line list is empty</i>
deMoreThanOneContourFound = -100002,	<i>more than one contour can be identified in the line list</i>
deEmptyHole = -100003,	<i>line list is empty (when defining a hole)</i>
deMoreThanOneHoleFound = -100004,	<i>more than one hole contour can be identified in the line list</i>
doeCOMError = -100005,	<i>internal COM server error</i>
deConcreteIndexOutOfBounds = -100006,	<i>Concrete's material index is out of bounds</i>
deRebarSteelGradeIndexOutOfBounds = -100007,	<i>rebar's material index index is out of bounds</i>
deThicknessMustBePositive = -100008,	<i>domain thickness must be a positive number</i>
deRebarPosMustBePositive = -100009,	<i>rebar position must be a positive number</i>
dePhiMustBePositiveOrZero = -100010,	<i>Phi must be a positive number</i>
deNuMustBePositiveOrZero = -100011,	<i>Nu must be a positive number</i>
deTauaMustBePositiveOrZero = -100012,	<i>Tau_a must be a positive number</i>
deAggregateSizeMustBePositive = -100013,	<i>aggregate size must be a positive number</i>
deStoreyIdOutOfBounds = -100014,	<i>storey index is out of bounds</i>
deReinforcementParametersNotExists = -100015,	<i>reinforcement parameters do not exists</i>
deHoleNotInDomainPlane = -100016,	<i>hole is not in the domain's plane</i>
dePropertyNotValidForThisDomainSurfaceType = -100017,	<i>StiffnessReduction property can return this error</i>
defseMustBePositive = -100018,	<i>seismic load factor fse must be a positive value</i>
deParametersRecordNotValidForUsedDesignCode = -100019	<i>used parameter record type is not valid for current design code</i>
deConcreteCoverMustBePositive = -100020,	<i>concrete cover must be a positive value</i>
deRebarDiameterMustBePositive = -100021,	<i>Rebar diameter must be a positive value</i>
deInvalidGroupId = -100022,	<i>GroupID is not valid</i>
deXLMmoduleNotAvailable = -100023,	<i>XLM module is not available</i>
deEnvironmentClassNotValidForUsedDesignCode = -100024,	<i>Environment Class is not valid for used design code</i>
deDomainIsNotMeshed = -100025,	<i>the domain is not meshed</i>
deAlphaVRdmaxIsInvalid = -100026,	<i>the angle of shear reinforcement is invalid</i>
deThetaVRdmaxIsInvalid = -100027,	<i>the angle of shear crack is invalid</i>
deShrinkageEpsMustBePositive = -100028,	<i>shrinkage strain must be positive</i>
deRCNonlinearSurfTypeIsInvalid = -100029	<i>nonlinear surface type is invalid</i>
deAlphaAngleIsInvalid = -100030,	<i>the angle of ξ reinforcement direction is invalid</i>
deBetaAngleIsInvalid = -100031,	<i>the angle between ξ and η reinf. directions is invalid</i>
de_k_torsionIsInvalid = -100032,	<i>k_torsion should be ≤ 0.1 and < 1</i>
de_k_shearIsInvalid = -100033,	<i>k_shear should be ≤ 0.1 and < 1</i>
de_k_bendingIsInvalid = -100034,	<i>k_bending should be ≤ 0.1 and < 1</i>
deCustomStiffnessMatrixUndefined = -100035,	<i>custom stiffness matrix is not defined</i>
deCustomStiffnessMatrixNonSymmetric = -100036,	<i>custom stiffness matrix is not symmetric</i>
deCustomStiffnessMatrixNonPositiveDefine = -100037,	<i>custom stiffness matrix is not positive define</i>
deLimitingCrackWidthIsInvalid = -100038,	<i>the limiting value for crack width is invalid</i>
dePolyLineIsNotContinuous = -100039,	<i>the selected polyline is not continuous</i>
deLineDoesNotReachDomainEdge = -100040,	<i>the line does not reach the edge of selected domain</i>
deNoSelectedLine = -100041,	<i>no line is selected</i>
deNoSelectedLineAndDomain = -100042,	<i>no line nor domain is selected</i>
deMaterialIndex = -100043,	<i>MaterialIndex must be : $1 \leq MaterialIndex \leq AxisVMMaterials.Count$</i>
deReferenceIndexOutOfBounds = -100044,	<i>reference index must be : $0 \leq ReferenceIndex \leq AxisVMReferences.Count$</i>
deDomainInvalidType = -100045,	<i>an enumerated type hase violated the valid range for that type</i>
deElasticFoundationNegative = -100046 }	<i>Elastic foundation cannot be negative</i>
<i>Errors during domain or hole definition</i>	

Records / structures

	RDomainMeshParameters = (
double EMeshType	MeshSize	average mesh size [m]
EBoolean	MeshType	type of the finite element mesh
double EBoolean	IsFitToPointLoad	fit mesh to point loads
	FitToPointLoad	minimum load to fit the mesh to the load point [kN]
double EBoolean	IsFitToLineLoad	fit mesh to line loads
	FitToLineLoad	minimum load to fit the mesh to the load line [kN/m]
double EBoolean	IsFitToSurfaceLoad	fit mesh to surface loads
	FitToSurfaceLoad	minimum load to fit the mesh to the load polygon [kN/m ²]
EMeshGeometryType	MeshGeometryType	Geometry type of the mesh
long	QuadMeshQuality	Value between 1 to 6 (including). 6 is for the smoothest mesh, meshing can be very slow
)		
	RElasticFoundationXYZ = (
double	x, y, z	elastic foundation stiffness in x, y, z directions [kN/m/m ²]
)		
	RNonLinearityXYZ = (
ELineNonLinearity	x, y, z	nonlinear behaviour in x, y, z directions
)		
	RResistancesXYZ = (
double	x, y, z	resistance in x, y, z directions [kN/m ²]
)		
	RSurfaceAttr = (
double ESurfaceType	Thickness	domain thickness [m]
long	SurfaceType	domain type
	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 RNonLinearityXYZ RResistancesXYZ	ElasticFoundation	elastic foundation of the domain
ESurfaceCharacteristics	NonLinearity	nonlinear behaviour of the elastic foundation
	Resistance	resistance values of the elastic foundation
	Characteristics	nonlinear behaviour of the domain (not used)
)		
	RDomainVariableThickness = (
EDomainVariableThicknessType	VariableThicknessType	type of variability
	P1	coordinates of reference point 1
	P2	coordinates of reference point 2
	P3	coordinates of reference point 3 (only for dvvtTwoDirections type)
Double	t1	thickness at reference point 1
Double	t2	thickness at reference point 2
Double	t3	thickness at reference point 3 (only for dvvtTwoDirections type)
)		
	RDomainExcentricity = (
EDomainExcentricityType	ExcentricityType	type of eccentricity
	P1	coordinates of reference point 1
	P2	coordinates of reference point 2
	P3	coordinates of reference point 3 (only for detTwoDirections type)
Double	exc1	eccentricity at reference point 1
Double	exc2	eccentricity at reference point 2
Double	exc3	eccentricity at reference point 3 (only for detTwoDirections type)
long	GroupID	Index of the group with same eccentricity parameters
)		
	RRibbedDomainParameters = (
EAutoExcentricityType	AutoExcentricityType	Eccentricity of ribs
	h	height of the rib (including slab)
	w	width of the rib
	d	spacing of the ribs
Double	exc	custom eccentricity (only if AutoExcentricityType = aetCustom)
)		

```
RSurfaceStiffnessFactors = (  
    double k_torsion           Torsion stiffness factor, must be >= 0.1 and < 1  
    double k_shear            Shear stiffness factor, must be >= 0.1 and < 1  
    double k_bending          Bending stiffness factor, must be >= 0.1 and < 1  
)
```

Functions

long	Add ([in] SAFEARRAY(long) Linelds , [i/o] RSurfaceAttr SurfaceAttr)	
	Linelds	<i>line indexes defining the domain's contour. Lines must be in the same plane.</i>
	SurfaceAttr	<i>domain properties</i>
	<i>Defines a new domain. If successful, returns the new domain index, otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBounds, deEmptyContour [<i>Linelds array is empty</i>], deMoreThanOneContourFound [<i>multiple contours</i>], deThicknessMustBePositive, deMaterialIndex, deReferenceIndexOutOfBounds, deDomainInvalidType, deElasticFoundationNegative).</i>	
long	Add_vb (Visual Basic compatible function of Add)	
long	BulkAdd ([in] SAFEARRAY(long) LineldCounts , [in] SAFEARRAY(long) BulkLinelds , [in] SAFEARRAY(RSurfaceAttr) SurfaceAttrs , [out] SAFEARRAY(long)* DomainIds)	
	LineldCounts	<i>list of contour line counts for each domain. There will be created as many domains as the count of <i>LineldCounts</i>.</i>
	BulkLinelds	<i>appended list of domain contour lines. <i>LineldCounts</i> controls its segmentation. If <i>LineldCounts</i> has 3 elements, 4, 12 and 1, then the first domain will use the first 4 line ids for its contour, the second domain will use the next 12, and the third domain will use the final line id (presumably a full circle). Each lineindex must satisfy : $1 \leq \text{LineIndex} \leq \text{AxisVMLines.Count}$</i>
	SurfaceAttrs	<i>domain properties for each domain</i>
	DomainIds	<i>the indexes of the newly created domains. If there was an error, it will be empty</i>
	<i>Creates several new domains in one step. If successful, returns the number of created domains. The possible error codes are : (errDatabaseNotReady, errIndexOutOfBounds, deEmptyContour, deMoreThanOneContourFound, errInternalException, errOutOfMemory, deThicknessMustBePositive, deMaterialIndex, deReferenceIndexOutOfBounds, deDomainInvalidType, deElasticFoundationNegative).</i>	
long	BulkGetDomains ([in] SAFEARRAY(long) DomainIds , [out] SAFEARRAY(long)* LineldCounts , [out] SAFEARRAY(long)* BulkLinelds , [out] SAFEARRAY(RSurfaceAttr)* SurfaceAttrs)	
	DomainIds	<i>list of domain indexes. Each index must satisfy : $1 \leq \text{DomainId} \leq \text{AxisVMDomains.Count}$</i>
	LineldCounts	<i>list of contour line counts for each domain.</i>
	BulkLinelds	<i>appended list of domain contour lines. <i>LineldCounts</i> controls its segmentation. If <i>LineldCounts</i> has 3 elements, 4, 12 and 1, then the first domain will have the first 4 line ids for its contour, the second domain will have the next 12, and the third domain will have the final line id (presumably a full circle)</i>
	SurfaceAttrs	<i>domain properties of each domain</i>
	<i>Queries several domains in one step. If successful, returns the number of queried domains. The possible error codes are : (errDatabaseNotReady, errIndexOutOfBounds, errInternalException, errOutOfMemory).</i>	
long	BulkSetDomains ([in] SAFEARRAY(long) DomainIds , [in] SAFEARRAY(long)* LineldCounts , [in] SAFEARRAY(long)* BulkLinelds , [in] SAFEARRAY(RSurfaceAttr)* SurfaceAttrs)	
	DomainIds	<i>list of domain indexes. Each index must satisfy : $1 \leq \text{DomainId} \leq \text{AxisVMDomains.Count}$</i>
	LineldCounts	<i>list of contour line counts for each domain.</i>
	BulkLinelds	<i>appended list of domain contour lines. <i>LineldCounts</i> controls its segmentation. If <i>LineldCounts</i> has 3 elements, 4, 12 and 1, then the first domain will use the first 4 line ids for its contour, the second domain will use the next 12, and the third domain will use the final line id (presumably a full circle)</i>
	SurfaceAttrs	<i>domain properties of each domain</i>
	<i>Modifies several domains in one step. The domains must already exist, to create new domains use BulkAdd instead. If successful, returns the number of modified domains. The possible error codes</i>	

are : ([errDatabaseNotReady](#), [errIndexOutOfBoundsException](#), [errInternalException](#), [errOutOfMemory](#), [deEmptyContour](#), [deMoreThanOneContourFound](#), [deThicknessMustBePositive](#), [deMaterialIndex](#), [deReferenceIndexOutOfBoundsException](#), [deDomainInvalidType](#), [deElasticFoundationNegative](#)).

long **CutSelected**

It cuts the selected domain with the selected cutting lines/polylines.

long **CreateNewExcentricityGroup**

Create new eccentricity group which for group of domains. If successful, returns a new group index used in [RDomainExcentricity](#), otherwise returns an error code (see [EGeneralErrors](#) or [EDomainsError](#)).

long **DeleteReinforcementParametersFromSelectedDomains**

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

long **DeleteSelected**

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

long **DeleteMeshes**([in] SAFEARRAY(long) **DomainIds**)

DomainIds array of domain indices

Deletes meshes from domains with index given in DomainIds array. If successful, returns number of domains. Otherwise returns an error code ([errDatabaseNotReady](#)).

Note: The mesh is deleted only if it exists. No error message is given if some domains do not have defined mesh.

long **DeleteAllMeshes**

Deletes meshes from domains in the model. If successful, returns number of domains. Otherwise returns an error code ([errDatabaseNotReady](#)).

Note: The mesh is deleted only if it exists. No error message is given if some domains do not have defined mesh.

long **DeleteNameOfAllDomains**

Deletes previously added default name of all domains. If successful, returns number of domains. Otherwise returns an error code ([errDatabaseNotReady](#)).

long **GenerateMeshOnSelectedDomains** ([i/o] [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 selected domains, otherwise returns an error code ([errDatabaseNotReady](#) or other negative numbers [in this case see [ErrorCodes](#), [ErrorPoints](#), [ErrorLines](#) for more information]).

long	GenerateMeshOnSelectedDomainsWithOriginalParams ([out] SAFEARRAY(long)* ErrorCodes , [out] SAFEARRAY(long)* ErrorPoints , [out] SAFEARRAY(long)* ErrorLines)
	ErrorCodes array of error codes
	ErrorPoints array of node indexes with errors
	ErrorLines array of line indexes with errors
	<i>If successful, returns the number of selected domains, otherwise returns an error code (errDatabaseNotReady).</i>
long	GetCustomStiffnessMatrix ([in] long Index , [i/o] RMatrix3x3 A , [i/o] RMatrix3x3 B , [i/o] RMatrix3x3 D , [i/o] RMatrix2x2 S)
	Index domain index
	A membrane stiffness matrix, more in manual: domain with custom stiffness matrix
	B coupling stiffness matrix, more in manual: domain with custom stiffness matrix
	D plate flexural stiffness matrix, more in manual: domain with custom stiffness matrix
	S adjusted shear stiffness matrix, more in manual: domain with custom stiffness matrix
	<i>Get the domain's custom stiffness matrixes. If successful, returns the domain index, otherwise returns an error code (see EGeneralError or EDomainsError).</i>
long	GetExcentricity ([in] long Index , [i/o] RDomainExcentricity DomainExcentricity)
	Index domain index
	DomainExcentricity parameters for defining eccentricity
	<i>If successful, returns the domain index, otherwise returns an error code (see EGeneralError or EDomainsError).</i>
long	GetRibbedDomainParameters ([in] long Index , [i/o] RPoint3d Origin , [i/o] RRibbedDomainParameters x , [i/o] RRibbedDomainParameters y)
	Index domain index
	Origin origin used for generating ribs
	x parameters for defining ribs in x direction
	y parameters for defining ribs in y direction
	<i>If successful, returns the domain index, otherwise returns an error code (see EGeneralError or EDomainsError).</i>
long	GetSelectedItemIds ([out] SAFEARRAY (long) * ItemIds)
	ItemIds list of selected domains
	<i>If successful, returns the number of selected elements ,otherwise returns an error code (see EGeneralErrors or EDomainsError).</i>
long	GetSelectedWallIds ([out] SAFEARRAY(long)* DomainIds)
	DomainIds array of selected wall domain indexes.
	<i>If successful, returns the number of selected wall domains, otherwise returns an error code (errDatabaseNotReady).</i>
long	GetSelectedSlabIds ([out] SAFEARRAY(long)* DomainIds)
	DomainIds array of selected slab domain indexes.
	<i>If successful, returns the number of selected slab domains, otherwise returns an error code (errDatabaseNotReady).</i>
long	GetSelectedOtherIds ([out] SAFEARRAY(long)* DomainIds)
	DomainIds array of selected other domain indexes.
	<i>If successful, returns the number of selected other domains, otherwise returns an error code (errDatabaseNotReady).</i>

long	GetVariableThickness ([in] long Index , [i/o] RDomainVariableThickness DomainVariableThickness)
	Index domain index
	DomainVariableThickness parameters for defining variable thickness
	<i>If successful, returns the domain index, otherwise returns an error code (see EGeneralError or EDomainsError).</i>
long	GetXLAMParameters ([in] long Index , [out] long XLAMIndex , [out] EXLAMTopLayerDirection TopLayerDirection)
	Index domain index
	XLAMIndex XLAM panel index, see IAxisVMXLAMpanels
	TopLayerDirection Direction of the top layer
	<i>If successful, returns the domain index, otherwise returns an error code (see EGeneralError or EDomainsError).</i>
long	RenameSelectedDomains ([in] long NewBase , [in] BSTR FormatStr)
	NewBase start number for renaming, must be a positive number
	FormatStr prefix string of the new name + "_" eg "Domain_"
	<i>Example: NewBase=100 and FormatStr= 'new_' then names are: 'new100', 'new101',...etc.</i>
	<i>If successful, returns number of selected domains. Otherwise returns an error code (errDatabaseNotReady).</i>
long	SelectAllWalls ([in] ELongBoolean Select)
	Select selection state
	<i>If Select is True, selects all wall domains.</i>
	<i>If Select is False, deselects all wall domains.</i>
	<i>If successful, returns the number of selected wall domains, otherwise returns an error code (errDatabaseNotReady).</i>
	<i>NOTE: Call Refresh function afterwards if not called between functions BeginUpdate and EndUpdate</i>
long	SelectAllSlabs ([in] ELongBoolean Select)
	Select selection state
	<i>If Select is True, selects all slab domains.</i>
	<i>If Select is False, deselects all slab domains.</i>
	<i>If successful, returns the number of selected slab domains, otherwise returns an error code (errDatabaseNotReady).</i>
	<i>NOTE: Call Refresh function afterwards if not called between functions BeginUpdate and EndUpdate</i>
long	SelectAllOthers ([in] ELongBoolean Select)
	Select selection state
	<i>If Select is True, selects all other type of domain.</i>
	<i>If Select is False, deselects all other type of domain.</i>
	<i>If successful, returns the number of selected other type of domain, otherwise returns an error code (errDatabaseNotReady).</i>
	<i>NOTE: Call Refresh function afterwards if not called between functions BeginUpdate and EndUpdate</i>

long	SelectAllWallsAtStorey ([in] ELongBoolean Select , [in] long StoreyId)
	Select selection state
	StoreyId storey index
	<i>If Select is True, selects all wall domains at storey with index StoreyId. If Select is False, deselects all wall domains at storey with index StoreyId. If successful, returns the number of selected wall domains, otherwise returns an error code (errDatabaseNotReady).</i>
	<i>NOTE: Call Refresh function afterwards if not called between functions BeginUpdate and EndUpdate</i>
long	SelectAllSlabsAtStorey ([in] ELongBoolean Select , [in] long StoreyId)
	Select selection state
	StoreyId storey index
	<i>If Select is True, selects all slab domains at storey with index StoreyId. If Select is False, deselects all slab domains at storey with index StoreyId. If successful, returns the number of selected slab domains, otherwise returns an error code (errDatabaseNotReady).</i>
	<i>NOTE: Call Refresh function afterwards if not called between functions BeginUpdate and EndUpdate</i>
long	SelectAllOthersAtStorey ([in] ELongBoolean Select , [in] long StoreyId)
	Select selection state
	StoreyId storey index
	<i>If Select is True, selects all other domains at storey with index StoreyId. If Select is False, deselects all other domains at storey with index StoreyId. If successful, returns the number of selected other domains, otherwise returns an error code (errDatabaseNotReady).</i>
	<i>NOTE: Call Refresh function afterwards if not called between functions BeginUpdate and EndUpdate</i>
long	SetCustomStiffnessMatrix ([in] long Index , [i/o] RMatrix3x3 A , [i/o] RMatrix3x3 B , [i/o] RMatrix3x3 D , [i/o] RMatrix2x2 S)
	Index domain index
	A membrane stiffness matrix, more in manual: domain with custom stiffness matrix
	B coupling stiffness matrix, more in manual: domain with custom stiffness matrix
	D plate flexural stiffness matrix, more in manual: domain with custom stiffness matrix
	S adjusted shear stiffness matrix, more in manual: domain with custom stiffness matrix
	<i>Set the domain's custom stiffness matrixes. If successful, returns the domain index, otherwise returns an error code (see EGeneralError or EDomainsError).</i>
long	SetExcentricity ([in] long Index , [i/o] RDomainExcentricity DomainExcentricity)
	Index domain index
	DomainExcentricity parameters for defining eccentricity
	<i>Returns the domain index, otherwise returns an error code (see EGeneralError or EDomainsError).</i>
long	SetRibbedDomainParameters ([in] long Index , [i/o] RPoint3d Origin , [i/o] RRibbedDomainParameters x , [i/o] RRibbedDomainParameters y)
	Index domain index
	Origin origin used for generating ribs
	x parameters for defining ribs in x direction
	y parameters for defining ribs in y direction
	<i>If successful, returns the domain index, otherwise returns an error code (see EGeneralError or EDomainsError).</i>

long **SetVariableThickness** ([in] long **Index**,
[i/o] [RDomainVariableThickness](#) **DomainVariableThickness**)
 Index domain index
DomainVariableThickness parameters for defining variable thickness
If successful, returns the domain index, otherwise returns an error code (see [EGeneralError](#) or [EDomainsError](#)).

long **SetXLAMParameters** ([in] long **Index**, [in] long **XLAMIndex**,
[in] [EXLAMTopLayerDirection](#) **TopLayerDirection**)
 Index domain index
 XLAMIndex XLAM panel index, see [IAxisVMXLAMpanels](#)
 TopLayerDirection Direction of the top layer
If successful, returns the domain index, otherwise returns an error code (see [EGeneralError](#) or [EDomainsError](#)).

Properties

AxisVMAAttachments *	Attachments Get the attachments interface
AxisVMAAttributes *	Attributes Get the attributes interface
double	Count Get number of domains in the model
ElongBoolean	HasVariableThickness [long Index] If lbTrue then the thickness vary (read only) Index index of the domain
AxisVMDomain *	Item [long Index] Get a domain by index Index index of the domain
long	IndexOfUID [long UID] Get index of the domain UID unique index of the domain
ElongBoolean	IsExcentic [long Index] If lbTrue then has excentric parameters (read only) Index index of the domain
ElongBoolean	IsRibbed [long Index] If lbTrue then has generated ribs (read only) Index index of the domain
ElongBoolean	IsXLAM [long Index] If lbTrue then it is an XLAM panel (read only) Index index of the domain
ELongBoolean	Selected [long Index] • Get or set the selection status of a domain Index index of the domain <i>NOTE: Call Refresh function afterwards if not called between functions BeginUpdate and EndUpdate</i>
long	SelCount Get number of selected domains in the model
double	StiffnessReduction [long Index] • Get or set stiffness reduction. Default value is 1. Used only for atLinearStatic and atLinearVibration analysis types . (only valid for STAS and Eurocode [RO] standards)
long	UID [long Index] Get unique index of the domain which remains the same while exists in the model Index index of the domain

IAxisVMDomain

An AxisVM domain interface.

Note:

In this interface you can define new holes (openings), reinforcement parameters, etc. in the domain.

Enumerated types

```
enum EEnvironmentClass = {  
    ecClassX0 = 0,           Environment class X0  
    ecClassXC1 = 1,          Environment class XC1  
    ecClassXC2 = 2,          Environment class XC2  
    ecClassXC3 = 3,          Environment class XC3  
    ecClassXC4 = 4,          Environment class XC4  
    ecClassXD1 = 5,          Environment class XD1  
    ecClassXD2 = 6,          Environment class XD2  
    ecClassXD3 = 7,          Environment class XD3  
    ecClassXD4 = 8,          Environment class XD4  
    ecClassXS1 = 9,          Environment class XS1  
    ecClassXS2 = 10,         Environment class XS2  
    ecClassXS3 = 11,         Environment class XS3  
    ecClassXS4 = 12,         Environment class XS4 (not available in EC_NL)  
    ecClassXF1 = 13,         Environment class XF1 (not available in EC_NL)  
    ecClassXF2 = 14,         Environment class XF2 (not available in EC_NL)  
    ecClassXF3 = 15,         Environment class XF3 (not available in EC_NL)  
    ecClassXF4 = 16,         Environment class XF4 (not available in EC_NL)  
    ecClassXA1 = 17,         Environment class XA1 (not available in EC)  
    ecClassXA2 = 18,         Environment class XA2 (not available in EC)  
    ecClassXA3 = 19,         Environment class XA3 (not available in EC)  
    ecClassXA4 = 20,         Environment class XA4 (not available in EC)  
    ecClassXM1 = 21,         Environment class M1 (not available in EC)  
    ecClassXM2 = 22,         Environment class M2 (not available in EC)  
    ecClassXM3 = 23,         Environment class M3 (not available in EC)  
    ecClassXD2B = 24,        Environment class XD2b (SIA only)  
}  
Environmental classes for all design codes
```

```
enum EStructClass_EC = {  
    scS1= 0,                Structural class S1  
    scS2= 1,                Structural class S2  
    scS3= 2,                Structural class S3  
    scS4= 3,                Structural class S4  
    scS5= 4,                Structural class S5  
    scS6= 5,                Structural class S6  
}  
Structural classes for all design codes
```

```
enum EReinforcementDirection = {  
    rdX = 0,                reinforcement in x direction  
    rdY = 1,                reinforcement in y direction  
    rdNone = 2 }             reinforcement in unknown direction  
Direction of reinforcement
```

```
enum ESLabLoadTransfer = {  
   slt_OneWay = 0,          One way spanning slab  
   slt_TwoWay = 1 }          Two way spanning slab  
Type of slab spanning
```

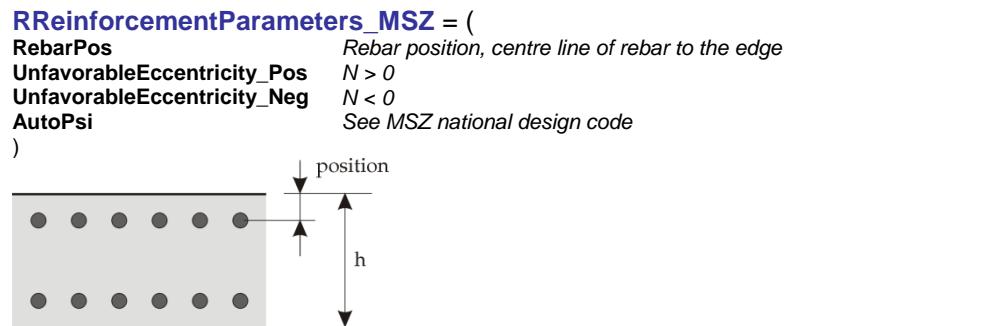
```
enum EReinforcementType = {  
    rtyp_Ortho = 0,          orthogonal x/y reinforcement  
    rtyp_Skew = 1 }           skew reinforcement  
Type of slab spanning
```

enum	ESlabLoadTransferDirection	
	= {	
	sld_OneWayX = 0	One way spanning slab in x direction
	sld_OneWayY = 1 }	One way spanning slab in y direction
	<i>Direction of one-way spanning slab</i>	
enum	ESurfaceCheck = {	
	sch_CanBeChecked = 0	See AxisVM manual reinforcement parameters
	sch_CannotBeChecked = 1	See AxisVM manual reinforcement parameters
	sch_ManipulatedAfterwards	See AxisVM manual reinforcement parameters
	= 2 }	
	<i>For RC design in accordance with NEN design code.</i>	
enum	ERCNonlinearSurfType = {	
	rcnlst_Shell = 0	Nonlinear behaviour is considered in membrane and flexural behaviour
	rcnlst_Wall = 1	Nonlinear behaviour is considered only in membrane behaviour
	rcnlst_Slab = 2 }	Nonlinear behaviour is considered only in flexural behaviour

Records / structures

	RMatrix3x3 = (
double	e11, e12, e13	
double	e21, e22, e23	
double	e31, e32, e33	
)	
	RMatrix2x2 = (
double	e11, e12	
double	e21, e22	
)	
	RPoint3d = (
double	x, y, z	<i>x, y, z coordinates of a 3D point or components of a 3D vector [m]</i>
)	
	RRebarPos = (
double	TopX	<i>Distance of top reinforcement centre line to top edge on x direction</i>
double	BottomX	<i>Distance of bottom reinforcement centre line to bottom edge on x direction</i>
	TopY	<i>Distance of top reinforcement centre line to top edge on y direction</i>
	BottomY	<i>Distance of bottom reinforcement centre line to bottom edge on y direction</i>
)	<i>Rebar position</i>
	RReinforcementParameters = (
long	ConcreteId	<i>Material index of the concrete</i>
long	RebarSteelGradeId	<i>Material index of the rebar steel</i>
Double	Thickness	<i>Thickness of the domain (surface element) used for RC design [m]</i>
)	
	RReinforcementParameters_DIN = (
Double	dxt	<i>diameter of bars in x direction at top</i>
Double	dxb	<i>diameter of bars in x direction at bottom</i>
Double	dyt	<i>diameter of bars in y direction at top</i>
Double	dyb	<i>diameter of bars in y direction at bottom</i>
	SlabLoadTransfer	<i>Type of load transfer</i>
	SlabLoadTransferDirection	<i>Direction of one-way spanning load</i>
	MainDirectionTop	<i>direction of the main reinforcement at top</i>
	MainDirectionBottom	<i>direction of the main reinforcement at bottom</i>
Double	ct	<i>cover of the main reinforcement at top (only if ApplyMinimumCover = lbFalse)</i>
Double	cb	<i>cover of the main reinforcement at bottom (only if ApplyMinimumCover = lbFalse)</i>
Double	AggregateSize	<i>maximum size of the concrete aggregate</i>
<u>ELongBoolean</u>	ApplyMinimumCover	<i>Calculate concrete covers to all reinforcements based on environment and structural class</i>
		<i>exposure class for top surface</i>
		<i>exposure class for bottom surface</i>
		<i>load factor for seismic forces (default is 1 => no change)</i>
		<i>angle of shear reinforcement for VRdmax calculation</i>

Double	ShearCrackAngle	angle of shear cracks for VRdmax calculation) For more info, see surface reinforcement in AxisVM manual design in accordance with German DIN.
RReinforcementParameters_EC = (
Double	dxt	diameter of bars in x direction at top
Double	dxb	diameter of bars in x direction at bottom
Double	dyt	diameter of bars in y direction at top
Double	dyb	diameter of bars in y direction at bottom
<u>ESlabLoadTransfer</u>	SlabLoadTransfer	Type of load transfer
<u>ESlabLoadTransferDirection</u>	SlabLoadTransferDirection	Direction of one-way spanning load
<u>EReinforcementDirection</u>	MainDirectionTop	direction of the main reinforcement at top
<u>EReinforcementDirection</u>	MainDirectionBottom	direction of the main reinforcement at bottom
Double	ct	cover of the main reinforcement at top (only if <i>ApplyMinimumCover = lbFalse</i>)
Double	cb	cover of the main reinforcement at bottom (only if <i>ApplyMinimumCover = lbFalse</i>)
Double	AggregateSize	maximum size of the concrete aggregate
<u>ELongBoolean</u>	ApplyMinimumCover	Calculate concrete covers to all reinforcements based on environment and structural class
<u>ESTructClass_EC</u>	StructClass	structural class
<u>EEnvironmentClass</u>	EnvClass_T	exposure class for top surface
<u>EEnvironmentClass</u>	ExpClass_B	exposure class for bottom surface
Double	fse	load factor for seismic forces (default is 1 => no change)
Double	UnfavorableEccentricity_Pos	$N > 0$
Double	UnfavorableEccentricity_Neg	$N < 0$
<u>ELongBoolean</u>	TakeConcTensileStrength	Take tensile strength of the concrete into account
<u>ELongBoolean</u>	ShortTerm	See reinforcement for surface elements and domains in AxisVM manual
<u>ELongBoolean</u>	TakeConcTensileStrengthNL	Take tensile strength of the concrete into account in nonlinear analysis
<u>ELongBoolean</u>	Usefctmfl	Consider flexural tensile strength in nonlinear analysis instead of tensile strength
Double	ShrinkageEps	shrinkage strain considered in nonlinear analysis
<u>ERCNonlinearSurfType</u>	RCNonlinearSurfType	nonlinear surface type
Double	ShearReinforcementAngle	angle of shear reinforcement for VRdmax calculation
Double	ShearCrackAngle	angle of shear cracks for VRdmax calculation
<u>EReinforcementType</u>	ReinforcementType	x/y orthogonal or skew reinforcement
Double	AlphaAngle	angle between local x and ξ reinf. direction [deg] (-360< α <360)
Double	BetaAngle	angle between ξ and η reinforcement directions [deg] (50≤ β ≤130)
<u>ELongBoolean</u>	CalcFromLimitingCrackWidth	calculate the reinforcement in SLS combinations based on limiting crack widths
Double	wk_b	limiting value for crack width at the level of bottom reinforcement
Double	wk2_b	limiting value for crack width at the bottom edge of concrete
Double	wk_t	limiting value for crack width at the level of top reinforcement
Double	wk2_t	limiting value for crack width at the top edge of concrete
<u>ELongBoolean</u>	ApproximateLevelArm	consideration of level arm by calculation of shear reinforcement <i>lbFalse</i> : level arm is calculated from the internal forces; <i>lbTrue</i> : 0.9*d level arm is used
<u>ELongBoolean</u>	SeelHoferMartiEquation	use the design formula of H.Seelhofer and P.Marti
) For more info, see surface reinforcement in AxisVM manual design in accordance with Eurocode.		



For more info see surface reinforcement in AxisVM manual design in accordance with Hungarian MSZ

RReinforcementParameters_NEN = (
Double	dxt	diameter of bars in x direction at top
Double	dxb	diameter of bars in x direction at bottom
Double	dyt	diameter of bars in y direction at top
Double	dyb	diameter of bars in y direction at bottom
ESurfaceCheck	SurfaceCheck_T	Surface check at top, see AxisVM manual RC design
ESurfaceCheck	SurfaceCheck_B	Surface check at bottom, see AxisVM manual RC design
EReinforcementDirection	MainDirectionTop	direction of the main reinforcement at top
EReinforcementDirection	MainDirectionBottom	direction of the main reinforcement at bottom
Double	ct	cover of the main reinforcement at top (only if <i>ApplyMinimumCover</i> = <i>lbFalse</i>)
Double	cb	cover of the main reinforcement at bottom (only if <i>ApplyMinimumCover</i> = <i>lbFalse</i>)
Double	AggregateSize	maximum size of the concrete aggregate
ELongBoolean	ApplyMinimumCover	Calculate concrete covers to all reinforcements based on environment and structural class
EStructClass EC	StructClass	structural class
EEnvironmentClass	EnvClass_T	exposure class for top surface
EEnvironmentClass	ExpClass_B	exposure class for bottom surface
ELongBoolean	ReductionOf5mm_T	Reduction of mm at top, see AxisVM manual reinforcement parameters
ELongBoolean	ReductionOf5mm_B	Reduction of mm at bottom, see AxisVM manual reinforcement parameters
ELongBoolean	CrackControl_T	Crack control at top, see AxisVM manual reinforcement parameters
ELongBoolean	CrackControl_B	Crack control at bottom, see AxisVM manual reinforcement parameters
)	

For more info, see surface reinforcement in AxisVM manual design in accordance with Eurocode.

RReinforcementParameters_ITA = (
Double	dxt	diameter of bars in x direction at top
Double	dxb	diameter of bars in x direction at bottom
Double	dyt	diameter of bars in y direction at top
Double	dyb	diameter of bars in y direction at bottom
ESlabLoadTransfer	SlabLoadTransfer	Type of load transfer
ESlabLoadTransferDirection	SlabLoadTransferDirection	Direction of one-way spanning load
EReinforcementDirection	MainDirectionTop	direction of the main reinforcement at top
EReinforcementDirection	MainDirectionBottom	direction of the main reinforcement at bottom
Double	ct	cover of the main reinforcement at top (only if <i>ApplyMinimumCover</i> = <i>lbFalse</i>)
Double	cb	cover of the main reinforcement at bottom (only if <i>ApplyMinimumCover</i> = <i>lbFalse</i>)
Double	AggregateSize	maximum size of the concrete aggregate
ELongBoolean	ApplyMinimumCover	Calculate concrete covers to all reinforcements based on environment and structural class
EStructClass EC	StructClass	structural class
EEnvironmentClass	EnvClass_T	exposure class for top surface
EEnvironmentClass	ExpClass_B	exposure class for bottom surface
Double	fse	load factor for seismic forces (default is 1 => no change)
Double	UnfavorableEccentricity_Pos	$N > 0$
Double	UnfavorableEccentricity_Neg	$N < 0$
ELongBoolean	TakeConcTensileStrength	Take tensile strength of the concrete into account
ELongBoolean	ShortTerm	See reinforcement for surface elements and domains in AxisVM manual
ELongBoolean	TakeConcTensileStrengthNL	Take tensile strength of the concrete into account in nonlinear analysis
ELongBoolean	Usefctmfl	Consider flexural tensile strength in nonlinear analysis instead of tensile strength
Double	ShrinkageEps	shrinkage strain considered in nonlinear analysis
ERCNonlinearSurfType	RCNonlinearSurfType	nonlinear surface type
Double	ShearReinforcementAngle	angle of shear reinforcement for VRdmax calculation
Double	ShearCrackAngle	angle of shear cracks for VRdmax calculation
EReinforcementType	ReinforcementType	x/y orthogonal or skew reinforcement
Double	AlphaAngle	angle between local x and ξ reinf. direction [deg] (-360< α <360)
Double	BetaAngle	angle between ξ and η reinforcement directions [deg] (50≤ β ≤130)
ELongBoolean	CalcFromLimitingCrackWidth	calculate the reinforcement in SLS combinations based on limiting crack widths
Double	wk_b	limiting value for crack width at the level of bottom reinforcement
Double	wk2_b	limiting value for crack width at the bottom edge of concrete
Double	wk_t	limiting value for crack width at the level of top reinforcement
Double	wk2_t	limiting value for crack width at the top edge of concrete
ELongBoolean	ApproximateLevelArm	consideration of level arm by calculation of shear reinforcement
ELongBoolean	SeelHoferMartiEquation	<i>lbFalse</i> : level arm is calculated from the internal forces; <i>lbTrue</i> : 0.9*d level arm is used use the design formula of H.Seelhofer and P.Marti

)
 For more info, see surface reinforcement in AxisVM manual design in accordance with Italian design code.

RReinforcementParameters_SIA = (

Double	dxt	diameter of bars in x direction at top
Double	dxb	diameter of bars in x direction at bottom
Double	dyt	diameter of bars in y direction at top
Double	dyb	diameter of bars in y direction at bottom
ESlabLoadTransfer	SlabLoadTransfer	Type of load transfer
ESlabLoadTransferDirection	SlabLoadTransferDirection	Direction of one-way spanning load
EReinforcementDirection	MainDirectionTop	direction of the main reinforcement at top
EReinforcementDirection	MainDirectionBottom	direction of the main reinforcement at bottom
Double	ct	cover of the main reinforcement at top (only if <i>ApplyMinimumCover</i> = <i>lbFalse</i>)
Double	cb	cover of the main reinforcement at bottom (only if <i>ApplyMinimumCover</i> = <i>lbFalse</i>)
Double	AggregateSize	maximum size of the concrete aggregate
ELongBoolean	ApplyMinimumCover	Calculate concrete covers to all reinforcements based on environment and structural class
EStructClass_EC	StructClass	structural class
EEvironmentClass	EnvClass_T	exposure class for top surface
EEvironmentClass	ExpClass_B	exposure class for bottom surface
Double	fse	load factor for seismic forces (default is 1 => no change)
Double	MaxCompressionHeight	See reinforcement for surface elements and domains in AxisVM manual
Double	kc_compression	See reinforcement for surface elements and domains in AxisVM manual
Double	kc_tension	See reinforcement for surface elements and domains in AxisVM manual
ELongBoolean	TakeConcTensileStrength	Take tensile strength of the concrete into account
ELongBoolean	ShortTerm	See reinforcement for surface elements and domains in AxisVM manual
ELongBoolean	TakeConcTensileStrengthNL	Take tensile strength of the concrete into account in nonlinear analysis
Double	ShrinkageEps	shrinkage strain considered in nonlinear analysis
ERCNonlinearSurfType	RCNonlinearSurfType	nonlinear surface type
Double	ShearReinforcementAngle	angle of shear reinforcement for VRdmax calculation
Double	ShearCrackAngle	angle of shear cracks for VRdmax calculation
EReinforcementType	ReinforcementType	x/y orthogonal or skew reinforcement
Double	AlphaAngle	angle between local x and ξ reinf. direction [deg] (-360< α <360)
Double	BetaAngle	angle between ξ and η reinforcement directions [deg] (50 $\leq\beta\leq$ 130)
ELongBoolean	CalcFromLimitingCrackWidth	calculate the reinforcement in SLS combinations based on limiting crack widths
Double	wk_b	limiting value for crack width at the level of bottom reinforcement
Double	wk2_b	limiting value for crack width at the bottom edge of concrete
Double	wk_t	limiting value for crack width at the level of top reinforcement
Double	wk2_t	limiting value for crack width at the top edge of concrete
ELongBoolean	ApproximateLevelArm	consideration of level arm by calculation of shear reinforcement <i>lbFalse</i> : level arm is calculated from the internal forces; <i>lbTrue</i> : 0.9* <i>d</i> level arm is used
ELongBoolean	SeelHoferMartiEquation	use the design formula of H.Seelhofer and P.Marti
)	For more info, see surface reinforcement in AxisVM manual design in accordance with Swiss design code SIA.

RReinforcementParameters_STAS = (

RRebarPos	RebarPos	Rebar position
Double	phi	See STAS national design code
Double	nu	See STAS national design code
Double	tau_a	See STAS national design code
Double	fse	load factor for seismic forces (default is 1 =>no change)
Double	mbc	See STAS national design code
Double	mbt	See STAS national design code
Double	ksi0	See STAS national design code
Double	UnfavorableEccentricity_Pos	$N > 0$
Double	UnfavorableEccentricity_Neg	$N < 0$
)	For more info see surface reinforcement in AxisVM manual design in accordance with Romanian STAS

Functions

long **AddHole** ([in] SAFEARRAY(long) **Linelds**)

Linelds Line indexes 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](#), [errIndexOutOfBoundsException](#), [deEmptyHole](#) [Linelds array is empty], [deMoreThanOneHoleFound](#) [multiple hole contours]).

long **AddHole_vb** (Visual Basic compatible function of [AddHole](#))

long **DeleteHole** ([in] long **HoleIndex**)

HoleIndex Index of the hole to delete

Deletes a hole from the domain. If successful, returns the hole index, otherwise returns an error code ([errDatabaseNotReady](#), [errIndexOutOfBoundsException](#)).

long **DeleteElasticFoundation**

If delete successful, returns domainID, otherwise returns an error code ([errDatabaseNotReady](#), [errIndexOutOfBoundsException](#)).

long **DeleteReinforcementParameters**

If successful, returns index of the domain, otherwise returns an error code([errDatabaseNotReady](#), [errIndexOutOfBoundsException](#))

long **DeleteMesh**

Deletes the mesh of domain. If successful, returns index of the domain, otherwise returns an error code([errDatabaseNotReady](#), [errIndexOutOfBoundsException](#))

long **GetElasticFoundation** ([i/o] [RStiffnessesXYZ](#) **StiffnessesXYZ**,
[i/o] [RNonLinearityXYZ](#) **NonlinearityXYZ**, [i/o] [RResistancesXYZ](#) **ResistancesXYZ**)

StiffnessesXYZ stiffnesses of the domain
NonlinearityXYZ nonlinear behaviour of the domain
ResistancesXYZ resistance of the domain

If successful, returns domainID, otherwise returns an error code ([errDatabaseNotReady](#), [errIndexOutOfBoundsException](#)).

long **GetInnerDomainIds** ([out] SAFEARRAY(long)* **DomainIds**)

DomainIds Domain indexes of inner domains

If successful, returns number of inner domains, otherwise returns an error code([errDatabaseNotReady](#), [errIndexOutOfBoundsException](#))

long **GenerateMesh** ([i/o] [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]).

long	GetCustomStiffnessMatrix ([i/o] RMatrix3x3 A, [i/o] RMatrix3x3 B, [i/o] RMatrix3x3 D, [i/o] RMatrix2x2 S)
	<p>A membrane stiffness matrix, more in manual: domain with custom stiffness matrix B coupling stiffness matrix, more in manual: domain with custom stiffness matrix D plate flexural stiffness matrix, more in manual: domain with custom stiffness matrix S adjusted shear stiffness matrix, more in manual: domain with custom stiffness matrix</p> <p>Get the domain's custom stiffness matrixes. If successful, returns the domain index. Possible error codes are errDatabaseNotReady.</p>
long	GetMeshParameters ([i/o] RDomainMeshParameters Value)
	<p>Get mesh parameters of the domain. If successful, returns the domain index. Possible error codes are errIndexOutOfBounds, errDatabaseNotReady.</p>
long	GetMeshSurfacesCoordinates ([out] SAFEARRAY(RSurfaceCoordinates) * SurfacesCoordinates)
	<p>SurfacesCoordinates Surface coordinates of all surfaces If successful, returns number of generated surface elements in domain, otherwise returns an error code(errDatabaseNotReady, errIndexOutOfBounds)</p>
long	GetNormalVector ([i/o] RPoint3D Value)
	<p>Get the normal vector of the domain. If successful, returns the domain index. Possible error codes are errIndexOutOfBounds, errDatabaseNotReady.</p>
long	GetReinforcementParameters ([i/o] RReinforcementParameters * ReinforcementParameters, [out] SAFEARRAY(byte) DesignCodeParameters)
	<p>ReinforcementParameters design code independent portion of the reinforcement parameters DesignCodeParameters a SafeArray for the design code dependent parts of the reinforcement parameters. It is an array of bytes, with a length equal to the length of the record used. The record type is determined by the active design code. Typecast to the corresponding record (see below) according to the active design code, see also How to read load data (GetLoad).</p> <p>If successful, returns index of the domain, otherwise returns an error code(errDatabaseNotReady, errIndexOutOfBounds) Record type depends on current NationalDesignCode</p> <p>RReinforcementParameters DIN: <i>ndcGerman_DIN1045_1</i></p> <p>RReinforcementParameters EC:</p> <ul style="list-style-type: none"> • <i>ndcEuroCode</i>, <i>ndcEuroCode_GER</i>, <i>ndcEuroCode_Austrian</i>, <i>ndcEuroCode_UK</i>, <i>ndcEuroCode_NL</i>, <i>ndcEuroCode_FIN</i>, <i>ndcEuroCode_HU</i>, <i>ndcEuroCode_RO</i>, <i>ndcEuroCode_CZ</i>, <i>ndcEuroCode_B</i>, <i>ndcEuroCode_PL</i>, <i>ndcEuroCode_DK</i>, <i>ndcEuroCode_S</i> <p>RReinforcementParameters ITA</p> <ul style="list-style-type: none"> • <i>ndcItalian</i> <p>RReinforcementParameters MSZ</p> <ul style="list-style-type: none"> • <i>ndcHungarian_MSZ</i> <p>RReinforcementParameters NEN</p> <ul style="list-style-type: none"> • <i>ndcDutch_NEN</i> <p>RReinforcementParameters SIA</p> <ul style="list-style-type: none"> • <i>ndcSwiss_SIA26x</i> <p>RReinforcementParameters STAS</p> <ul style="list-style-type: none"> • <i>ndcRomanian_STAS</i>

long	GetSurfaceAttr ([i/o] RSurfaceAttr Value)
Get the domain properties. If successful, returns the domain index. Possible error codes are errIndexOutOfBounds , errDatabaseNotReady .	
long	GetSurfaceStiffnessFactors ([i/o] RSurfaceStiffnessFactors Value)
Get the stiffness factors of the domain. If successful returns surface index, otherwise returns an error code (errDatabaseNotReady).	
long	GetTrMatrix ([i/o] RMatrix3x3 Value)
Get the transformation matrix of the domain. If successful, returns the domain index. Possible error codes are errIndexOutOfBounds , errDatabaseNotReady .	
long	Modify ([in] SAFEARRAY(long) Linelds , [i/o] RSurfaceAttr SurfaceAttr)
Linelds line indexes defining the domain SurfaceAttr domain properties	
Modifies domain geometry and / or domain properties. If successful, the return value > 0, otherwise returns an error code (errDatabaseNotReady , errIndexOutOfBounds , deEmptyContour [Linelds array is empty], deMoreThanOneContourFound [multiple contours]).	
long	Modify_vb (Visual Basic compatible function of Modify)
long	ModifyHole ([in] long HoleIndex , [in] SAFEARRAY(long) Linelds)
HoleIndex Index of the hole Linelds Array of line indexes around hole	
If successful, returns HoleIndex, otherwise returns an error code(errDatabaseNotReady , errIndexOutOfBounds , deEmptyHole , deMoreThanOneHoleFound)	
long	ModifyHole_vb (Visual Basic compatible function of ModifyHole)
long	SetContourLines ([in] SAFEARRAY(long) Linelds)
Set only contour lines of the domain. If successful, returns the domain index. Possible error codes are errIndexOutOfBounds , errDatabaseNotReady , deEmptyContour , deMoreThanOneContourFound .	
long	SetContourLines_vb (Visual Basic compatible function of SetContourLines)
long	SetCustomStiffnessMatrix ([i/o] RMatrix3x3 A , [i/o] RMatrix3x3 B , [i/o] RMatrix3x3 D , [i/o] RMatrix2x2 S)
A membrane stiffness matrix, more in manual: domain with custom stiffness matrix B coupling stiffness matrix, more in manual: domain with custom stiffness matrix D plate flexural stiffness matrix, more in manual: domain with custom stiffness matrix S adjusted shear stiffness matrix, more in manual: domain with custom stiffness matrix	
Set the domain's custom stiffness matrixes. If successful, returns the domain index. Possible error codes are errDatabaseNotReady .	
long	SetElasticFoundation ([i/o] RStiffnessesXYZ StiffnessesXYZ , [i/o] RNonLinearityXYZ NonlinearityXYZ , [i/o] RResistancesXYZ ResistancesXYZ)
StiffnessesXYZ stiffnesses of the domain NonlinearityXYZ nonlinear behaviour of the domain ResistancesXYZ resistance of the domain	
If successful, returns domainID, otherwise returns an error code (errDatabaseNotReady , errIndexOutOfBounds).	
long	SetMeshParameters ([i/o] RDomainMeshParameters Value)
Set mesh parameters of the domain. If successful, returns the domain index. Possible error codes are errIndexOutOfBounds , errDatabaseNotReady .	

long	SetSurfaceAttr ([i/o] RSurfaceAttr Value) Set the domain properties. If successful, returns the domain index. Possible error codes are errIndexOutOfBounds , errDatabaseNotReady .
long	SetSurfaceStiffnessFactors ([i/o] RSurfaceStiffnessFactors Value) Set the stiffness factors of the domain. If successful returns surface index, otherwise returns an error code (errDatabaseNotReady).
long	SetReinforcementParameters ([i/o] RReinforcementParameters * ReinforcementParameters , [in] SAFEARRAY (byte) DesignCodeParameters) ReinforcementParameters design code independent portion of the reinforcement parameters DesignCodeParameters a SafeArray containing the design code dependent parts of the reinforcement parameters. It is an array of bytes, with a length equal to the length of the record used. The record type is determined by the active design code. Use the corresponding record (see below) according to the active design code, see also How to read load data (GetLoad) . If successful, returns index of the domain, otherwise returns an error code(errDatabaseNotReady , errIndexOutOfBounds , deConcreteIdIndexOutOfBounds , deRebarSteelGradeIdIndexOutOfBounds , deThicknessMustBePositive , deRebarPosMustBePositive , dePhiMustBePositiveOrZero , deNuMustBePositiveOrZero , deTauaMustBePositiveOrZero , deAggregateSizeMustBePositive)
	Record type depends on current NationalDesignCode RReinforcementParameters DIN: <i>ndcGerman_DIN1045_1</i>
	RReinforcementParameters EC: <ul style="list-style-type: none"> • <i>ndcEuroCode</i>, <i>ndcEuroCode_GER</i>, <i>ndcEuroCode_Austrian</i>, <i>ndcEuroCode_UK</i>, <i>ndcEuroCode_NL</i>, <i>ndcEuroCode_FIN</i>, <i>ndcEuroCode_HU</i>, <i>ndcEuroCode_RO</i>, <i>ndcEuroCode_CZ</i>, <i>ndcEuroCode_B</i>, <i>ndcEuroCode_PL</i>, <i>ndcEuroCode_DK</i>, <i>ndcEuroCode_S</i> RReinforcementParameters ITA <ul style="list-style-type: none"> • <i>ndcItalian</i> RReinforcementParameters MSZ <ul style="list-style-type: none"> • <i>ndcHungarian_MSZ</i> RReinforcementParameters NEN <ul style="list-style-type: none"> • <i>ndcDutch_NEN</i> RReinforcementParameters SIA <ul style="list-style-type: none"> • <i>ndcSwiss_SIA26x</i> RReinforcementParameters STAS <i>ndcRomanian_STAS</i>
long	SetReinforcementParameters_vb ([i/o] RReinforcementParameters * ReinforcementParameters , [i/o] SAFEARRAY (byte) DesignCodeParameters) Visual Basic compatible function of SetReinforcementParameters

Properties

EArchitectElemType	ArchitectElemType • Get or set architect element type
double	Area Get domain area [m^2]
unsigned long	ContourColour • Get or set contour colour. If value is 0xFFFFFFFF then same as assigned material colour
long	ContourColour_vb • Visual Basic compatible property of ContourColour
SAFEARRAY (long)*	ContourLines Get line indexes of the contour
IAxisVMLines3d *	ContourPolygon Get contourpolygon

<u>ELongBoolean</u>	ElasticFoundationExists True if finds elastic foundation. (Read only property)
long	HoleCount Get number of holes in the domain
SAFEARRAY(long)*	HoleLines [long HoleIndex] Get line indexes of a hole contour by index
double	HoleArea [long HoleIndex] Get hole area by hole index [m^2]
unsigned long	MaterialColour • Get or set material colour. If value is 0xFFFFFFFF then same as assigned material colour.
long	MaterialColour_vb • Visual Basic compatible property of MaterialColour
<u>ELongBoolean</u>	MeshExists True if the domain is meshed (Read only property)
SAFEARRAY(long)*	MeshSurfaceIds Get array of mesh surface element indexes according to <u>IAxisVMSurfaces</u>
BSTR	Name Get name of the domain (read only property)
<u>ELongBoolean</u>	IsWall True if domain is a wall (read only property)
<u>ELongBoolean</u>	IsSlab True if domain is a slab (read only property)
<u>ELongBoolean</u>	IsOtherType True if domain is an other type (read only property)
long	OuterDomainId Get index of the outer domain, if successful l. Otherwise returns an error code(<u>errDatabaseNotReady</u> , <u>errIndexOutOfBounds</u>)
<u>ELongBoolean</u>	ReinforcementParametersExists True if finds reinforcement parameters. (read only property)
long	SeismicStoreyId Get seismic storey index of the domain, if successful l. Otherwise returns an error code(<u>errDatabaseNotReady</u> , <u>errIndexOutOfBounds</u>)
double	StiffnessReduction • Get or set stiffness reduction. Default value is 1. Used only for atLinearStatic and atLinearVibration analysis <u>types</u> . (only valid for STAS and Eurocode [RO] standards)
long	StoreyId Get storey index of the domain, if successful l. Otherwise returns an error code(<u>errDatabaseNotReady</u> , <u>errIndexOutOfBounds</u>)
<u>ELongBoolean</u>	VariableThickness True if thickness varies. (read only property)
double	Volume Get total volume of the domain [m^3]
double	Weight Get total mass of the domain [kg]
long	UID Get unique index of the domain which remains the same while exists in the model

IAxisVMDomainSupports

Domain supports of the model.

Note:

In this interface you can access supports of the domains.

Functions

long **AddDomainElasticFoundation** ([in] long **DomainId**,
[i/o] **RStiffnessesXYZ** **StiffnessesXYZ**, [i/o] **RNonLinearityXYZ** **NonlinearityXYZ**,
[i/o] **RResistancesXYZ** **ResistancesXYZ**)

DomainId index of the domain support
($0 < \text{DomainId} \leq \text{IAxisVMDomainSupports.Count}$)

StiffnessesXYZ stiffnesses of the domain support

NonlinearityXYZ nonlinear behaviour of the domain support

ResistancesXYZ resistance of the domain support

Adds a domain support. If successful, returns number of domain supports, otherwise returns an error code ([errDatabaseNotReady](#), [errIndexOutOfBoundsException](#)).

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

Index index of the domain support $1 \leq \text{Index} \leq \text{Count}$

If successful returns domain support index, otherwise returns an error code ([errDatabaseNotReady](#), [errIndexOutOfBoundsException](#)).

long **DeleteSelected**

Returns number of deleted domain supports

long **GetSelectedItemIds** ([out] SAFEARRAY (long) * **ItemIds**)

ItemIds list of selected domain supports

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

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

Select selection state

If Select is True, selects all domain supports.

If Select is False, deselects all domain supports.

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

NOTE: Call [Refresh](#) function afterwards if not called between functions [BeginUpdate](#) and [EndUpdate](#)

Properties

[IAxisVMDomainSupport](#) **Item** [long **index**] Get domain support interface

Index index of the domain support, $1 \leq \text{Index} \leq \text{Count}$

long **Count** Get number of domain supports in the model

long **DomainId** [long **Index**] Get domain index

Index index of the domain support, $1 \leq \text{Index} \leq \text{Count}$

[ELongBoolean](#) **Selected** [long **Index**] • Get or set the selection status of a surface support

NOTE: Call [Refresh](#) function afterwards if not called between functions [BeginUpdate](#) and [EndUpdate](#)

Index index of the domain support, $1 \leq \text{Index} \leq \text{Count}$

long **SelCount** Get number of selected surface supports in the model.

Negative number is an error code ([errDatabaseNotReady](#)).

IAxisVMDomainSupport

AxisVM Domain support interface.

Enumerated types

```
enum EDomainSupportType = {  
    dstDomainElasticFoundation  Elastic type of foundation  
    = 0x0}
```

Functions

long	GetNonLinearityXYZ ([i/o] RNonLinearityXYZ* NonLinearityXYZ)
<p>NonLinearityXYZ nonlinearity of the domain support <i>If successful, returns the index of the domain support otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBounds).</i></p>	
<hr/>	
long	GetStiffnessesXYZ ([i/o] RStiffnessesXYZ* StiffnessesXYZ)
<p>StiffnessesXYZ stiffnesses of the elastic foundation <i>If successful, returns the index of the domain support otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBounds).</i></p>	
<hr/>	
long	GetResistancesXYZ ([i/o] RResistancesXYZ* ResistancesXYZ)
<p>ResistancesXYZ resistance of the domain support <i>If successful, returns the index of the domain support otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBounds).</i></p>	
<hr/>	
long	SetStiffnessesXYZ ([i/o] RStiffnessesXYZ* StiffnessesXYZ)
<p>StiffnessesXYZ stiffnesses of the domain support <i>If successful, returns the index of the domain support otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBounds).</i></p>	
<hr/>	
long	SetNonLinearityXYZ ([i/o] RNonLinearityXYZ* NonLinearityXYZ)
<p>NonLinearityXYZ nonlinearity of the domain support <i>If successful, returns the index of the domain support otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBounds).</i></p>	
<hr/>	
long	SetResistancesXYZ ([i/o] RResistancesXYZ* ResistancesXYZ)
<p>ResistancesXYZ resistance of the domain support <i>If successful, returns the index of the domain support otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBounds).</i></p>	
<hr/>	

Properties

long	DomainId Get domain support index
EDomainSupportType	SupportType Get type of domain support

IAxisVMDrawingsLibrary

Drawings library of the model.

Functions

long **AddWindow** ([in] long **WindowIndex**, [in] BSTR **Name**)

WindowIndex index of the displayed window, $1 \leq Index \leq IAxisVMWindows.Count$

Name name of the new drawing

Add displayed window to library. If successful, returns index of the drawing, otherwise an error code (see [EGeneralError](#)).

long **AddWindows** ([in] BSTR **Name**)

Name name of the new drawing

Add all displayed windows to library. If successful, returns index of the drawing, otherwise an error code (see [EGeneralError](#)).

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

Index index of the drawing, $1 \leq Index \leq Count$

If successful returns drawing index, otherwise returns an error code (see [EGeneralError](#)).

long **DisplayDrawing** ([in] long **Index**, [in] long **WindowIndex**,

[in] [ElongBoolean](#) **RestoreResultComponent**, [in] [ElongBoolean](#) **Units**)

Index index of the drawing, $1 \leq Index \leq Count$

WindowIndex index of the displayed window, $0 \leq Index \leq IAxisVMWindows.Count$

If WindowIndex = 0 than all windows will be displayed

RestoreResultComponent If lbTrue result components will be restored

Units If lbTrue units will be displayed also

Display drawing from library. If successful, returns drawing index, otherwise returns an error code (see [EGeneralError](#)).

Properties

long **Count** Get number of drawings.

BSTR **Name** [long **Index**] • Get or set the name of the drawing with index **Index**.

IAxisVMDynamicLoadFunctions



Dynamic load functions of the model. If property returning this interface is null (nil) then the extension module DYN is not available.

Error codes

enum	EFunctionsError = {
	fuePointIndexOutOfBounds = -100001
	fueFailedToModifyFunction = -100002
	fueFileExists = -100003
	fueFailedToAddFromFile = -100004
	fueNameAlreadyExists = -100005
	fuelInvalidFunction = -100006 }

*point of function out of bounds
function modification failed
error during saving function, file already exists
error during loading function, file open failed
function with the same name already exists in the model
function points not valid, see AxisVM manual for more info*

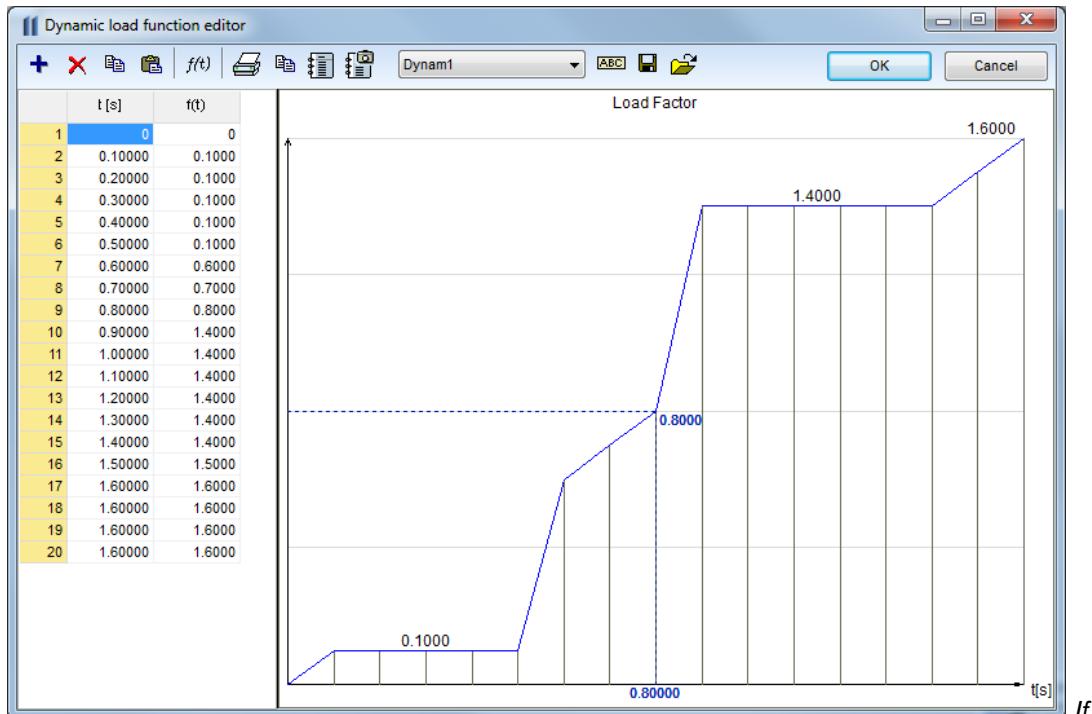
Functions

long **Add ([in] BSTR Name, [in] SAFEARRAY(RPoint2D)* FunctionPoints)**

Name *name of the new dynamic load function*

FunctionPoints *Function points of the dynamic load function, where Coord1 is time in seconds [s] and Coord2 is the load factor [-]*

Adds a new dynamic load function, see example:



If successful, returns index of the new dynamic load function, otherwise an error code ([fueNameAlreadyExists](#), [fuelInvalidFunction](#), [errDatabaseNotReady](#), [errCOMServerInternalError](#)).

long **Add_vb** (Visual Basic compatible function of [Add](#))

long **AddFromFile ([in] BSTR Name, [in] BSTR FileName)**

Name *Name of the dynamic load function*

FileName *Name of the file containing dynamic load function*

If successful, returns index of the new dynamic load function, otherwise an error code ([errDatabaseNotReady](#), [fueFailedToAddFromFile](#)).

long	AddPoint ([in] long index , [i/o] RPoint2d* FunctionPoint)
	index dynamic load function index
	FunctionPoint coordinates of the added point in the dynamic load function
	<i>If successful, returns number of points after adding point to the function, otherwise an error code (errDatabaseNotReady or errIndexOutOfBounds).</i>
long	Clear
	<i>Deletes all dynamic load functions.</i>
	<i>If successful, returns 1, otherwise an error code (errDatabaseNotReady).</i>
long	Delete ([in] long index)
	index dynamic load function index
	<i>If successful, returns number of dynamic load functions after delete, otherwise an error code (errDatabaseNotReady or errIndexOutOfBounds).</i>
long	DeletePoint ([in] long index , [in] long PointIndex)
	index dynamic load function index
	PointIndex index of a point in dynamic load function
	<i>If successful, returns number of points after deleting a point from function, otherwise an error code (fuePointIndexOutOfBounds, errDatabaseNotReady or errIndexOutOfBounds).</i>
long	DeletePoints ([in] long index , [in] long StartPointIndex , [in] long EndPointIndex)
	index dynamic load function index
	StartPointIndex index of the start point in dynamic load function
	EndPointIndex index of the end point in dynamic load function
	<i>Deletes function point range starting with point index StartPointIndex (including) and ending with EndPointIndex (including).</i>
	<i>If successful, returns number of points after deleting points from function, otherwise an error code (fuePointIndexOutOfBounds, errDatabaseNotReady or errIndexOutOfBounds).</i>
long	GetPoints ([in] long index , [out] SAFEARRAY(RPoint2d*) FunctionPoints)
	index dynamic load function index
	FunctionPoints point array of the dynamic load function
	<i>If successful, returns number of points of the function, otherwise an error code (errDatabaseNotReady, errIndexOutOfBounds).</i>
long	IndexOf ([in] BSTR Name)
	Name Name of the dynamic load function
	<i>If successful, returns dynamic load function index, otherwise an error code (errDatabaseNotReady).</i>
long	Modify ([in] long index , [in] SAFEARRAY(RPoint2d*) FunctionPoints)
	index dynamic load function index
	FunctionPoints point array of the dynamic load function, see function Add for more info about point coordinates.
	<i>If successful, returns number of points of the function, otherwise an error code (errDatabaseNotReady, errIndexOutOfBounds, fueFailedToModifyFunction or errCOMServerInternalError).</i>
long	Modify_vb (Visual Basic compatible function of Modify)

long **SaveToFile** ([in] long **index**, [in] BSTR **FileName**)
 index *dynamic load function index*
 FileName *Name of the file used for saving*
Saves function to AxisVM directory \dfn with file extension: dfn
If successful, returns dynamic load function index, otherwise an error code
([errDatabaseNotReady](#), [errIndexOutOfBounds](#) or [fueFileExists](#)).

Properties

long **Count** *Get number of dynamic load functions.*

BSTR **Name [long Index]** *Get or set name of the dynamic load function with index Index.*

long **PointCount [long Index]** *Get number of points of the dynamic load function with index Index.*

IAxisVMEdgeConnections

Edge connections of the model.

Error codes

enum	EEdgeConnectionError: = {	
	eceLineIndexOutOfBounds = -100001	<i>EEdgeConnectionRec.LineId in IAxisVMEdgeConnections.Add is invalid</i>
	eceDomainIndexOutOfBounds = -100002	<i>EEdgeConnectionRec.DomainId in IAxisVMEdgeConnections.Add is invalid</i>
	eceErrorAdding = -100003}	<i>IAxisVMEdgeConnections.Add was not successful</i>

Records / structures

	REdgeConnectionRec = (
long	LineID	<i>Id of line (edge)</i>
long	DomainID	<i>Id of domainID</i>
<u>RStiffnesses</u>	Stiffnesses	<i>Stiffnesses</i>
<u>RResistances</u>	Resistances	<i>Resistances</i>
)	
		<i>IAxisVMEdgeConnections.Add , .GetRec and .SetRec functions are using this record</i>

Functions

long	Add ([i/o] REdgeConnectionRec * EdgeConnectionRec)	
	EdgeConnectionRec <i>Edge connection parameters</i>	
	<i>If successful returns EdgeConnectionID, otherwise an error code (errDatabaseNotReady, eceLineIndexOutOfBounds, eceDomainIndexOutOfBounds, eceErrorAdding)</i>	
long	Clear	
	<i>Returns number of deleted edge connections, otherwise an error code (errDatabaseNotReady).</i>	
long	Delete ([in] long Index)	
	Index <i>index of the edge connection, 1 ≤ Index ≤ Count</i>	
	<i>If successful returns EdgeConnectionID, otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBoundsException)</i>	
long	DeleteSelected	
	<i>Returns number of deleted edge connections, otherwise an error code (errDatabaseNotReady).</i>	
long	GetRec ([in] long Index , [i/o] REdgeConnectionRec * EdgeConnectionRec)	
	Index <i>Edge connection index, 1 ≤ Index ≤ Count</i>	
	EdgeConnectionRec <i>Edge connection parameters</i>	
	<i>If successful returns EdgeConnectionID, otherwise an error code (errDatabaseNotReady, errIndexOutOfBoundsException)</i>	
long	GetSelectedItemIds ([out] SAFEARRAY (long) * ItemIds)	
	ItemIds <i>Index list of selected edge connections</i>	
	<i>Returns the number of selected edge connections</i>	
long	SelectAll ([in] ELongBoolean Select)	
	Select <i>selection state</i>	
	<i>If Select is True, selects all edge connections.</i>	
	<i>If Select is False, deselects all edge connections.</i>	
	<i>If successful, returns the number of selected edge connections, otherwise returns an error code (errDatabaseNotReady)</i>	
long	SetRec ([in] long Index , [i/o] REdgeConnectionRec * EdgeConnectionRec)	
	Index <i>Edge connection index, 1 ≤ Index ≤ Count</i>	
	EdgeConnectionRec <i>Edge connection parameters</i>	
	<i>If successful returns EdgeConnectionID, otherwise an error code (errDatabaseNotReady, errIndexOutOfBoundsException)</i>	

Properties

long **Count** *Get number of edge connections in the model*

[ELongBoolean](#) **Selected** [long **Index**] • *Get or set the selection status of the edge connection*
Index *index of the edge connection, $1 \leq Index \leq Count$*

NOTE: Call [Refresh](#) function afterwards if not called between functions [BeginUpdate](#) and [EndUpdate](#)

long **SelCount** *Get number of selected edge connections in the model*

IAxisVMEvelopes

Envelope of results. Envelopes are generated automatically for various types of analysis.

Envelope index and EnvelopeUID

Default envelopes are automatically created. Several envelopes with Name="Default" can be generated automatically for each analysis type. They will contain all load combinations if exist in model alternatively all load cases. Use property AnalysisType to filter them.

If EnvelopeUID is 0, then envelope is calculated as follow:

- Model has only load cases: Envelope is calculated from load cases .
- Model has load combinations: Envelope is calculated from load combinations only. Depends on type of results which load combinations are taken into account in the envelope. ULS load combinations are considered for forces, stresses, design values, reinforcement values and shear capacity values. SLS combinations are considered for displacements and crack widths.
- Envelope with EnvelopeUID = 0 is not in this interface. It is used only in other interfaces to get same envelope results as in older versions of AxisVM.

Error codes

enum	EEnvelopesError: = {	
	eeLoadCaseIdOutOfBounds = -100001	<i>LoadCaseId is invalid</i>
	eeLoadCombinationIdOutOfBounds = -100002	<i>LoadCombinationId is invalid</i>
	eeErrorAddingEnvelope = -100003	<i>Can't add new envelope</i>
	eeErrorModifyingEnvelope = -100004	<i>Can't edit or modify the envelope</i>
	eeNotUserDefinedEnvelope = -100005 }	<i>Not an user defined envelope</i>

Enumerated types

enum	EEnvelopeGroup = {	
	egDefault = 0x00 ,	<i>default envelope (used in all version prior to version 12 of AxisVM)</i>
	egLoadCases = 0x01 ,	<i>envelope of all load cases</i>
	egLoadCombinations = 0x02 ,	<i>envelope of all load combinations</i>
	egULS_ALL = 0x03 ,	<i>envelope of all ULS load combinations</i>
	egULS = 0x04 ,	<i>envelope of fundamental ULS combinations (See 6.4.3.2 EN 1990)</i>
	egULSSeismic = 0x05 ,	<i>envelope of seismic ULS combinations (See 6.4.3.4 EN 1990)</i>
	egULSExceptional = 0x06 ,	<i>envelope of exceptional ULS combinations (See 6.4.3.3 EN 1990)</i>
	egULSab = 0x07 ,	<i>envelope of worst of a or b combinations type considered , see EN 1990:6.10(a,b)</i>
	egSLS_ALL = 0x08 ,	<i>envelope of all SLS load combinations</i>
	egSLSCharacteristic = 0x09 ,	<i>envelope of characteristic SLS combinations (See 6.5.3 EN 1990)</i>
	egSLSFrequent = 0x0A ,	<i>envelope off frequent SLS combinations (See 6.5.3 EN 1990)</i>
	egSLSQuasipermanent = 0x0B ,	<i>envelope of quasi-permanent SLS combinations (See 6.5.3 EN 1990)</i>
	egCustomCombinations = 0x0C ,	<i>envelope of custom load combinations</i>
	egGeo = 0x0D ,	<i>envelope of geotechnic load combinations</i>
	egGeoULS_A1 = 0x0E ,	<i>envelope of ULS and ULS type A1 combinations</i>
	egGeoULS_A2 = 0x10 ,	<i>envelope of ULS and ULS type A2 combinations</i>
	egGeoULSab_A1 = 0x11 ,	<i>envelope of ULSab and ULS type A1 combinations</i>
	egGeoULSab_A2 = 0x12 }	<i>envelope of ULSab and ULS type A2 combinations</i>
	<i>Type of envelope group</i>	

Functions

long **Add** ([in] BSTR **Name**, [in] EAnalysisType **AnalysisType**,
[in] SAFEARRAY(long)* **LoadCaseIDs**, [in] SAFEARRAY(long)* **LoadCombinationIDs**)

Name name of the envelope

AnalysisType Type of [Analysis](#)

LoadCaseIDs indexes of load cases considered in the envelope

LoadCombinationIDs indexes of load combinations considered in the envelope

Add new user defined envelope based on parameters.

If successful, returns the envelope index, otherwise returns an error code ([errDatabaseNotReady](#)).

long **ClearUserEnvelopes**

Deletes all user defined envelopes.

If successful, returns 1, otherwise returns an error code ([errDatabaseNotReady](#)).

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

index envelope index

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

long **GetEnvelope** ([in] long **index**, [out] SAFEARRAY(long)* **LoadCaseIDs**,

[out] SAFEARRAY(long)* **LoadCombinationIDs**)

index envelope index

LoadCaseIDs indexes of load cases considered in the envelope

LoadCombinationIDs indexes of load combinations considered in the envelope

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

long **IndexOf** ([in] EAnalysisType **AnalysisType**, [in] SAFEARRAY(ElongBoolean)* **Envelope**)

AnalysisType Type of [Analysis](#)

Envelope The length of the array has to be equal to

- AnalysisType = atLinearStatic: [IAxisVMLoadcases](#).Count + [IAxisVMLoadCombinations](#).Count;

AnalysisType = atNonLinearStatic/atDynamic: the sum of LoadLevelCounts of [IAxisVMResults](#) for every result cases.

If Envelop[i] = lbTrue, it is indicating that this element is involved in the Envelope.

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

long **IndexOfName** ([in] BSTR **Name**)

Name name of the envelope

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

long **IndexOfStandard** ([in] EAnalysisType **AnalysisType**, [in] EEnvelopeGroup **EnvelopeGroup**)

AnalysisType Type of [Analysis](#)

EnvelopeGroup Type of envelope group, correspond to load combination type

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

long **IndexOfDefault** ([in] EAnalysisType **AnalysisType**)

AnalysisType Type of [Analysis](#)

If successful, returns index of the default envelope (used in AxisVM versions prior to version 12), otherwise returns an error code ([errDatabaseNotReady](#)).

long **IndexOfUID** ([in] long **EnvelopeUID**)

EnvelopeUID Unique index of the envelope, which doesn't change in the model

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

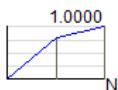
long **SetEnvelope** ([in] long **index**, [in] SAFEARRAY(long)* **LoadCaseIDs**,
[in] SAFEARRAY(long)* **LoadCombinationIDs**)
 index envelope index
 LoadCaseIDs indexes of load cases considered in the envelope
 LoadCombinationIDs indexes of load combinations considered in the envelope
If successful, returns index, otherwise returns an error code ([errDatabaseNotReady](#) or [errIndexOutOfBounds](#)).

long **Update**
Update envelopes, call after adding load group, etc. If successful, returns number of all envelopes in the model for all analysis types, otherwise returns an error code ([errDatabaseNotReady](#)).

Properties

EAnalysisType	AnalysisType [long Index] Get analysis type of the envelope with index
long	Count Get number of all envelopes in the model for all analysis types.
long	EnvelopeUID [long Index] Unique index of the envelope with index, which will remain static.
EEnvelopeGroup	Group [long Index] Get type of envelope group
BSTR	Name [long Index] Name of the envelope with index. Names can duplicate but AnalysisType will vary.
long	UserEnvelopeCount Get number of user defined envelopes in the model

IAxisVMIncrementFunctions



Increment functions of the model. Used for defining load increment functions in non-linear analysis

Functions

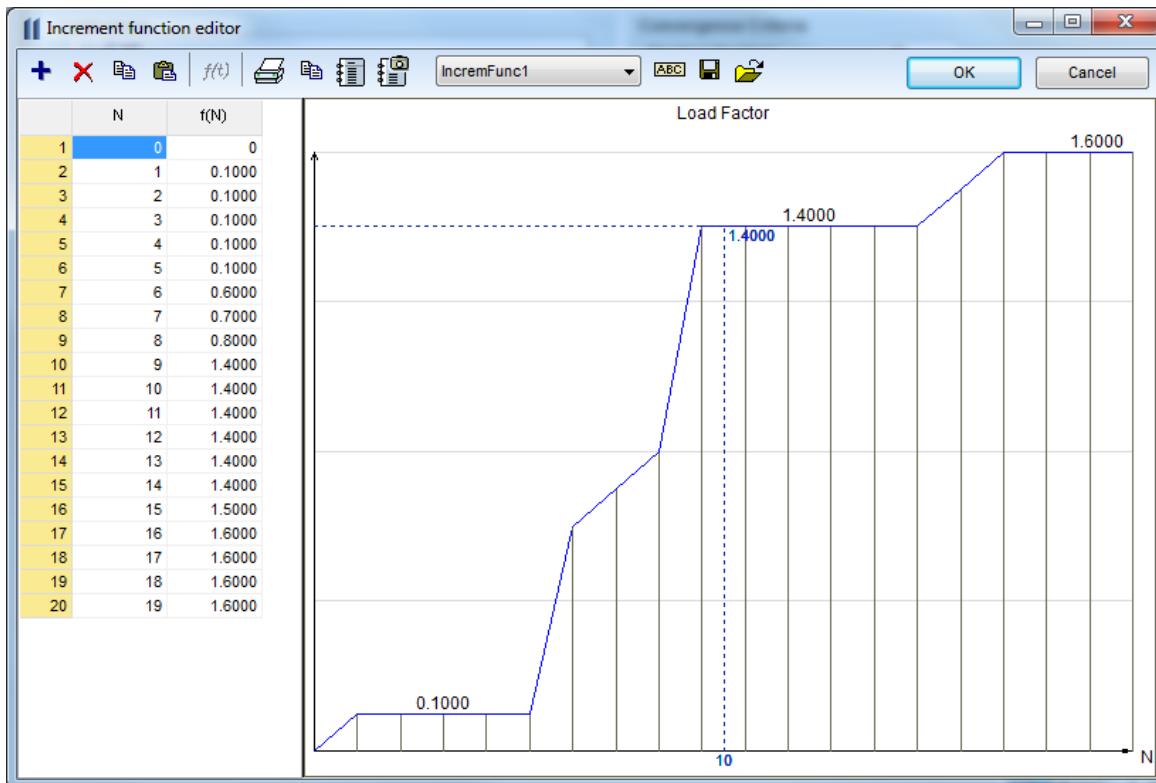
long **Add** ([in] BSTR Name, [in] SAFEARRAY(RPoint2D) FunctionPoints)

Name name of the new increment function

FunctionPoints Function points of the increment function, where Coord 1 is the function's step N and Coord2 is the load factor.

Coord1 must integer number. First point must be [0,0].

Adds a new increment function, see example:



If successful, returns index of the new function, otherwise an error code ([fueNameAlreadyExists](#), [fueInvalidFunction](#), [errDatabaseNotReady](#), [errCOMServerInternalError](#)).

long **Add_vb** (Visual Basic compatible function of **Add**)

long **AddFromFile** ([in] BSTR Name, [in] BSTR FileName)

Name Name of the increment function

FileName Name of the file containing increment function

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

long **AddPoint** ([in] long index, [i/o] RPoint2d* FunctionPoint)

index increment function index

FunctionPoint coordinates of the added point in the increment function

If successful, returns number of points after adding point to the function, otherwise an error code ([errDatabaseNotReady](#) or [errIndexOutOfBounds](#)).

long **Clear**

Deletes all increment functions. If successful, returns 1, otherwise an error code ([errDatabaseNotReady](#)).

long	Delete ([in] long index)	index time increment function index If successful, returns number of increment functions after delete, otherwise an error code (errDatabaseNotReady or errIndexOutOfBounds).
long	DeletePoint ([in] long index , [in] long PointIndex)	index increment function index PointIndex index of a point in increment function If successful, returns number of points after delete a point from function, otherwise an error code (fuePointIndexOutOfBounds , errDatabaseNotReady or errIndexOutOfBounds).
long	DeletePoints ([in] long index , [in] long StartPointIndex , [in] long EndPointIndex)	index increment function index StartPointIndex index of the start point in increment function EndPointIndex index of the end point in increment function Deletes function point range starting with point index StartPointIndex (including) and ending with EndPointIndex (including). If successful, returns number of points after delete points from function, otherwise an error code (fuePointIndexOutOfBounds , errDatabaseNotReady or errIndexOutOfBounds).
long	GetPoints ([in] long index , [out] SAFEARRAY(RPoint2d)* FunctionPoints)	index increment function index FunctionPoints point array of the increment function If successful, returns number of points of the function, otherwise an error code (errDatabaseNotReady , errIndexOutOfBounds).
long	IndexOf ([in] BSTR Name)	Name Name of the increment function If successful, returns function index, otherwise an error code (errDatabaseNotReady).
long	Modify ([in] long index , [in] SAFEARRAY(RPoint2d)* FunctionPoints)	index increment function index FunctionPoints point array of the increment function, see function Add for more info about point coordinates. If successful, returns number of points of the function, otherwise an error code (errDatabaseNotReady , errIndexOutOfBounds , fueFailedToModifyFunction or errCOMServerInternalError).
long	Modify_vb (Visual Basic compatible function of Modify)	
long	SaveToFile ([in] long index , [in] BSTR FileName)	index increment function index FileName Name of the file used for saving Saves function to AxisVM directory \inc with file extension: inc. If successful, returns function index, otherwise an error code (errDatabaseNotReady , errIndexOutOfBounds or fueFileExists).

Properties

long	Count	Get number of increment functions.
BSTR	Name [long Index]	• Get or set name of the increment function with index Index .
long	PointCount [long Index]	Get number of points of the increment function with index Index .

IAxisVMLayers

Layers of the model.

Note:

Only AxisVM layers can be created and modified.

Enumerated types

enum	ELayerType = {	
	ItAxisVM = 0x0,	<i>AxisVM layer</i>
	ItDXF = 0x1 ,	<i>DXF layer, created with DXF import</i>
	ItPDF = 0x2 }	<i>PDF layer, created with PDF import</i>
<i>Layer types</i>		
enum	ELayerPenStyle = {	
	IpsSolid = 0x0,	<i>solid line</i>
	IpsDash = 0x1 ,	<i>solid line</i>
	IpsDot = 0x2 ,	<i>doted line</i>
	IpsDashDot = 0x3 ,	<i>dash dot line</i>
	IpsDashDotDot = 0x4 }	<i>dash dot dot line</i>
<i>Pen style</i>		
enum	ELayerShapeType = {	
	Ist3DLine = 0x0,	<i>3D line</i>
	Ist3DPolygon = 0x1 ,	<i>3D polygon</i>
	Ist3DPolygonFilled = 0x2 }	<i>closed 3D polygon</i>
<i>Type Shape</i>		
enum	ETextHorizontalAlignment = {	
	thaLeft = 0x0,	<i>left alignment</i>
	thaCenter = 0x1 ,	<i>center alignment</i>
	thaRight = 0x2 ,	<i>right alignment</i>
	thaAligned = 0x3,	<i>custom alignment</i>
	thaMiddle = 0x4 ,	<i>middle alignment</i>
	thaFit = 0x5 }	<i>fitted alignment</i>
<i>Horizontal text alignment</i>		
enum	ETextVerticalAlignment = {	
	tvaBaseline = 0x0,	<i>aligned to baseline</i>
	tvaBottom = 0x1 ,	<i>aligned to bottom</i>
	tvaCenter = 0x2 ,	<i>aligned to center</i>
	thaAligned = 0x3,	<i>aligned to baseline</i>
	tvaTop = 0x4 }	<i>aligned to top</i>
<i>Vertical text alignment</i>		

Error codes

enum	ELayerError = {	
	laelInvalidName = -100001	<i>Emty string or name already exists</i>
	laelInvalidPenWidth = -100002	<i>width should be more than 0 and less than or equal 2.11</i>
	laelInvalidFontName = -100003	<i>Font not found</i>
{}		

Records / structures

	RLayerShapeAttributes = (
long	LayerIndex	<i>index of the layer where shape is</i>
unsigned long	Colour	<i>see Colour</i>
ELayerPenStyle	PenStyle	<i>pen style</i>
double	PenWidth	<i>width of the pen</i>
)		
	RLayerTextParams = (
long	LayerIndex	<i>index of the layer</i>
unsigned long	Colour	<i>see Colour</i>
ELayerPenStyle	PenStyle	<i>pen style</i>
double	PenWidth	<i>width of the pen</i>
double	FontSize	<i>height of the text</i>
double	Angle	<i>in radians</i>
RPoint3D	AlignmentPoint1	<i>Alignment point 1</i>
RPoint3D	AlignmentPoint2	<i>Alignment point 2</i>
RPoint3D	NormalVector	<i>Normal vector</i>
ETextHorizontalAlignment	HorizontalAlignment	<i>Type of horizontal alignment</i>
ETextVerticalAlignment	VerticalAlignment	<i>Type of vertical alignment</i>
)		

Functions

long **AddLayer** ([in] BSTR Name, [in] long Colour, [in] [ELayerPenStyle](#) PenStyle, [in] double PenWidth)

Name *Name of the new layer*

Colour *base colour of the layer*

PenStyle *style of the pen*

PenWidth *style of the pen*

Adds new AxisVM layer to the model. If successful, returns the layer index otherwise an error code (see [EGeneralError](#) or [ELayerError](#)).

long **AddShape** ([in] [ELayerShapeType](#) ShapeType, [in] [IAxisVMLines3d](#) * Polygon, [i/o] [RLayerShapeAttributes](#) LayerShapeAttributes)

ShapeType *type of the shape*

Polygon *the polygon or line defining the shape*

LayerShapeAttributes *attributes of the shape*

Adds new shape to the layer. If successful, returns the shape index otherwise an error code (see [EGeneralError](#) or [ELayerError](#)).

long **AddText** ([in] BSTR Text, [in] BSTR FontName, [i/o] [RLayerTextParams](#) TextParams)

Text *text*

FontName *name of the font, should be installed in MS Windows*

TextParams *parameters of the text*

Adds new text to the layer. If successful, returns the text index otherwise an error code (see [EGeneralError](#) or [ELayerError](#)).

long **DeleteLayer** ([in] long LayerIndex)

LayerIndex *index of the layer*

Deletes the layer. If successful, returns LayerIndex otherwise returns an error code (see [EGeneralError](#) or [ELayerError](#)).

long **DeleteShape** ([in] long ShapeIndex)

ShapeIndex *index of the shape*

Deletes the shape from a layer. If successful, returns ShapeIndex otherwise returns an error code (see [EGeneralError](#) or [ELayerError](#)).

long	DeleteText ([in] long TextIndex) TextIndex <i>index of the text</i> Deletes the text from a layer. If successful, returns TextIndex otherwise returns an error code (see EGeneralError or ELayerError).
long	GetShapeAttributes ([in] long ShapeIndex , [i/o] RLayerShapeAttributes LayerShapeAttributes) ShapeIndex <i>index of the shape</i> LayerShapeAttributes <i>attributes of the shape</i> Get attributes of the shape. If successful, returns the shape index otherwise an error code (see EGeneralError or ELayerError).
long	GetShapeIDs ([in] long LayerIndex , [out] SAFEARRAY (long) * ShapeIDs) LayerIndex <i>index of the layer</i> ShapeIDs <i>indexes of shapes on the layer</i> Get shape indexes. If successful, returns length of the array, otherwise an error code (see EGeneralError or ELayerError).
long	GetShapePolygon ([in] long ShapeIndex , [out] IAxisVMLines3d * Polygon) ShapeIndex <i>index of the shape</i> Polygon <i>the polygon or line defining the shape</i> Get shape polygon. If successful, returns shape index, otherwise an error code (see EGeneralError or ELayerError).
long	GetTextParams ([in] BSTR TextIndex , [i/o] RLayerTextParams TextParams) TextIndex <i>text index</i> TextParams <i>parameters of the text</i> Get parameters of the text. If successful, returns the text index otherwise an error code (see EGeneralError or ELayerError).
long	GetTextIDs ([in] long LayerIndex , [out] SAFEARRAY (long) * TextIDs) LayerIndex <i>index of the layer</i> TextIDs <i>indexes of texts on the layer</i> Get text indexes. If successful, returns length of the array, otherwise an error code (see EGeneralError or ELayerError).
long	SetShapeAttributes ([in] long ShapeIndex , [i/o] RLayerShapeAttributes LayerShapeAttributes) ShapeIndex <i>index of the shape</i> LayerShapeAttributes <i>attributes of the shape</i> Set attributes of the shape. If successful, returns the shape index otherwise an error code (see EGeneralError or ELayerError).
long	SetTextParams ([in] BSTR TextIndex , [i/o] RLayerTextParams TextParams) TextIndex <i>text index</i> TextParams <i>parameters of the text</i> Set parameters of the text. If successful, returns the text index otherwise an error code (see EGeneralError or ELayerError).

Properties

long	Count <i>Get number of layers.</i>
unsigned long	Colour [long LayerIndex] • Get or set the base colour of the layer with index LayerIndex .
BSTR	FontName [long TextIndex] • Get or set the font name of the text with index TextIndex .

<u>ElongBoolean</u>	IsEmpty [long LayerIndex] <i>Is lbTrue if the layer with index LayerIndex is empty.(read only)</i>
<u>ELayerType</u>	LayerType [long LayerIndex] <i>Get type of the layer with index LayerIndex.</i>
BSTR	Name [long LayerIndex] • <i>Get or set name of the layer with index LayerIndex.</i>
<u>ELayerPenStyle</u>	PenStyle [long LayerIndex] • <i>Get or set style of the pen of layer with index LayerIndex.</i>
double	PenWidth [long LayerIndex] • <i>Get or set width of the pen of layer with index LayerIndex.</i>
long	ShapesCount <i>Get total number of shapes in all layers.</i>
<u>ELayerShapeType</u>	ShapeType [long ShapeIndex] <i>Get type of the shape with index ShapeIndex.</i>
BSTR	Text [long TextIndex] • <i>Get or set text with index TextIndex.</i>
long	TextsCount <i>Get total number of texts in all layers.</i>

IAxisVMLines

Lines of the model.

Note:

All structural line elements: beams, ribs, trusses, edges are also lines.

Enumerated types

```
enum ELineGeomType = {
    IgStraightLine = 0x0,           straigth line
    IgCircleArc = 0x1 }             circle or arc
```

Line geometry.

Records / structures

```
RCircleArcGeomData = (
    RPoint3d          Center           arc centerpoint
    RPoint3d          NormalVector   arc plane normal
    double            Alfa            signed angle between the two endpoints [rad]. Positive angle is counter-clockwise if seen from a direction opposite to the plane normal. Alfa starts from the first point (noted as StartNode or Node1)
)

REllipseArcGeomData = (
    RPoint3d          Center           arc centerpoint
    RPoint3d          NormalVector   arc plane normal
    double            Alfa            signed angle between the two endpoints [rad]. Positive angle is counter-clockwise if seen from a direction opposite to the plane normal.
    double            Ratio            Ellipse ratio
    double            StartAlfa       angle between the two endpoints at begining
    double            EndAlfa         angle between the two endpoints at end
)

RLineGeomData = (
    CircleArc        CircleArc      arc geometry
    EllipseArc       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]
)

RLineData = (
    long             NodeId1        start node id
    long             NodeId2        end node id
    GeomType         GeomType       Type of line geometry (straight line, arc, ...)
    CircleArc        CircleArc      arc geometry, if the GeomType is IgCircleArc
    EllipseArc       EllipseArc    ellipse arc geometry (not used)
)

RLineAttr = (
    ELineType        LineType       type of the line. Depending on it only the relevant parts of the record will be taken into account
    MaterialIndex    MaterialIndex material index
    StartCrossSectionIndex StartCrossSectionIndex index of the starting cross section
    EndCrossSectionIndex EndCrossSectionIndex index of the ending cross section
    AutoEccentricityType AutoEccentricity eccentricity type of ribs
    StartEccentricity StartEccentricity starting eccentricity of ribs
    EndEccentricity   EndEccentricity ending eccentricity of ribs
    TrussType        TrussType     nonlinear behaviour of the truss
    Resistance       Resistance    axial resistance (absolute value) only for TrussTypes of InTensionOnly and InCompressionOnly
)

EAutoExcentricityType = (
    RPoint3D          ServiceClass   service class of timber (valid values are 1, 2, 3)
    RPoint3D          kdef          Kdef value (deformation factor for timber members)
    double            kx            friction resistance between rib and surface
    long              Domain1       domain index (filled if the rib connects to a domain)
    double            Domain2       domain index (filled if 2 domains are required to determine the rib's orientation, or as a backup domain, if a domain should still remain after the other one gets deleted)
)

EGapType = (
    GapType          GapType       gap type
    ActiveStiffness ActiveStiffness gap stiffness when active [kN/m]
    InactiveStiffness InactiveStiffness gap stiffness when inactive [kN/m]
    double           InitialOpening InitialOpening initial opening [m]
    double           MinPenetration MinPenetration minimum penetration [m]
)
```

	<pre>double MaxPenetration double AdjustmentRatio ESpringDirection SpringDirection RStiffnesses Stiffnesses)</pre>	<p><i>maximum penetration [m]</i> <i>adjustment ratio of the active stiffness</i> <i>coordinate system of spring stiffnesses</i> <i>stiffnesses</i></p>
Not all fields are active at the same time. For a detailed walkthrough of the valid field combinations for RLineAttr, check out IAxisVMLines.BulkSetAttr .		
Functions		
<p>long Add ([in] long StartNode, [in] long EndNode, [in] ELineGeomType GeomType, [i/o] RLineGeomData GeomData,)</p> <p>StartNode <i>index of the startpoint (0 < StartNode ≤ AxisVMNodes.Count)</i></p> <p>EndNode <i>index of the endpoint (0 < EndNode ≤ AxisVMNodes.Count)</i></p> <p>GeomType <i>line geometry</i></p> <p>GeomData <i>geometry data</i></p> <p><i>Adds a straight line or arc between two existing nodes. If successful, returns the line index, otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBounds).</i></p>		
<hr/> <p>long AddWithXYZ ([i/o] RPoint3d StartNode, [i/o] RPoint3d EndNode, [in] ELineGeomType GeomType, [i/o] RLineGeomData GeomData,)</p> <p>StartNode <i>startpoint coordinates</i></p> <p>EndNode <i>endpoint coordinates</i></p> <p>GeomType <i>line geometry</i></p> <p>GeomData <i>geometry data</i></p> <p><i>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).</i></p>		
<hr/> <p>long AddWithXYZDOF ([i/o] RPoint3d StartNode, [i/o] RPoint3d EndNode, [in] long StartDOF, [in] long EndDOF, [in] ELineGeomType GeomType, [i/o] RLineGeomData GeomData,)</p> <p>StartNode <i>startpoint coordinates</i></p> <p>EndNode <i>endpoint coordinates</i></p> <p>StartDOF <i>nodal degrees of freedom for the start node</i></p> <p>EndDOF <i>nodal degrees of freedom for the end node</i></p> <p>GeomType <i>line geometry</i></p> <p>GeomData <i>geometry data</i></p> <p><i>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.</i></p> <p><i>If successful, returns the line index, otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBounds).</i></p>		
<hr/> <p>long BulkAdd ([in] SAFEARRAY (RLineData) LineData, [out] SAFEARRAY (long) * Linelds)</p> <p>LineData <i>list of the line data records indexes of lines .The LineData Nodeld1 and Nodeld2 must satisfy: (1 ≤ Nodeld ≤ AxisVMNodes.Count)</i></p> <p>Linelds <i>indexes of the created lines .Each index will satisfy: (1 ≤ Index ≤ AxisVMLines.Count)</i></p> <p><i>Adds new lines to the model, between already existing nodes. If successful, returns the number of lines created, otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBounds, errInternalException, errOutOfMemory).</i></p>		
<hr/> <p>long BulkGetLineData ([in] SAFEARRAY(long) Linelds, [out] SAFEARRAY (RLineData) * LineData)</p> <p>Linelds <i>indexes of lines .Each index must satisfy: (1 ≤ Index ≤ AxisVMLines.Count)</i></p> <p>LineData <i>list of the line data records</i></p> <p><i>Queries the line data of a list of lines. If successful, returns the number of queried records, otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBounds, errInternalException, errOutOfMemory).</i></p>		
<hr/> <p>long BulkGetAttr ([in] SAFEARRAY(long) Linelds, [out] SAFEARRAY (RLineAttr) * LineAttr)</p> <p>Linelds <i>indexes of lines .Each index must satisfy: (1 ≤ Index ≤ AxisVMLines.Count)</i></p>		

	LineAttr <i>list of the line attribute records. Not all fields are active at the same time. For a detailed walkthrough of the valid field combinations for RLineAttr, check out IAxisVMLines.BulkSetAttr. In addition to the LineTypes handled in IAxisVMLines.BulkSetAttr, the rest of the possible LineTypes are returned in BulkGetLineAttr (like ItEdge, ItLLLink, ...). In those cases however none of the other fields will contain relevant data, and calling BulkSetAttr with such records will result in an error.</i>
	<i>Queries the line attributes of a list of lines. Depending on the LineType and MaterialIndex, only some combination of the record fields make sense. See BulkSetLineAttr for some valid combinations. ServiceClass and kdef makes only sense if the material is timber. If successful, returns the number of queried records, otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBounds, errInternalException, errOutOfMemory).</i>
long	BulkGetMemberIds ([in] SAFEARRAY (long) Linelds , [out] SAFEARRAY (long) * MemberIds) Linelds <i>list of the line indexes to query. Each index must satisfy: (1 ≤ Index ≤ AxisVMLines.Count)</i> MemberIds <i>list of the member indexes. Each index will satisfy: (0 ≤ Index ≤ AxisVMMembers.Count). 0 means the line is not part of a member.</i> <i>Queries the member identifiers of several lines at once. If successful, returns the number of lines queried, otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBounds, errInternalException, errOutOfMemory).</i>
long	BulkSetAttr ([in] SAFEARRAY(long) Linelds , [in] SAFEARRAY (RLineAttr) LineAttr) Linelds <i>indexes of lines .Each index must satisfy: (1 ≤ Index ≤ AxisVMLines.Count)</i> LineAttr <i>list of the line attribute records. Not all fields of the RLineAttr are active at the same time. The main factor influencing what is active is LineType, which always has to be specified. Only the following subset of LineType is valid : ItSimpleLine, ItTruss, ItBeam, ItRib, ItSpring, ItGap. Trying to set for example a LineType of ItLLLink will result in an error (however querying an already existing line with a LineType of ItLLLink through BulkGetLineAttr is valid).</i> <i>LineType= ItSimpleLine : no other field is required</i> <i>LineType=ItTruss : MaterialIndex, StartCrossSectionIndex=EndCrossSectionIndex (they must be equal), TrussType, Resistance. If MaterialIndex refers to timber, then also ServiceClass and kdef.</i> <i>LineType=ItBeam : MaterialIndex, StartCrossSectionIndex, EndCrossSectionIndex. If MaterialIndex refers to timber, then also ServiceClass and kdef.</i> <i>LineType=ItRib : MaterialIndex, StartCrossSectionIndex, EndCrossSectionIndex, AutoEccentricityType, StartEccentricity, EndEccentricity, kx. If the rib is attached to a domain then Domain1 and situationally Domain2. If MaterialIndex refers to timber, then also ServiceClass and kdef.</i> <i>LineType= ItSpring : SpringDirection, Stiffnesses.</i> <i>LineType= ItGap : GapType, ActiveStiffness, InactiveStiffness, InitialOpening, MinPenetration, MaxPenetration, AdjustmentRatio.</i> <i>If LineAttr MaterialIndex is active, it must satisfy: (1 ≤ MaterialIndex ≤ AxisVMMaterials.Count)</i> <i>If LineAttr StartCrossSectionIndex is active, it must satisfy: (1 ≤ StartCrossSectionIndex ≤ AxisVMCrossSections.Count)</i> <i>If LineAttr EndCrossSectionIndex is active, it must satisfy: (1 ≤ EndCrossSectionIndex ≤ AxisVMCrossSections.Count)</i> <i>If LineAttr Domain1 is active, it must satisfy: (0 ≤ Domain1 ≤ AxisVMDomains.Count)</i> <i>If LineAttr Domain2 is active, it must satisfy: (0 ≤ Domain2 ≤ AxisVMDomains.Count)</i> <i>If LineAttr ServiceClass is active, it must be 1, 2 or 3</i> <i>Sets the attributes of a list of lines. It must be used on already existing lines. To add new lines, first use the BulkAdd function, which creates them with a default attribute with LineType= ItSimpleLine, then set their attributes with this function. If the length of Indexes isn't equal to the length of LineAttr, an errIndexOutOfBounds will result. If successful, returns the number of queried records, otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBounds, errInternalException, errOutOfMemory, IneMaterialIndexOutOfBounds, IneCrossSectionIndexOutOfBounds, IneStartEndCrossSectionTypeIncompatible, IneIllegalServiceClassValue, IneDomainIndexOutOfBounds).</i>

long	BulkSetLineData ([in] SAFEARRAY(long) Linelds , [in] SAFEARRAY (RLineData) LineData)
	<p>Linelds indexes of lines .Each index must satisfy: $(1 \leq \text{Index} \leq \text{AxisVMLines.Count})$</p> <p>LineData list of the line data records. The LineData Nodeld1 and Nodeld2 must satisfy: $(1 \leq \text{Nodeld} \leq \text{AxisVMNodes.Count})$</p> <p>Sets the line data of a list of lines. It must be used on already existing lines. To Add new lines use the BulkAdd function instead. If the length of Indexes isn't equal to the length of LineData, an errIndexOutOfBounds will result. If successful, returns the number of queried records, otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBounds, errInternalException, errOutOfMemory).</p>
long	ChangeLocalDirection ([in] long Index)
	<p>Index line index ($0 < \text{Index} \leq \text{Count}$)</p> <p>Reverts local x direction. If end node indexes are i and j and local x direction points from i to j ChangeLocalDirection makes it point from j to i. If successful, returns the line index, otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBounds).</p>
long	Clear
	<p>Removes all lines. It returns the number of lines removed. If it returns a negative number, that is an error code (errDatabaseNotReady).</p>
long	CrossLines ([in] SAFEARRAY(long) Linelds)
	<p>Linelds indexes of lines to intersect</p> <p>Calculates intersection points for a set of lines. At intersection points nodes are inserted and lines are broken. If successful, returns the length of the array.</p> <p>Please note: Nodes, Lines etc. will be re-indexed. Use UID property if you want refer to them after this function.</p> <p>Possible error codes are errIndexOutOfBounds, errDatabaseNotReady, leEmptyLineList.</p>
long	CrossLines_vb (Visual Basic compatible function of CrossLines)
long	DefineSelectedLinesAsRigidBody
	<p>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 an error code (errDatabaseNotReady, leNoLinesAreSelected [see IAxisVMLine]).</p>
long	Delete ([in] double Index)
	<p>Index index of the line to delete</p> <p>Deletes a line. $1 \leq \text{Index} \leq \text{Count}$. If successful, returns the Index, otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBounds).</p>
long	DeleteNameOfAllTrusses
	<p>Deletes previously added default name of all trusses. If successful, returns number of lines. Otherwise returns an error code (errDatabaseNotReady).</p>
long	DeleteSelected
	<p>Deletes the selected lines. If successful, returns the number of deleted lines, otherwise returns an error code. (errDatabaseNotReady).</p>
long	GetFiniteElementCount ([in] ELineType LineType)
	<p>LineType Type of member</p> <p>Get number of FE line elements (beams, trusses or ribs) if successful, otherwise returns an error code (errDatabaseNotReady).</p>

long	GetContinuousLineIDs ([<i>i/o</i>] SAFEARRAY(long)* LineIDs , [<i>in</i>] double MaxAngleDifferenceX , [<i>in</i>] double MaxAngleDifferenceZ , [<i>out</i>] SAFEARRAY (long) * ContinousLineIDs)	
	LineIDs	array of line indices
	MaxAngleDifferenceX	maximum angle difference between angles of local X axes of adjacent lines
	MaxAngleDifferenceZ	maximum angle difference between angles of local Z axes of adjacent lines
	ContinousLineIDs	array of line indices that are continuous
<i>Get indices of continuous line indices. If successful It returns the number of continuous lines, otherwise it returns an error code (errDatabaseNotReady, IneEmptyLineList, IneLinesNotContinuous).</i>		
long	GetMidpoint ([<i>in</i>] long Index , [<i>i/o</i>] RPoint3d Value)	
	Index	line index ($0 < Line \leq Count$)
	Value	midpoint coordinates
<i>Retrieves the midpoint coordinates of a line. If successful, returns Index, otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBounds).</i>		
long	GetSelectedBeamIds ([<i>out</i>] SAFEARRAY (long) * Linelds)	
	Linelds	Index list of selected horizontal line elements
<i>Get indexes of selected horizontal line elements (beams, trusses or ribs). If successful, returns the number of selected horizontal elements, otherwise returns an error code (errDatabaseNotReady).</i>		
long	GetSelectedColumnIds ([<i>out</i>] SAFEARRAY (long) * Linelds)	
	Linelds	Index list of selected vertical line elements
<i>Get indexes of selected vertical line elements, If successful, returns the number of selected vertical line elements, otherwise returns an error code (errDatabaseNotReady).</i>		
long	GetSelectedItemIds ([<i>out</i>] SAFEARRAY (long) * ItemIds)	
	ItemIds	Index list of selected lines
<i>If successful, returns the number of selected lines otherwise returns an error code (errDatabaseNotReady).</i>		
long	GetSelectedOtherIds ([<i>out</i>] SAFEARRAY (long) * Linelds)	
	Linelds	list of selected neither vertical or horizontal lines
<i>Get indexes of selected neither vertical or horizontal lines. If successful, returns the number of selected neither vertical or horizontal lines, otherwise returns an error code (errDatabaseNotReady).</i>		
long	GetSurfaces ([<i>in</i>] long Lineld , [<i>out</i>] SAFEARRAY (long) * Surfaces)	
	Lineld	line index. It must satisfy: ($1 \leq Lineld \leq AxisVMLines.Count$)
	Surfaces	list of surfaces having LinelID as an edge. Each surface index will satisfy: ($1 \leq Index \leq AxisVMSurfaces.Count$)
<i>Retrieves the surfaces having LinelID as an edge. If successful, returns the number of attached surfaces. The possible error codes are (errDatabaseNotReady, errIndexOutOfBounds, errInternalException, errOutOfMemory).</i>		
long	IndexOf ([<i>in</i>] long StartNode , [<i>in</i>] long EndNode)	
	StartNode	start node index of the line ($0 < StartNode \leq AxisVMNodes.Count$)
	EndNode	end node index of the line ($0 < EndNode \leq AxisVMNodes.Count$)
<i>Retrieves the index of the line by its end nodes. If successful, returns the index of the line, otherwise returns an error code (errDatabaseNotReady, errNotFound).</i>		
long	IndexOf2 ([<i>in</i>] long StartNode , [<i>in</i>] long EndNode , [<i>in</i>] long FirstLine)	
	StartNode	start node index of the line ($0 < StartNode \leq AxisVMNodes.Count$)

EndNode end node index of the line
 $(0 < EndNode \leq \text{AxisVMNodes.Count})$
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 an error code ([errDatabaseNotReady](#), [errNotFound](#)).

long **IndexOfFiniteElementNumber** ([in] [ELineType](#) LineType, [in] long FiniteElementNumber)

LineType Type of member / line

FiniteElementNumber finite element number,
 $1 \leq \text{FiniteElementNumber} \leq \text{GetFiniteElementCount}$.
More info [here](#).

If successful it retrieves the index of the line by its finite element number, otherwise returns an error code ([errDatabaseNotReady](#), [InelInvalidLineTypeOrFENumber](#)).

long **SelectAll** ([in] [ELongBoolean](#) 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 an error code ([errDatabaseNotReady](#)).

NOTE: Call [Refresh](#) function afterwards if not called between functions [BeginUpdate](#) and [EndUpdate](#)

long **SelectAllColumns** ([in] [ELongBoolean](#) Select)

Select selection state

If Select is True, selects all vertical lines.

If Select is False, deselects all vertical lines.

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

NOTE: Call [Refresh](#) function afterwards if not called between functions [BeginUpdate](#) and [EndUpdate](#)

long **SelectAllBeams** ([in] [ELongBoolean](#) Select)

Select selection state

If Select is True, selects all horizontal lines.

If Select is False, deselects all horizontal lines.

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

NOTE: Call [Refresh](#) function afterwards if not called between functions [BeginUpdate](#) and [EndUpdate](#)

long **SelectAllOthers** ([in] [ELongBoolean](#) Select)

Select selection state

If Select is True, selects all neither horizontal or vertical lines.

If Select is False, deselects all neither horizontal or vertical lines.

If successful, returns the number of selected neither horizontal or vertical lines, otherwise returns an error code ([errDatabaseNotReady](#)).

NOTE: Call [Refresh](#) function afterwards if not called between functions [BeginUpdate](#) and [EndUpdate](#)

long **SelectAllColumnsAtStorey** ([in] [ELongBoolean](#) Select, [in] long StoreyId)

Select selection state

StoreyId storey index

If Select is True, selects all vertical lines at storey.

If Select is False, deselects all vertical lines at storey.

If successful, returns the number of selected vertical lines at storey with index StoreyId, otherwise returns an error code ([errDatabaseNotReady](#)).

NOTE: Call [Refresh](#) function afterwards if not called between functions [BeginUpdate](#) and [EndUpdate](#)

long	SelectAllBeamsAtStorey ([in] ELongBoolean Select , [in] long StoreyId)
	Select selection state
	StoreyId storey index
	<i>If Select is True, selects all horizontal lines at storey.</i>
	<i>If Select is False, deselects all horizontal lines at storey.</i>
	<i>If successful, returns the number of selected horizontal lines at storey with index StoreyId, otherwise returns an error code (errDatabaseNotReady).</i>
	<i>NOTE: Call Refresh function afterwards if not called between functions BeginUpdate and EndUpdate</i>
long	SelectAllOthersAtStorey ([in] ELongBoolean Select , [in] long StoreyId)
	Select selection state
	StoreyId storey index
	<i>If Select is True, selects all neither horizontall or verical lines at storey.</i>
	<i>If Select is False, deselects all neither horizontall or verical lines at storey.</i>
	<i>If successful, returns the number of selected neither horizontall or verical lines at storey with index StoreyId, otherwise returns an error code (errDatabaseNotReady).</i>
	<i>NOTE: Call Refresh function afterwards if not called between functions BeginUpdate and EndUpdate</i>

NOTE: To rename selected beams and ribs see functions of *IAxisVMMembers*

Properties

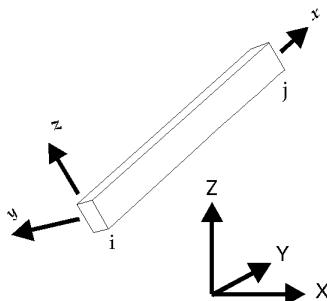
<u>AxisVMAAttachments</u> *	Attachments Get the attachments interface
<u>AxisVMAAttributes</u> *	Attributes Get the attributes interface
long	Count number of lines in the model
<u>AxisVMLines</u> *	Item [long Index] line interface by index
long	IndexOfUID [long UID] Get index of the line
	UID unique index of the line
long	LineByMidpoint [long Index] Get the index of the line by the given midpoint index. (Index > <u>AxisVMLines.Count</u>)
<u>ELongBoolean</u>	LocalX_is_ij [long Index] Is lbTrue if local x axis of the line is in i-j direction (Index > <u>AxisVMLines.Count</u>)
long	MidpointCount number of line midpoints (it is the sum of the number of ribs and surface edges)
long	MidpointDOF [long Index] • Get or set the degrees of freedom for line midpoint
long	MidpointId [long Index] Get the line midpoint index
BSTR	Name [long Index] Get name of the line
	Index index of the line
long	RigidBodyCount number of lines defined as rigid bodies
<u>ELongBoolean</u>	Selected [long Index] • Get or set the selection status of a line <i>NOTE:</i> Call <u>Refresh</u> function afterwards if not called between functions <u>BeginUpdate</u> and <u>EndUpdate</u>
long	SelCount number of selected lines in the model
double	StiffnessReduction_A [long Index] • Get or set axial stiffness factor. Cross sections area is multiplied with this value in stiffness matrix. Default value is 1. Used only for atLinearStatic and atLinearVibration analysis <u>types</u> . (only valid for STAS and Eurocode [RO] standards)
double	StiffnessReduction_I [long Index] • Get or set flexural stiffness factor. Cross sections inertia is multiplied with this value in stiffness matrix. Default value is 1. Used only for atLinearStatic and atLinearVibration analysis <u>types</u> . (only valid for STAS and Eurocode [RO] standards)
long	UID [long Index] Get unique index of the line which remains the same while exists in the model
	Index index of the line

IAxisVMLine

AxisVM line interface.

Note:

In this interface you can define rib or beam from a line.



Warning : after a line gets deleted either directly (for example through `IAxisVMLines.Delete`) or indirectly (for example through `IAxisVMLine.SplitByN`), its previously acquired `IAxisVMLine` interface will become invalid.

Enumerated types

enum	EArchitectElemType = {	
	aetAuto = 0,	AxisVM automatically assigns proper value
	aetColumn = 1,	Architectural type is column, only for lines and structural members.
	aetBeam = 2,	Architectural type is beam, only for lines and structural members.
	aetWall = 3,	Architectural type is wall, only for surfaces and domains.
	aetSlab = 4 }	Architectural type is slab, only for surfaces and domains.
enum	EColour = {	
	cColourByMaterial =	Colour depends on assigned material colour
	0xFFFFFFFF }	
enum	EGapType = {	
	agtActiveInTension = 0x0,	gap active in tension only
	agtActiveInCompression = 0x1 }	gap active in compression only
		Gap element type.
enum	ELineNonlinearity = {	
	lntTensionAndCompression =	linear behaviour
	0x0,	
	lntTensionOnly = 0x1,	tension only
	lntCompressionOnly = 0x2 }	compression only
		Types of nonlinear behaviour.
enum	ELineType = {	
	ltTruss = 0x0,	truss
	ltBeam = 0x1,	beam
	ltRib = 0x2,	rib
	ltSpring = 0x3,	spring
	ltGap = 0x4,	gap element
	ltEdge = 0x5,	surface edge
	ltHole = 0x6,	hole edge
	ltSimpleLine = 0x7,	line without element properties
	ltNNLink = 0x8,	node-to-node link element
	ltLLLink = 0x9 }	line-to-line link element
		Structural type of the line.

Note: To check if it is part of any rigid body check `RigidBodyId` property

enum	EReleaseType = {	
	rtRigid = 0x0,	<i>rigid</i>
	rtHinged = 0x1,	<i>hinged</i>
	rtSemiRigid = 0x2,	<i>semi-rigid</i>
	rtPlastic = 0x3,	<i>plastic</i>
	rtPushover = 0x4 }	<i>for push over</i>
	<i>Type of end release.</i>	
enum	ESpringDirection = {	
	sdGlobal = 0x0,	<i>global</i>
	sdGeometry = 0x1,	<i>by geometry</i>
	sdPointReference = 0x2,	<i>by reference point</i>
	sdVectorReference = 0x3,	<i>by reference vector</i>
	sdElementRelative = 0x4,	<i>by element</i>
	sdNodeRelative = 0x5 }	<i>by node</i>
	<i>Coordinate system for spring stiffness components.</i>	
enum	EAutoExcentricityType = {	
	aetCustom = 0x0,	<i>Custom eccentricity</i>
	aetTop = 0x1,	<i>Bottom of the rib aligned with top of the domain</i>
	aetMid = 0x2,	<i>Centre of the rib aligned with centre of the domain</i>
	aetBottom = 0x3 }	<i>Top of the rib aligned with bottom of the domain</i>
	<i>Type of auto eccentricity for ribs</i>	

Error codes

enum	ELineError = {	
	IneNodeIndexOutOfBounds = -100001	<i>node index is out of bounds</i>
	IneReferenceIndexOutOfBounds = -100002	<i>reference index is out of bounds</i>
	IneReadOnlyPropertyForThisLineType = -100003	<i>the property is read-only for this line type</i>
	InePropertyNotValidForThisLineType = -100004	<i>the property is not compatible with this line type</i>
	IneMaterialIndexOutOfBounds = -100005	<i>material index is out of bounds</i>
	IneCrossSectionIndexOutOfBounds = -100006	<i>cross-section index is out of bounds</i>
	IneNoLinesAreSelected = -100007	<i>no lines are selected</i>
	IneLineHasNoMidPoint = -100008	<i>line has no midpoint (i.e. it is not a rib or edge)</i>
	IneEmptyLineList = -100009	<i>resulting line list is empty</i>
	IneSectionIndexOutOfBounds = -100010	<i>invalid section index</i>
	IneNotBeam = -100011	<i>the line is not a beam</i>
	IneNotGap = -100012	<i>the line is not a gap element</i>
	IneNotRib = -100013	<i>the line is not a rib element</i>
	IneNotSpring = -100014	<i>the line is not a spring element</i>
	IneNotTruss = -100015	<i>the line is not a truss element</i>
	IneNodeNotOnLine = -100016	<i>node is not on the line</i>
	IneErrorSplittingLine = -100017	<i>splitting lines was not successful</i>
	IneNMustBeGreaterThan1 = -100018	<i>splitting by value n was less than 1</i>
	IneIllegalServiceClassValue = -100019	<i>service class value of the timber is incorrect</i>
	IneDomainIndexOutOfBounds = -100020	<i>provided DomainID was incorrect using functions: IAxisVMLine.DefineAsRibWithAutoExcentricity and IAxisVMLine.DefineAsTimberRibWithAutoExcentricity</i>
	IneStoreyIdOutOfBounds = -100021	<i>storey index invalid</i>
	IneInvalidLineType = -100022	<i>invalid line type</i>
	IneReinforcementParametersNotExist = -100023	<i>reinforcement parameters not exist</i>
	IneInvalidColumnRebarsId = -100024	<i>invalid column rebar index</i>
	IneInvalidConcreteMaterialId = -100025	<i>invalid concrete material index</i>
	IneInvalidRebarSteelGradeId = -100026	<i>invalid rebar steel grade index</i>
	IneInvalidRelease = -100027	<i>other release error</i>
	IneInvalidLineTypeOrFENumber = -100028	<i>invalid line type</i>
	IneInvalidFunctionIdOfRelease = -100029	<i>invalid function index related to the DOF</i>
	IneReleaseInitAndLimitMustBe0 = -100030	<i>Initial rotational stiffness and moment resistance of the release must be 0</i>
	IneFunctionIdMustBe0 = -100031	<i>FunctionId of the release must be 0</i>
	IneLinesNotContinuous = -100032	<i>Lines in the array do not compose a continuous line</i>
	IneStartEndCrossSectionTypeIncompatible = -100033	<i>can not generate various custom cross-sections</i>
	IneInvalidRCCheckingParameters = -100034	<i>at least one of the RC checking parameters is invalid</i>
	IneRCShrinkageEpsMustBePositive = -100035	<i>shrinkage strain must be positive</i>
	IneStirrupParametersAreInvalid = -100036	<i>stirrup parameters are invalid</i>
	IneShearCrackAngleIsInvalid = -100037	<i>defined shear crack angle is too low/high</i>
	IneInvalidSteelMaterialId = -100038 }	<i>invalid steel grade index</i>

Records / structures

RCircleArcGeomData	RLineGeomData = (
REllipseArcGeomData	CircleArc <i>arc geometry</i>
	EllipseArc <i>ellipse arc geometry (not used)</i>
)
double	RPoint3d = (
	x, y, z <i>x, y, z coordinates of a a 3D point or components of a 3D vector [m]</i>
)
EReleaseType	RRelease = (
double	Release <i>release type (rigid / hinged / semi-rigid / plastic)</i>
double	Init <i>(if Release = rtSemiRigid) initial rotational stiffness of the connection [kNm/rad]</i>
double	Limit <i>(if Release = rtSemiRigid) moment resistance of the connection [kNm]</i>
long	FunctionId <i>index of the used (pushover hinge) function, see here</i>
)
	RReleases = (
RRelease	x <i>release in x direction (x.Release = rtRigid v. rtHinged)</i>
RRelease	y <i>release in y direction (y.Release = rtRigid v. rtHinged)</i>
RRelease	z <i>release in z direction (z.Release = rtRigid v. rtHinged)</i>
RRelease	xx <i>release around the x axis (xx.Release = rtRigid v. rtHinged)</i>
RRelease	yy <i>release around the y axis</i>
RRelease	zz <i>release around the z axis</i>
)
	RStiffnesses = (
double	x, y, z <i>stiffness in x, y, z direction [kN/m]</i>
double	xx, yy, zz <i>rotational stiffness around x, y, z axes [kNm/rad]</i>
)

Functions

long **ChangeLocalDirection**

Reverts local x direction. If end node indexes are i and j and local x direction points from i to j ChangeLocalDirection makes it point from j to i. If successful, returns the line index, otherwise returns an error code ([errDatabaseNotReady](#), [errIndexOutOfBounds](#)).

long **DefineAsBeam** ([in] long **MaterialIndex**, [in] long **StartCrossSectionIndex**,
[in] long **EndCrossSectionIndex**, [i/o] **RPoint3d** **StartEccentricity**,
[i/o] **RPoint3d** **EndEccentricity**)

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

StartCrossSectionIndex *index of the start cross-section (according to local x direction)
(0 < StartCrossSectionIndex ≤ [AxisVMCrossSections.Count](#))*

EndCrossSectionIndex *index of the end cross-section(according to local x direction)
(0 < EndCrossSectionIndex ≤ [AxisVMCrossSections.Count](#))*

StartEccentricity *eccentricity at the start node (not used)*
EndEccentricity *eccentricity at the end node (not used)*

Defines a line as a beam. For beams with a constant cross-section StartCrossSectionIndex = EndCrossSectionIndex. New member is also created with ID which can be read from property [MemberId](#)

If successful, returns the line index according to [IAxisVMLines](#) , otherwise returns an 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
	ActiveStiffness gap stiffness when active [kN/m]
	InactiveStiffness gap stiffness when inactive [kN/m]
	InitialOpening initial opening [m]
	MinPenetration minimum penetration [m]
	MaxPenetration maximum penetration [m]
	AdjustmentRatio adjustment ratio of the active stiffness
	<i>Defines a line as a gap element.</i>
	<i>If successful, returns the line index according to IAxisVMLines, otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBoundsException).</i>
long	DefineAsRib ([in] long MaterialIndex, [in] long StartCrossSectionIndex, [in] long EndCrossSectionIndex, [i/o] RPoint3d StartEccentricity, [i/o] RPoint3d EndEccentricity)
	MaterialIndex index of the material ($0 < \text{MaterialIndex} \leq \text{AxisVMMaterials.Count}$)
	StartCrossSectionIndex index of the start cross-section (according to local x direction) ($0 < \text{StartCrossSectionIndex} \leq \text{AxisVMCrossSections.Count}$)
	EndCrossSectionIndex index of the end cross-section (according to local x direction) ($0 < \text{EndCrossSectionIndex} \leq \text{AxisVMCrossSections.Count}$)
	StartEccentricity eccentricity at the start node (only the z value is used)
	EndEccentricity eccentricity at the end node (only the z value is used)
	<i>Defines a line as a rib. For ribs with a constant cross-section StartCrossSectionIndex = EndCrossSectionIndex. New member is also created with ID which can be read from property MemberId</i>
	<i>If successful, returns the line index according to IAxisVMLines, otherwise returns an error code (eCrossSectionIndexOutOfBoundsException, eMaterialIndexOutOfBoundsException, errDatabaseNotReady, errIndexOutOfBoundsException).</i>
long	DefineAsSpring ([in] ESpringDirection SpringDirection, [i/o] RStiffnesses Stiffnesses)
	SpringDirection coordinate system of spring stiffnesses
	Stiffnesses stiffnesses
	<i>Defines a line as a spring element.</i>
	<i>If successful, returns the line index according to IAxisVMLines, otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBoundsException).</i>
long	DefineAsTruss ([in] long MaterialIndex, [in] long CrossSectionIndex, [in] ELineNonLinearity TrussType, [in] double Resistance)
	MaterialIndex index of the material ($0 < \text{MaterialIndex} \leq \text{AxisVMMaterials.Count}$)
	CrossSectionIndex index of the cross-section ($0 < \text{CrossSectionIndex} \leq \text{AxisVMCrossSections.Count}$)
	TrussType nonlinear behaviour
	Resistance Axial resistance (absolute value) only for <i>InTensionOnly</i> and <i>InCompressionOnly</i> types of truss
	<i>Defines a line as a truss element. New member is also created with ID which can be read from property MemberId</i>
	<i>If successful, returns the line index according to IAxisVMLines, otherwise returns an error code (eCrossSectionIndexOutOfBoundsException, eMaterialIndexOutOfBoundsException, errDatabaseNotReady, errIndexOutOfBoundsException).</i>

long **DefineAsTimberTruss** ([in] long **MaterialIndex**, [in] long **ServiceClass**, [in] double **kdef**,
[in] long **CrossSectionIndex**, [in] **ELineNonLinearity** **TrussType**, [in] double **Resistance**)

MaterialIndex Material Index

ServiceClass Service class of timber (valid values are 1, 2, 3)

kdef Kdef value (deformation factor for timber members)

CrossSectionIndex CrossSection index

TrussType (non)linearity type of the truss

Resistance Axial resistance (absolute value) only for *InTensionOnly* and
InCompressionOnly types of truss.

Defines a line as a timber truss element. New member is also created with ID which can be read from property [MemberId](#)

If successful, returns line index, otherwise returns an error code ([errDatabaseNotReady](#),
[errIndexOutOfBounds](#), [leMaterialIndexOutOfBounds](#), [leCrossSectionIndexOutOfBounds](#),
[leIllegalServiceClassValue](#)).

long **DefineAsTimberBeam** ([in] long **MaterialIndex**, [in] long **ServiceClass**, [in] double **kdef**,
[in] long **StartCrossSectionIndex**, [in] long **EndCrossSectionIndex**,
[i/o] **RPoint3d** **StartExcentricity**, [i/o] **RPoint3d** **EndExcentricity**)

MaterialIndex Material Index

ServiceClass Service class of timber (valid values are 1, 2, 3)

kdef Kdef value (deformation factor for timber members)

StartCrossSectionIndex CrossSection index at the beginning of the line (according to local x direction)

EndCrossSectionIndex CrossSection index at the end of the line (according to local x direction)

StartExcentricity Eccentricity at the beginning of the line (according to local x direction)

EndExcentricity Eccentricity at the end of the line (according to local x direction)

Defines a line as a timber beam element. New member is also created with ID which can be read from property [MemberId](#)

If successful, returns line index, otherwise returns an error code ([errDatabaseNotReady](#),
[errIndexOutOfBounds](#), [leMaterialIndexOutOfBounds](#), [leCrossSectionIndexOutOfBounds](#),
[leIllegalServiceClassValue](#)).

long **DefineAsTimberRib** ([in] long **MaterialIndex**, [in] long **ServiceClass**, [in] double **kdef**,
[in] long **StartCrossSectionIndex**, [in] long **EndCrossSectionIndex**,
[i/o] **RPoint3d** **StartExcentricity**, [i/o] **RPoint3d** **EndExcentricity**)

MaterialIndex Material Index

ServiceClass Service class of timber (valid values are 1, 2, 3)

kdef Kdef value (deformation factor for timber members)

StartCrossSectionIndex CrossSection index at the beginning of the line (according to local x direction)

EndCrossSectionIndex CrossSection index at the end of the line (according to local x direction)

StartExcentricity Eccentricity at the beginning of the line (according to local x direction)

EndExcentricity Eccentricity at the end of the line (according to local x direction)

Defines a line as a timber rib element. New member is also created with ID which can be read from property [MemberId](#)

If successful, returns line index, otherwise returns an error code ([errDatabaseNotReady](#),
[errIndexOutOfBounds](#), [leMaterialIndexOutOfBounds](#), [leCrossSectionIndexOutOfBounds](#),
[leIllegalServiceClassValue](#)).

long	DefineAsRibWithAutoExcentricity ([in] long MaterialIndex , [in] long StartCrossSectionIndex , [in] long EndCrossSectionIndex , [in] EAutoExcentricityType AutoExcentricityType , [in] double kx , [in] long Domain1 , [in] long Domain2)
	MaterialIndex <i>Material Index</i>
	StartCrossSectionIndex <i>CrossSection index at the beginning of the line (according to local x direction)</i>
	EndCrossSectionIndex <i>CrossSection index at the end of the line (according to local x direction)</i>
	AutoExcentricityType <i>Type of the automatic eccentricity</i>
	kx <i>Friction resistance between rib and surface</i>
	Domain1 <i>Domain index (filled if connected to domain)</i>
	Domain2 <i>Domain index (filled if rib is between two domains)</i>
	<i>Defines a line as a rib element, where eccentricity is calculated using domain thicknesses (Domain1,Domain2). New member is also created with ID which can be read from property MemberId</i>
	<i>If successful, returns line index, otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBounds, leMaterialIndexOutOfBounds, leCrossSectionIndexOutOfBounds, leIllegalServiceClassValue, lineerrDomainIndexOutOfBounds).</i>
long	DefineAsTimberRibWithAutoExcentricity ([in] long MaterialIndex , [in] long ServiceClass , [in] double kdef , [in] long StartCrossSectionIndex , [in] long EndCrossSectionIndex , [in] EAutoExcentricityType AutoExcentricityType , [in] double kx , [in] long Domain1 , [in] long Domain2)
	MaterialIndex <i>Material Index</i>
	ServiceClass <i>Service class of timber (valid values are 1, 2, 3)</i>
	kdef <i>Kdef value (deformation factor for timber members)</i>
	StartCrossSectionIndex <i>CrossSection index at the beginning of the line (according to local x direction)</i>
	EndCrossSectionIndex <i>CrossSection index at the end of the line (according to local x direction)</i>
	AutoExcentricityType <i>Type of the automatic eccentricity</i>
	kx <i>Friction resistance between rib and surface</i>
	Domain1 <i>Domain index (filled if connected to domain)</i>
	Domain2 <i>Domain index (filled if rib is between two domains)</i>
	<i>Defines a line as a timber rib element, where eccentricity is calculated using domain thicknesses (Domain1,Domain2). New member is also created with ID which can be read from property MemberId</i>
	<i>If successful, returns line index, otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBounds, leMaterialIndexOutOfBounds, leCrossSectionIndexOutOfBounds, leIllegalServiceClassValue, lineerrDomainIndexOutOfBounds).</i>
long	DeleteColumnReinforcementParameters
	<i>If successful, returns line index, otherwise returns an error code (errDatabaseNotReady).</i>
long	DeleteLineElement
	<i>Delete all assigned properties (cross-sections, materials, etc) from the line. If successful, returns line index, otherwise returns an error code (errDatabaseNotReady).</i>

long	GetBeamData ([out] long MaterialIndex , [out] long StartCrossSectionIndex , [out] long EndCrossSectionIndex , [i/o] RPoint3d StartEccentricity , [i/o] RPoint3d EndEccentricity)
	<p style="margin-left: 20px;">MaterialIndex <i>index of the material ($0 < \text{MaterialIndex} \leq \text{AxisVMMaterials.Count}$)</i></p> <p style="margin-left: 20px;">StartCrossSectionIndex <i>index of the start cross-section (according to local x direction) ($0 < \text{StartCrossSectionIndex} \leq \text{AxisVMCrossSections.Count}$)</i></p> <p style="margin-left: 20px;">EndCrossSectionIndex <i>index of the end cross-section (according to local x direction) ($0 < \text{EndCrossSectionIndex} \leq \text{AxisVMCrossSections.Count}$)</i></p> <p style="margin-left: 20px;">StartEccentricity <i>eccentricity at the start node (not used)</i></p> <p style="margin-left: 20px;">EndEccentricity <i>eccentricity at the end node (not used)</i></p>
	<p><i>Get properties of a beam. If successful, returns the line index according to IAxisVMLines, otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBounds, leNotBeam).</i></p>
long	GetColumnReinforcementParameters ([i/o] RColumnReinforcementParameters ColumnReinforcementParameters)
	<p style="margin-left: 20px;">ColumnReinforcementParameters <i>column reinforcement parameters</i></p>
	<p><i>If successful, returns the line index, otherwise returns an error code (errDatabaseNotReady, InReinforcementParametersNotExist, InInvalidLineType).</i></p>
long	GetEndReleases ([i/o] RReleases Value)
	<p><i>Get the end releases (only if LineType = ItBeam or ItRib). If successful, returns the line index according to IAxisVMLines, otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBounds).</i></p>
long	GetGapData ([out] EGapType GapType , [out] double ActiveStiffness , [out] double InactiveStiffness , [out] double InitialOpening , [out] double MinPenetration , [out] double MaxPenetration , [out] double AdjustmentRatio)
	<p style="margin-left: 20px;">GapType <i>gap type</i></p> <p style="margin-left: 20px;">ActiveStiffness <i>stiffness of the elemect when it is active</i></p> <p style="margin-left: 20px;">InactiveStiffness <i>stiffness of the elemect when it is inactive</i></p> <p style="margin-left: 20px;">InitialOpening <i>if GapType = agtActiveInTension: initial penetration if GapType = agtActiveInCompression: initial opening</i></p> <p style="margin-left: 20px;">MinPenetration <i>if auto active stiffness adjustment is on stiffness is reduced if penetration is under this value</i></p> <p style="margin-left: 20px;">MaxPenetration <i>if auto active stiffness adjustment is on stiffness is multiplied if penetration is above this value</i></p> <p style="margin-left: 20px;">AdjustmentRatio <i>factor used in auto active stiffness adjustment. Stiffness is divided by this factor under MinPenetration and multiplied by this factor above MaxPenetration. Between MinPenetration and MaxPenetration ActiveStiffness remains unchanged.</i></p>
	<p><i>Get properties of a gap element. If successful, returns the line index according to IAxisVMLines, otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBounds, leNotGap).</i></p>
long	GetGeomData ([i/o] RLineGeomData Value)
	<p><i>Get the geometry data for the arc (only if GeomType = lgtCircleArc). If successful, returns the line index according to IAxisVMLines, otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBounds, lePropertyNotValidForThisLineType).</i></p>
long	GetMidpoint ([i/o] RPoint3d Value)
	<p><i>Retrieves the midpoint coordinates. If successful, returns the line index according to IAxisVMLines, otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBounds).</i></p>

long	GetRibData ([out] long MaterialIndex , [out] long StartCrossSectionIndex , [out] long EndCrossSectionIndex , [i/o] RPoint3d StartEccentricity , [i/o] RPoint3d EndEccentricity)
	<p>MaterialIndex <i>index of the material ($0 < \text{MaterialIndex} \leq \text{AxisVMMaterials.Count}$)</i></p> <p>StartCrossSectionIndex <i>index of the start cross-section (according to local x direction) ($0 < \text{StartCrossSectionIndex} \leq \text{AxisVMCrossSections.Count}$)</i></p> <p>EndCrossSectionIndex <i>index of the end cross-section (according to local x direction) ($0 < \text{EndCrossSectionIndex} \leq \text{AxisVMCrossSections.Count}$)</i></p> <p>StartEccentricity <i>eccentricity at the beginning(according to local x direction)</i></p> <p>EndEccentricity <i>eccentricity at the end (according to local x direction)</i></p>
	<i>Get properties of a rib. If successful, returns the line index according to IAxisVMLines, otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBounds, leNotRib).</i>
long	GetRibDataWithAutoExcentricity ([out] long MaterialIndex , [out] long StartCrossSectionIndex , [out] long EndCrossSectionIndex , [out] EAutoExcentricityType AutoExcentricityType , [out] double kx , [out] long Domain1 , [out] long Domain2)
	<p>MaterialIndex <i>Material Index</i></p> <p>StartCrossSectionIndex <i>CrossSection index at the beginning of the line (according to local x direction)</i></p> <p>EndCrossSectionIndex <i>CrossSection index at the end of the line (according to local x direction)</i></p> <p>AutoExcentricityType <i>Type of the automatic eccentricity</i></p> <p>kx <i>Friction resistance between rib and surface</i></p> <p>Domain1 <i>Domain index (filled if connected to domain)</i></p> <p>Domain2 <i>Domain index (filled if rib is between two domains)</i></p>
	<i>If successful, returns line index, otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBounds, leNotRib).</i>
long	GetSpringData ([out] ESpringDirection SpringDirection , [out] RStiffnesses Stiffnesses)
	<p>SpringDirection <i>spring direction</i></p> <p>Stiffnesses <i>spring stiffnesses</i></p>
	<i>Get properties of a spring. If successful, returns the line index according to IAxisVMLines, otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBounds, leNotSpring).</i>
long	GetStartReleases ([i/o] RReleases Value)
	<i>Get the start releases (only if LineType = ItBeam or ItRib). If successful, returns the line index according to IAxisVMLines, otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBounds).</i>
long	GetTrussData ([out] long MaterialIndex , [out] long CrossSectionIndex , [out] ELineNonlinearity TrussType , [out] double Resistance)
	<p>MaterialIndex <i>index of the material ($0 < \text{MaterialIndex} \leq \text{AxisVMMaterials.Count}$)</i></p> <p>CrossSectionIndex <i>index of the cross-section ($0 < \text{CrossSectionIndex} \leq \text{AxisVMCrossSections.Count}$)</i></p> <p>TrussType <i>nonlinear behaviour of the truss</i></p> <p>Resistance <i>Axial resistance (absolute value) only for InTensionOnly and InCompressionOnly types of truss</i></p>
	<i>Get properties of a truss. If successful, returns the line index according to IAxisVMLines, otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBounds, leNotTruss).</i>

long	GetTimberTrussData ([out] long MaterialIndex, [out] long ServiceClass, [out] double kdef, [out] long CrossSectionIndex, [out] ELineNonLinearity TrussType, [out] double Resistance)
	MaterialIndex Material Index ServiceClass Service class of timber (valid values are 1, 2, 3) kdef Kdef value (deformation factor for timber members) CrossSectionIndex CrossSection index TrussType nonlinear behaviour of the truss Resistance Axial resistance (absolute value) only for InTensionOnly and InCompressionOnly types of truss
	<i>If successful, returns line index, otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBounds, leNotTruss).</i>
long	GetTimberBeamData ([out] long MaterialIndex, [out] long ServiceClass, [out] double kdef, [out] long StartCrossSectionIndex, [out] long EndCrossSectionIndex, [i/o] RPoint3d StartExcentricity, [i/o] RPoint3d EndExcentricity)
	MaterialIndex Material Index ServiceClass Service class of timber (valid values are 1, 2, 3) kdef Kdef value (deformation factor for timber members) StartCrossSectionIndex CrossSection index at the beginning of the line (according to local x direction) EndCrossSectionIndex CrossSection index at the end of the line (according to local x direction) StartExcentricity Eccentricity at the beginning (according to local x direction) EndExcentricity Eccentricity at the end (according to local x direction)
	<i>If successful, returns line index, otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBounds, leNotBeam).</i>
long	GetTimberRibData ([out] long MaterialIndex, [out] long ServiceClass, [out] double kdef, [out] long StartCrossSectionIndex, [out] long EndCrossSectionIndex, [i/o] RPoint3d StartExcentricity, [i/o] RPoint3d EndExcentricity)
	MaterialIndex Material Index ServiceClass Service class of timber (valid values are 1, 2, 3) kdef Kdef value (deformation factor for timber members) StartCrossSectionIndex CrossSection index at the beginning of the line (according to local x direction) EndCrossSectionIndex CrossSection index at the end of the line (according to local x direction) StartExcentricity Eccentricity at the beginning (according to local x direction) EndExcentricity Eccentricity at the end (according to local x direction)
	<i>If successful, returns line index, otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBounds, leNotRib).</i>

long	GetTimberRibDataWithAutoExcentricity ([out] long MaterialIndex , [out] long ServiceClass , [out] double kdef , [out] long StartCrossSectionIndex , [out] long EndCrossSectionIndex , [out] EAutoExcentricityType AutoExcentricityType , [out] double kx , [out] long Domain1 , [out] long Domain2)
	MaterialIndex <i>Material Index</i>
	ServiceClass <i>Service class of timber (valid values are 1, 2, 3)</i>
	kdef <i>Kdef value (deformation factor for timber members)</i>
	StartCrossSectionIndex <i>CrossSection index at the beginning of the line (according to local x direction)</i>
	EndCrossSectionIndex <i>CrossSection index at the end of the line (according to local x direction)</i>
	AutoExcentricityType <i>Type of the automatic eccentricity</i>
	kx <i>Friction resistance between rib and surface</i>
	Domain1 <i>Domain index (filled if connected to domain)</i>
	Domain2 <i>Domain index (filled if rib is between two domains)</i>
	<i>If successful, returns line index, otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBounds, leNotRib).</i>
long	SetColumnReinforcementParameters ([i/o] RColumnReinforcementParameters ColumnReinforcementParameters)
	ColumnReinforcementParameters <i>column reinforcement parameters</i>
	<i>If successful, returns the line index, otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBounds, IneInvalidConcreteMaterialId, IneInvalidRebarSteelGradeId, IneInvalidConcreteMaterialId, IneInvalidColumnRebarsId, IneInvalidLineType).</i>
long	SetEndReleases ([i/o] RReleases Value)
	<i>Set the end releases (only if LineType = ItBeam or ItRib).</i>
	<i>If successful, returns the line index according to IAxisVMLines, otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBounds).</i>
long	SetGeomData ([i/o] RLineGeomData Value)
	<i>Set the geometry data for the arc (only if GeomType = lgtCircleArc).</i>
	<i>If successful, returns the line index according to IAxisVMLines, otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBounds, lePropertyNotValidForThisLineType).</i>
long	SetGeomType ([in] ELineGeomType GeomType , [i/o] RLineGeomData Value)
	<i>Set the geometry type and the data for the arc (only if GeomType = lgtCircleArc).</i>
	<i>If successful, returns the line index according to IAxisVMLines, otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBounds, lePropertyNotValidForThisLineType).</i>
long	SetStartReleases ([i/o] RReleases Value)
	<i>Set the start releases (only if LineType = ItBeam or ItRib).</i>
	<i>If successful, returns the line index according to IAxisVMLines, otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBounds).</i>
long	SplitByNode ([in] long Nodeld , [out] long Lineld1 , [out] long Lineld2)
	Nodeld <i>Index of node which will dictate the location of split</i>
	Lineld1 <i>Line index of first part after split</i>
	Lineld2 <i>Line index of second part after split</i>
	<i>If successful, returns 1, otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBounds, leNodeIndexOutOfBounds, leNodeNotOnLine, leErrorSplittingLine).</i>
	<i>After a successful SplitByNode the IAxisVMLine interface will become invalid (the original line gets deleted), it shouldn't be used for further interaction.</i>
long	SplitByN ([in] long N)
	N <i>Number of parts after line split</i>

If successful, returns 1, otherwise returns an error code ([errDatabaseNotReady](#), [errIndexOutOfBounds](#), [leNMustBeGreaterThan1](#)). After a successful `SplitByN` the `IAxisVMLine` interface will become invalid (the original line gets deleted), it shouldn't be used for further interaction.

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 `leReferenceIndexOutOfBoundsException`.

EArchitectElemType	ArchitectElemType • Get or set architect element type
ELongBoolean	ColumnReinforcementParametersExists True if column reinf. parameters exists (read only property)
unsigned long	ContourColour • Get or set contour colour. If value is 0xFFFFFFFF then same as assigned material colour
long	ContourColour_vb • Visual Basic compatible property of ContourColour
long	EndNode • Get or set index of the end node according to IAxisVMNodes . For lines with a local axis, this is the node at the end of the local x axis.
ELineGeomType	GeomType Get line geometry type (straight line or arc)
ELongBoolean	IsBeam Get True if line is horizontal (read only property)
ELongBoolean	IsColumn Get True if line is vertical (read only property)
ELongBoolean	IsOtherType Get True if line is neither horizontal or vertical (read only property)
long	Length Get length of the line [m]
ELineType	LineType Get line element type
unsigned long	MaterialColour • Get or set material colour. If value is 0xFFFFFFFF then same as assigned material colour.
long	MaterialColour_vb • Visual Basic compatible property of MaterialColour
long	MemberId Get member index of the line
long	MidpointDOF • Get or set degrees of freedom for the line midpoint
long	MidpointId Get line midpoint index
ELineNonLinearity	NonLinearity • Get or set nonlinear behaviour (can be written only if LineType = ItTruss)
long	Reference • Get or set index of the reference (according to IAxisVMReferences). If an automatic reference is set, Reference = 0.
long	RigidBodyId Get index of the rigid bod. Returns 0 if it is not in any rigid body.
long	SectionCount [EAnalysisType AnalysisType] Get number of sections where results can be obtained
double	SectionPos [EAnalysisType AnalysisType , long SectionId] Get position of a given section measured from the origin of the local x axis
long	StartNode • Get or set index of the start node according to IAxisVMNodes . For lines with a local axis, this is the node at the start of the local x axis.
double	StiffnessReduction_A [long Index] • Get or set axial stiffness factor. Cross sections area is multiplied with this value in stiffness matrix. Default value is 1. Used only for atLinearStatic and atLinearVibration analysis types . (only valid for STAS and Eurocode [RO] standards)
double	StiffnessReduction_I [long Index] • Get or set flexural stiffness factor. Cross sections inertia is multiplied with this value in stiffness matrix. Default value is 1. Used only for atLinearStatic and atLinearVibration analysis types . (only valid for STAS and Eurocode [RO] standards)
long	StoreyId Get storey index of the line
double	TrussResistance • Get or set (only if LineType = ItTruss) resistance of the truss [kN] (absolute value) only for InTensionOnly and InCompressionOnly types of truss

double **Timber_kdef** • Get or set kdef value (deformation factor) of timber member
long **Timber_ServiceClass** • Get or set service class of timber member
double **Volume** Get volume of the line
double **Weight** Get weight of the line

IAxisVMLineSupports

Line supports of the model.

Records / structures

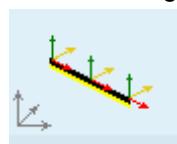
<u>ELineNonlinearity</u>	RNonlinearity = (
	x, y, z, <i>nonlinear behaviour in x, y, z direction</i>
	xx, yy, zz <i>and around the x, y, z axis</i>
)
	RNonlinearityXYZ = (
	x, y, z <i>nonlinear behaviour in x, y, z direction</i>
)
	RResistances = (
double	x, y, z, <i>resistances in x, y, z direction [kN/m]</i>
	xx, yy, zz <i>and around the x, y, z axis [kNm/m]</i>
)
	RResistancesXYZ = (
double	x, y, z <i>resistances in x, y, z direction [kN/m]</i>
)
	RStiffnesses = (
double	x, y, z <i>stiffnesses in x, y, z direction [kN/m/m]</i>
double	xx, yy, zz <i>rotational stiffnesses around the x, y, z axis [kN/rad/m]</i>
)
	RStiffnessesXYZ = (
double	x, y, z <i>stiffnesses in x, y, z direction [kN/m/m]</i>
)
NOTE:	<i>Direction of nonlinearity, stiffness and resistance depends on line support type <u>ELineSupportType</u></i>

Error codes

```
enum ELineSupportsError = {
    EseSectionIdOutOfBounds = -100001
    EsePadFootingNotDefined = -100002
}
```

index of the section is out of range
Line support has no pad footing defined

Functions

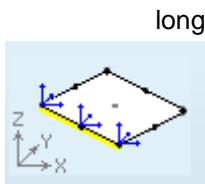


AddBeamElasticFoundation ([in] long BeamId,
[i/o] RStiffnessesXYZ StiffnessesXYZ, [i/o] RNonLinearityXYZ NonlinearityXYZ,
[i/o] RResistancesXYZ ResistancesXYZ)

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

StiffnessesXYZ	<i>stiffnesses of the elastic foundation in local direction</i>
NonlinearityXYZ	<i>nonlinear behaviour of the elastic foundation in local direction</i>
ResistancesXYZ	<i>resistances for stiffness components in local direction</i>

Adds an elastic foundation to a beam in beam's coordination system. If successful, returns the support index, otherwise returns an error code ([lelInvalidLineType](#), if the line is not a beam or [errDatabaseNotReady](#), [errIndexOutOfBoundsException](#)). The length of the beam should be smaller than $0.5 * \min((4^*Ex^*Iz/stiffness.y)^{0.25}, (4^*Ex^*Iy/stiffness.z)^{0.25})$



AddEdgeGlobal ([*i/o*] [RStiffnesses](#) Stiffnesses, [*i/o*] [RNonLinearity](#) Nonlinearity,
[*i/o*] [RResistances](#) Resistances, [*in*] long Edgeld, [*in*] long Surfaceld1,
[*in*] long Surfaceld2, [*in*] long DomainId1, [*in*] long DomainId2)

Stiffnesses stiffness components in the global system
Nonlinearity nonlinear behaviour of the support

Nonlinearity Resistances

Edgeld index of the edge

Edgeld Index of the edge
 $(0 < Edgeld \leq \text{AxisVMLines.Count})$

SurfaceId1 *first connecting surface*

(0 < SurfaceId1 ≤ AxisVMSurfaces.Count)

SurfaceId2 second connecting surface
(0 < SurfaceId2 < AxisVMSurfaces_Count)

DomainId1 $(0 < SurfaceIdz \leq \text{AxisVMSum})$
first connecting domain

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

DomainId2 long	<p><i>second connecting domain (0 < DomainId2 ≤ AxisVMDomains.Count)</i></p> <p>Adds an edge support in the 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 an error code (errDatabaseNotReady, errIndexOutOfBoundsException).</p>
 long	<p>AddEdgeRelative ([i/o] RStiffnesses Stiffnesses, [i/o] RNonLinearity Nonlinearity, [i/o] RResistances Resistances, [in] long Edgeld, [in] long Surfaceld1, [in] long Surfaceld2, [in] long DomainId1, [in] long DomainId2)</p> <p>Stiffnesses stiffness components in the local system of the edge</p> <p>Nonlinearity nonlinear behaviour of the support</p> <p>Resistances resistances for stiffness components</p> <p>Edgeld index of the edge (0 < Edgeld ≤ AxisVMLines.Count)</p> <p>Surfaceld1 first connecting surface (0 < Surfaceld1 ≤ AxisVMSurfaces.Count)</p> <p>Surfaceld2 second connecting surface (0 < Surfaceld2 ≤ AxisVMSurfaces.Count)</p> <p>DomainId1 first connecting domain (0 < DomainId1 ≤ AxisVMDomains.Count)</p> <p>DomainId2 second connecting domain (0 < DomainId2 ≤ AxisVMDomains.Count)</p>
 long	<p>Adds an edge support in the edge's coordination 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 indexes 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 indexes 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 an error code (errDatabaseNotReady, errIndexOutOfBoundsException).</p>
 long	<p>AddRibElasticFoundation ([in] long RibId, [i/o] RStiffnessesXYZ StiffnessesXYZ, [i/o] RNonLinearityXYZ NonlinearityXYZ, [i/o] RResistancesXYZ ResistancesXYZ)</p> <p>RibId index of the rib (0 < RibId ≤ AxisVMLines.Count)</p> <p>StiffnessesXYZ stiffnesses of the elastic foundation in local direction</p> <p>NonlinearityXYZ nonlinear behaviour of the elastic foundation in local direction</p> <p>ResistancesXYZ resistances for stiffness components in local direction</p>
 long	<p>Adds an elastic foundation to a rib in the rib's coordination system. If successful, returns the support index, otherwise returns an error code (leInvalidLineType, if the line is not a beam or errDatabaseNotReady, errIndexOutOfBoundsException). The length of the rib should be smaller than $0.5 * \min((4*Ex*Iz/stiffnes.y)^{0.25}, (4*Ex*Iy/stiffnes.z)^{0.25})$</p>
 long	<p>Delete ([in] long Index)</p> <p>Index support index</p> <p>Deletes the line support indexed by Index. If successful, returns the line support index otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBoundsException).</p>
 long	<p>DeleteSelected</p> <p>Deletes the selected line supports. If successful, returns number of deleted supports, otherwise returns an error code (errDatabaseNotReady)</p>
 long	<p>GetSelectedItemIds ([out] SAFEARRAY (long) * ItemIds)</p> <p>ItemIds list of selected line supports</p> <p>If successful, returns the number of selected elements otherwise returns an error code</p>
 long	<p>GetNodeIds ([in] long Index, [out] long StartNodeID, [out] long EndNodeID)</p>

Index *line support index*
StartNodeID *index of starting node*
EndNodeID *index of ending node*
If successful, returns the *line support index*, otherwise returns an error code
([errIndexOutOfBounds](#))

long **GetTrMatrix** ([in] long **Index**, [i/o] [RMatrix3x3 Value](#))

Index *line support index*

Value *Transformation matrix of the line support*

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

Properties

long **Count** *Get number of line supports in the model*

[EBoolean](#) **HaveStiffnessCalcParam**[long **Index**] *Returns lbTrue if support has defined parameters for calculating stiffness*

[AxisVMLineSupport](#)* **Item** [long **Index**] *Get line support interface by index*

long **LineID** [long **Index**] *Get line index of the support by index*

long **SectionCount** [long **Index**, [EAnalysisType AnalysisType](#)] *Get number of sections where support force results can be obtained*

[EBoolean](#) **Selected** [long **Index**] • *Get or set the selection status of a line support*

NOTE: Call [Refresh](#) function afterwards if not called between functions [BeginUpdate](#) and [EndUpdate](#)

long **SelCount** *Get number of selected line supports*

IAxisVMLineSupport

An AxisVM line support interface.

Enumerated types

```
enum ELineSupportType = {  
    IstEdgeGlobal = 0x0,  
    IstEdgeRelative = 0x1,  
    IstRibElasticFoundation = 0x2,  
    IstBeamElasticFoundation = 0x3,  
    IstEdgeReference = 0x4 }  
    Line support type
```

NOTE: Direction of nonlinearity, stiffness and resistance depends on line support type [ELineSupportType](#)

Records / structures

RElementStiffnessParams RWallStiffnessParams RWallStiffnessParams	<pre>double RWallStiffnessParams = (CalcParams common calculation parameters WallThickness thickness of the supporting element (wall)) RLineSupportStiffParams = (Top calculation parameters of the supporting element (wall) above support Bottom calculation parameters of the supporting element (wall) below support)</pre>
---	---

Functions

long	GetFootingDimensions ([i/o] RPadFootingDimensions Value)
	Warning! This function has become obsolete, was superseded by GetFootingParams_V153
	Get the calculated dimensions of the pad footing. If successful, returns the index of the line support, otherwise returns an error code (errDatabaseNotReady).
long	GetFootingParams ([i/o] RPadFootingParams Params)
	Warning! This function has become obsolete, was superseded by GetFootingParams_V153
	Get the pad footing design calculation parameters of the support. If successful, returns the index of the line support, otherwise returns an error code(errDatabaseNotReady).
long	GetFootingParams_V153 ([i/o] RLinearFootingParams Params)
	Get the specified and calculated footing parameters of the support. If successful, returns the index of the line support, otherwise returns an error code(errDatabaseNotReady).
long	GetNonLinearity ([i/o] RNonLinearity Value)
	Get the nonlinear behaviour of the line support (only if <i>SupportType</i> = <i>IstEdgeGlobal</i> or <i>IstEdgeRelative</i>). If successful, returns the index of the line support, otherwise returns an error code(errIndexOutOfBounds , errDatabaseNotReady).
long	GetNonLinearityXYZ ([i/o] RNonlinearityXYZ Value)
	Get the nonlinear behaviour of the elastic foundation (only if <i>SupportType</i> = <i>IstBeamElasticFoundation</i> or <i>IstRibElasticFoundation</i>). If successful, returns the index of the line support, otherwise returns an error code(errIndexOutOfBounds , errDatabaseNotReady).
long	GetResistance ([i/o] RResistances Value)
	Get the resistance components of the line support (only if <i>SupportType</i> = <i>IstEdgeGlobal</i> or <i>IstEdgeRelative</i>). If successful, returns the index of the line

support, otherwise returns an error code ([errIndexOutOfBounds](#), [errDatabaseNotReady](#)).

long	GetResistanceXYZ ([i/o] RResistancesXYZ Value) <i>Get the resistance components of the line support (only if SupportType = IstBeamElasticFoundation or IstRibElasticFoundation). If successful, returns the index of the line support, otherwise returns an error code (errIndexOutOfBounds, errDatabaseNotReady).</i>
long	GetStiffnesses ([i/o] RStiffnesses Value) <i>Get the stiffness components of the line support (only if SupportType = IstEdgeGlobal or IstEdgeRelative). If successful, returns the index of the line support, otherwise returns an error code (errIndexOutOfBounds, errDatabaseNotReady).</i>
long	GetStiffnessesXYZ ([i/o] RStiffnessesXYZ Value) <i>Get the stiffness components of the line support (only if SupportType = IstBeamElasticFoundation or IstRibElasticFoundation). If successful, returns the index of the line support, otherwise returns an error code (errIndexOutOfBounds, errDatabaseNotReady).</i>
long	GetStiffnessCalcParams ([i/o] RLineSupportStiffParams Value) <i>Get calculation parameters for calculating the stiffness of the line support. If unsuccessful, returns an error code(errIndexOutOfBounds, errDatabaseNotReady).</i>
long	SetNonLinearityXYZ ([i/o] RNonlinearityXYZ Value) <i>Set the nonlinear behaviour of the elastic foundation (only if SupportType = IstBeamElasticFoundation or IstRibElasticFoundation). If successful, returns the index of the line support, otherwise returns an error code (errIndexOutOfBounds, errDatabaseNotReady).</i>
long	SetResistance ([i/o] RResistances Value) <i>Set the resistance components of the line support (only if SupportType = IstEdgeGlobal or IstEdgeRelative). If successful, returns the index of the line support, otherwise returns an error code (errIndexOutOfBounds, errDatabaseNotReady).</i>
long	SetResistanceXYZ ([i/o] RResistancesXYZ Value) <i>Set the resistance components of the line support (only if SupportType = IstBeamElasticFoundation or IstRibElasticFoundation). If successful, returns the index of the line support, otherwise returns an error code (errIndexOutOfBounds, errDatabaseNotReady).</i>
long	SetStiffnesses ([i/o] RStiffnesses Value) <i>Set the stiffness components of the line support (only if SupportType = IstEdgeGlobal or IstEdgeRelative). If successful, returns the index of the line support, otherwise returns an error code (errIndexOutOfBounds, errDatabaseNotReady).</i>
long	SetStiffnessesXYZ ([i/o] RStiffnessesXYZ Value) <i>Set the stiffness components of the line support (only if SupportType = IstBeamElasticFoundation or IstRibElasticFoundation). If successful, returns the index of the line support, otherwise returns an error code (errIndexOutOfBounds, errDatabaseNotReady).</i>
long	SetStiffnessCalcParams ([i/o] RLineSupportStiffParams Value) <i>Set calculation parameters for calculating the stiffness of the line support. If unsuccessful, returns an error code(errIndexOutOfBounds, errDatabaseNotReady).</i>

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 (<i>SupportType = IstEdgeGlobal or IstEdgeRelative</i>) first domain connecting to the edge
long	DomainId2 (<i>SupportType = IstEdgeGlobal or IstEdgeRelative</i>) second domain connecting to the edge
long	Edgeld (<i>SupportType = IstEdgeGlobal or IstEdgeRelative</i>) edge index according to IAxisVMLines
EPadFootingType	FootingType Type of pad footing if it has been defined in AxisVM
EBoolean	HasFooting <i>lbTrue</i> if footing have been defined for the support in AxisVM
long	LineId line index according to IAxisVMLines
long	SectionCount [EAnalysisType AnalysisType] number of sections where support force results can be obtained
double	SectionPos [EAnalysisType AnalysisType , long SectionId] position of a given cross-section along the line element ($0 < \text{SectionId} \leq \text{SectionCount}[\text{AnalysisType}]$)
ELineSupportType	SupportType line support type
long	Surfaceld1 (<i>SupportType = IstEdgeGlobal or IstEdgeRelative</i>) first surface connecting to the edge
long	Surfaceld2 (<i>SupportType = IstEdgeGlobal or IstEdgeRelative</i>) first surface connecting to the edge

IAxisVMLinkElements

Link elements of the model.

Enumerated types

```
enum ELinkElementType = {
    letNN = 0x0,    Node to node link
    letLL = 0x1}    line to line link
    Interface types.
```

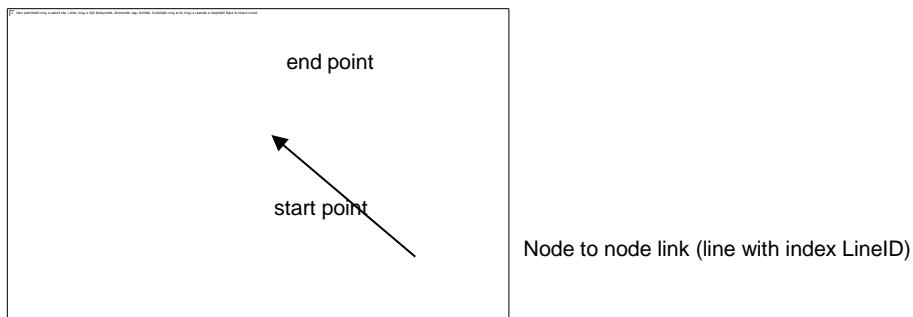
Error codes

enum ELinkElementError: = {	
leeLineIndexOutOfBounds = -100001	line index invalid
leeInvalidSystemType = -100002	Invalid SystemGLR during setting the
leeReferenceIndexOutOfBounds = -100003	reference index invalid
leeErrorAddingNN = -100004	error during NN interface creation
leeErrorAddingLL = -100005	error during LL interface creation
leeInvalidLinkElementType = -100006	unknown interface type
leeNotConnectingMasterLineAndMasterStartLink = -	master/slave line and master start/end link not
100007	connected
leeNotConnectingMasterLineAndMasterEndLink = -	master/slave line and master start/end link not
100008	connected
leeNotConnectingSlaveLineAndMasterStartLink = -	master/slave line and master start/end link not
100009	connected
leeNotConnectingSlaveLineAndMasterEndLink = -	master/slave line and master start/end link not
100010}	connected

Records / structures

Node to Node link

Example: A main girder-purlin connection. The main girder is an IPE-400 in X-Z plane, the purlin is an I-200 (Pict. 1). These elements are represented by their line of gravity. The link has to be placed between these two axes at their point of intersection (if seen from above). Therefore, this link has to be assigned to a vertical line having a length equal to the distance of axes i.e. 30 cm (40/2 + 20/2). The interface always has to be placed at the actual point of contact. In this case the interface is located 20 cm far (40/2) from the master node (i.e. the main girder axis). So the interface position is 20/30 = 0.666 (relative value) from master point (girder) or 0.2m (absolute value).

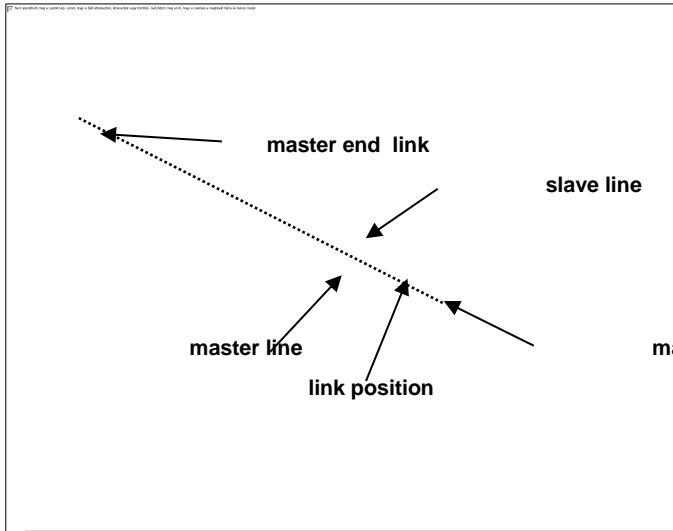


Pict. 1 - Node to node link

	RNNLinkElementRec = (
long	LineID	Index of the line where NN link will be created
<u>ESystem</u>	SystemGLR	coordinate system (local or global)
long	MasterPoint	1=start point or 2 = end point of the line with index LineID
long	RefZId	In case of local coordinate system the referenceID (if 0 then automatic)
<u>EBeamRibDistributionType</u>	PositionType	brdtProjected for relative position and brdtLength for absolute position of interface (in metres).
double	Position	distance of interface from master line (see 4.9.17 Node-to-Node (N-N) Link in Axis VM Manual)
<u>RStiffnesses</u>	Stiffnesses	stiffness of the NN link
<u>RResistances</u>	Resistances	resistance of the NN link
<u>RNonLinearity</u>	NonLinearity	nonlinearity of the NN link
)	

Line to Line link

Example: A wall to slab connection. The master line is edge of 0.4m THK slab in the centre plane. The slave line is top of 0.3m THK wall in the centre plane. The link has to be placed between these two lines. Therefore, this link has a distance equal to the distance of lines i.e. 0.2m (0.4/2). The interface always has to be placed at the actual line of contact. In this case the interface is located 0.2m far from the master line (i.e. the edge of slab). So the interface position is 0.2/0.2 = 1 (relative value) from master line (girder) or 0.2m (absolute value).



Pict. 2 - Line to line link

```

RLLLLinkElementRec = (
    long MasterLine Line index of the master line
    long SlaveLine Line index of the slave line
    long MasterStartLink Line index of the master start link
    long MasterEndLink Line index of the master end link
    PositionType brdtProjected for relative position and brdtLength for absolute position of interface (in metres).
    double Position distance of interface from master line (see 4.9.17 Line-to-Line (L-L) Link in Axis VM Manual)
    RStiffnesses stiffness of the LL link
    RResistances resistance of the LL link
    RNonLinearity nonlinearity of the LL link
    )

RLinkElementRec = (
RNNLinkElementRec;
RLLLLinkElementRec;
)
IAxisVMLinkElements.GetRec ans IAxisVMLinkElements.SetRec are using this record. GetRec is using the corresponding type record of record. When calling SetRec , corresponding type of record of record has to be used.
```

Functions

long **AddNN ([i/o] RNNLinkElementRec* NNLinkElementRec)**

NNLinkElementRec *NN link element parameters*

*Add node to node link. If successful returns LinkID, otherwise an error code
[\(errDatabaseNotReady, leeLineIndexOutOfBounds, leeInvalidSystemType, leeReferenceIndexOutOfBounds, leeErrorAddingNN\)](#)*

long **AddLL ([i/o] RLLLLinkElementRec* LLLLinkElementRec)**

LLLLinkElementRec *LL link element parameters*

*Add line to line link. If successful returns LinkID, otherwise an error code
[\(errDatabaseNotReady, leeLineIndexOutOfBounds, leeErrorAddingLL, leeNotConnectingMasterLineAndMasterStartLink, leeNotConnectingMasterLineAndMasterEndLink, leeNotConnectingSlaveLineAndMasterStartLink, leeNotConnectingSlaveLineAndMasterEndLink\)](#)*

long	Clear
	<i>Returns number of link elements before delete, otherwise an error code (errDatabaseNotReady).</i>
long	Delete ([in] long Index)
	<p>Index index of the link element, $1 \leq \text{Index} \leq \text{Count}$</p> <p>If successful returns index, otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBounds)</p>
long	DeleteSelected
	<i>Returns number of deleted link elements, otherwise an error code (errDatabaseNotReady).</i>
long	GetRec ([in] long Index, [i/o] RLinkElementRec* LinkElementRec)
	<p>Index link element index, $1 \leq \text{Index} \leq \text{Count}$</p> <p>LinkElementRec link element record</p> <p>If successful returns LinkID, otherwise an error code (errDatabaseNotReady, leeLineIndexOutOfBounds, leeInvalidLinkElementType)</p>
long	GetSelectedItemIds ([out] SAFEARRAY(long) * ItemIds)
	<p>ItemIds Index list of selected link elements</p> <p>Returns the number of selected link elements</p>
long	SelectAll ([in] EBoolean Select)
	<p>Select selection state</p> <p>If Select is True, selects all link elements.</p> <p>If Select is False, deselects all link elements.</p> <p>If successful, returns the number of selected link elements, otherwise returns an error code (errDatabaseNotReady)</p> <p><i>NOTE: Call Refresh function afterwards if not called between functions BeginUpdate and EndUpdate</i></p>
long	SetRec ([in] long Index, [i/o] RLinkElementRec* LinkElementRec)
	<p>Index link element index, $1 \leq \text{Index} \leq \text{Count}$</p> <p>LinkElementRec link element record</p> <p>If successful returns LinkID, otherwise an error code (errDatabaseNotReady, leeLineIndexOutOfBounds, leeInvalidLinkElementType, leeReferenceIndexOutOfBounds, leeInvalidLinkElementType, leeNotConnectingMasterLineAndMasterStartLink, leeNotConnectingMasterLineAndMasterEndLink, leeNotConnectingSlaveLineAndMasterStartLink, leeNotConnectingSlaveLineAndMasterStartLink)</p>

Properties

long **Count**

Get number of link elements

[ELinkElementType](#) **LinkElementType** [long **Index**]

Index *index of the link element, $1 \leq Index \leq Count$*

Get element type, otherwise returns an error code ([errDatabaseNotReady](#), [errIndexOutOfBounds](#), [leelInvalidLinkElementType](#))

[ELongBoolean](#) **Selected** [long **Index**]

Index *index of the link element, $1 \leq Index \leq Count$*

Get or set the selection status of the link element

NOTE: Call [Refresh](#) function afterwards if not called between functions [BeginUpdate](#) and [EndUpdate](#)

long **SelCount**

Get number of selected link elements in the model

IAxisVMLoadCases

Load cases in the model. Also you can set and get parameters for seismic, pushover, imperfections, wind and snow loads in this interface.

Note: Model has to have at least one load case (ST1).

Enumerated types

enum	EBaseHeightType = {	
	bhtLowest = 0x0,	<i>Lowest node of model</i>
	bhtCustom = 0x1 }	<i>Use BaseHeight set by user</i>
	<i>How to determinate base height</i>	
enum	ELoadCaseType = {	
	IctStandard = 0x0,	<i>standard loads</i>
	IctInfluenceLine = 0x1,	<i>influence line loads</i>
	IctSeismic = 0x2,	<i>seismic loads</i>
	IctVibration = 0x3,	<i>vibration loads</i>
	IctPreStress = 0x4,	<i>prestress loads</i>
	IctMoving = 0x5,	<i>moving loads</i>
	IctDynamic = 0x6,	<i>dynamic loads</i>
	IctPushOver = 0x7,	<i>push over loads, output only</i>
	IctImperfection = 0x8 ,	<i>imperfection loads, output only</i>
	IctSnow = 0x9 ,	<i>snow loads, output only</i>
	IctSnowExcept = 0xA ,	<i>exceptional snow loads, output only</i>
	IctWind = 0xB,	<i>wind loads, output only</i>
	IctManualSeismic = 0xC ,	<i>Manual seismic loads</i>
	IctManualPreStress = 0xD	<i>Manual prestress loads</i>
	IctFire = 0xE }	<i>fire load case/fire effects</i>
	<i>Load case type.</i>	
enum	ELoadDurationClass = {	
	ldcOther = 0x0,	<i>Other</i>
	ldcPermanent = 0x1,	<i>Permanent</i>
	ldcLong = 0x2,	<i>Long</i>
	ldcMedium = 0x3,	<i>Medium</i>
	ldcShort = 0x4,	<i>Short</i>
	ldcInstant = 0x5 }	<i>Instant</i>
	<i>Load duration class</i>	
enum	EModalCombType = {	
	mctAuto = 0x0,	<i>automatic</i>
	mctSRSS = 0x1,	<i>Square Root of Sum of Squares</i>
	mctCQC = 0x2 }	<i>Complete Quadratic Combination</i>
	<i>Combination type for modal responses in one direction</i>	
enum	ESeismicCombType = {	
	sctQuadratic = 0x0,	<i>quadratic mean</i>
	sctMax = 0x1,	<i>combination with 30%</i>
	sctAuto = 0x2 }	<i>automatic</i>
	<i>Combination type for spatial components.</i>	
enum	EVibrationType = {	
	vtFirstOrder = 0x0,	<i>first order vibration</i>
	vtSecondOrder = 0x1 }	<i>second order vibration</i>
	<i>Vibration result type.</i>	

enum	ESwayDirection = {	
	sdPlusX = 0x0,	<i>Sway in positive x direction</i>
	sdMinX = 0x1,	<i>Sway in negative x direction</i>
	sdPlusY = 0x2,	<i>Sway in positive y direction</i>
	sdMinY = 0x3,	<i>Sway in negative y direction</i>
	sdCustom = 0x4,	<i>Sway in custom direction</i>
	<i>Sway direction</i>	
enum	ETerrainCategory = {	
	tc0 = 0x0,	<i>Terrain category 0</i>
	tcl = 0x1,	<i>Terrain category I</i>
	tcII = 0x2,	<i>Terrain category II</i>
	tcIII = 0x3,	<i>Terrain category III</i>
	tcIV = 0x4	<i>Terrain category IV</i>
	}	
	<i>Terrain category</i>	
enum	ERoofType = {	
	rtUndefined = 0x0,	<i>Undefined roof</i>
	rtFlat = 0x1,	<i>Flat roof</i>
	rtMonopitch = 0x2,	<i>Monopitch roof</i>
	rtDuopitch = 0x3,	<i>Duopitch roof</i>
	rtHip = 0x4,	<i>Hip roof</i>
	rtBarrel = 0x5	<i>Barrel roof</i>
	}	
	<i>Roof type</i>	
enum	EFlatRoofEdgeType = {	
	retNone = 0x0,	<i>Nothing on the edge</i>
	retSharpEaves = 0x1,	<i>Edge with sharp eaves</i>
	retWithParapetWall = 0x2,	<i>Edge with parapet wall</i>
	retRoundEaves = 0x3,	<i>Edge with round eaves</i>
	retMansardEaves = 0x4,	<i>Edge with mansard eaves</i>
	}	
	<i>Edge type of flat roof</i>	
enum	ESeismicLimitState = {	
	selsOther = 0,	<i>other</i>
	selsOperational = 1,	<i>operational limit state</i>
	selsDamage = 2,	<i>damage limit state</i>
	selsLifeSafety = 3,	<i>life-safety limit state</i>
	selsCollapse = 4,	<i>collapse limit state</i>
	}	
	<i>seismic limit state for NationalDesignCode=ndcItalian</i>	

Error codes

enum	ELoadCasesError = {	
	IcaePropertyNotValidForThisType = -100001	<i>property is not compatible with the load case type</i>
	IcaeNameExists = -100002	<i>load case with the same name already exists</i>
	IcaeErrorCreatingStandardSeismicCases = -100003	<i>Error creating standard seismic cases</i>
	IcaeGroupIdOutOfBounds = -100004	<i>loadcase group index out of bounds</i>
	IcaeErrorCreatingPushOverCases = -100005	<i>Error creating standard pushover cases</i>
	IcaeInvalidAnalysisTypeDirX = -100006	<i>Invalis analysis type for loadcase in x direction</i>

IcaeLoadCaseIndexOutOfBoundsDirX = -100007	<i>index out of bounds for loadcase in x direction</i>
IcaeVibrationModeIndexOutOfBoundsDirX = -100008	<i>Vibration mode index out of bounds for loadcase in x direction</i>
IcaeInvalidAnalysisTypeDirY = -100009	<i>Invalis analysis type for loadcase in y direction</i>
IcaeLoadCaseIndexOutOfBoundsDirY = -100010	<i>index out of bounds for loadcase in y direction</i>
IcaeVibrationModeIndexOutOfBoundsDirY = -100011	<i>Vibration mode index out of bounds for loadcase in y direction</i>
IcaeErrorCreatingPreStressCases = -100012	<i>Prestress case can not be created</i>
IcaeNoPushOverCases = -100013	<i>Push over cases don't exist or push over not supported by national design code</i>
IcaeInvalidLoadCaseType = -100014	<i>Load case type is not valid, use dedicated Create.. functions</i>
IcaeNoModeShapesForLoadCaseInDirectionX = -100015	<i>Calculate mode shapes for the selected loadcase or loadcombination in x direction</i>
IcaeNoModeShapesForLoadCaseInDirectionY = -100016	<i>Calculate mode shapes for the selected loadcase or loadcombination in y direction</i>
IcaeNoModeShapesInDirectionX = -100017	<i>Calculate mode shapes for the selected loadcase or loadcombination in x direction</i>
IcaeNoModeShapesInDirectionY = -100018	<i>Calculate mode shapes for the selected loadcase or loadcombination in y direction</i>
IcaeInvalidLoadGroupType = -100019	<i>Load case type is not valid, use dedicated functions: e.g.: Create...</i>
IcaeSWGmoduleNotAvailable = -100020	<i>Extension module SWG is not available</i>
IcaeNoSnowLoadCases = -100021	<i>Create snow load cases before calling function returning this error</i>
IcaeNoWindLoadCases = -100022	<i>Create wind load cases before calling function returning this error</i>
IcaeInvalidName = -100023	<i>Name of the load case is invalid</i>
IcaeNoSeismicParams = -100024	<i>Seismic parameters are not available</i>
IcaeSE1moduleNotAvailable = -100025	<i>Extension module SE1 is not available</i>
IcaeErrorSettingSeismicParams = -100026	<i>Seismic parameters are invalid</i>
IcaeLoadCaseLoadCombinationNotFound = -100027	<i>Load case or load combination index is invalid</i>
IcaeSeismicInvalidGroupId = -100028	<i>No seismic group with the specified ID</i>
}	

Records / structures

RImperfectionParams = (

ESwayDirection	SwayDirection	<i>sway direction</i>
double	SwayAngle	<i>sway angle alpha, see Imperfections in AxisVM manual</i>
EBaseHeightType	BaseHeightType	<i>base height type</i>
double	BaseHeight	<i>base height, see Imperfections in AxisVM manual [m]</i>
ELongBoolean	StructureAutoHeight	<i>calculated by AxisVM, see Imperfections in AxisVM manual</i>
double	StructureHeight	<i>structure height, see Imperfections in AxisVM manual [m]</i>
long	ColumnsInvolved	<i>number of involved columns, see Imperfections in AxisVM manual</i>
double	Alpha_h	<i>α_h, see Imperfections in AxisVM manual</i>
double	Alpha_m	<i>α_m, see Imperfections in AxisVM manual</i>
double	Phi0)	<i>ϕ_0, see Imperfections in AxisVM manual</i>

RPushOverDirectionParams = (

ELongBoolean	Uniform	<i>If true, then uniform load distribution</i>
ELongBoolean	Modal	<i>If true, then modal load distribution</i>
EAnalysisType	VibrationAnalysisType	<i>Analysis type, only atLinearVibration or atNonLinearVibration are valid</i>
long	VibrationLoadCase	<i>Index of the load case or load combination used in vibration analysis IMPORTANT NOTE: If it is the index of the load combination then: Load combination index = VibrationLoadCase - IAxisVMloadcases.count</i>
long	VibrationMode	<i>Vibration mode; 0=< and <=ModeShapes, 0 if AutoDominantMode=lbTrue</i>
ELongBoolean	AutoDominantMode	<i>If true, then dominant vibration mode is selected and VibrationMode value is ignored</i>
long	AccidentalEcc)	<i>Accidental eccentricity</i>

RPushOverParams = (

RPushOverDirectionParams	X	<i>Pushover parameters in x direction</i>
RPushOverDirectionParams	Y	<i>Pushover parameters in y direction</i>
)	

	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) ψ 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
ELongBoolean	Torsion	(*) tells if torsion effects are taken into account
double	ExcCoeff	(*) eccentricity coefficient
double	C	(STAS) C
double	nu	(STAS) v
long	LoadCaseCombination	Compound load case which we used for mode shape calculations IMPORTANT NOTE: Compound load case means that for load combinations : LoadCaseCombination = Load combination index + IAxisVMLoadcases.Count
double	eta	damping correction factor (see EN1998-1) Note: Valid only for ndcEuroCode, ndcRomanian_STAS, ndcSwiss_SIA26x, ndcItalian, ndcEuroCode_Austrian, ndcEuroCode_UK ENationalDesignCode .
)		

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 other fields apply for all design codes.

	RSeismicParams_V153 = (
EVibrationType	VibrType	vibration result type
	if NationalDesignCode = ndcHungarian_MSZ	
double	kg	k_g seismic constant
double	ks	k_s importance factor of the building
double	kt	k_t soil factor
double	psi	ψ damping factor
	if NationalDesignCode is in ndcEuroCode (including any of its subcodes), ndcRomanian_STAS, ndcSwiss_SIA26x, ndcItalian	
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
ELongBoolean	Torsion	tells if torsion effects are taken into account
double	ExcCoeff	eccentricity coefficient
double	C	(STAS) C
double	nu	(STAS) v
long	LoadCaseCombination	Compound load case which we used for mode shape calculations IMPORTANT NOTE: Compound load case means that for load combinations : LoadCaseCombination = Load combination index + IAxisVMLoadcases.Count
double	eta	damping correction factor (see EN1998-1) Note: Valid only for ndcEuroCode, ndcRomanian_STAS, ndcSwiss_SIA26x, ndcItalian, ndcEuroCode_Austrian, ndcEuroCode_UK ENationalDesignCode .
long	GroupID	seismic group index (use SeismicGroupIds to query the available group indexes)
double	qdy	q_d behaviour factor for vertical displacements (if the national design code allows it)
ESeismicLimitState	SeismicLimitState	seismic limit state, valid only for NationalDesignCode=ndcItalian
)		

	RWindLoadParameters = (
double	A	Altitude above sea level
double	v_b0	Basic wind velocity
double	c_season	Season factor
double	c_o	Orography factor
ELongBoolean	TerrainCategoryDifferent	Terrain category different in directions
ETerrainCategory	TerrainCat_Xp	Terrain category +X (default if TerrainCategoryDifferent = lbFalse)
ETerrainCategory	TerrainCat_Xm	Terrain category -X
ETerrainCategory	TerrainCat_Yp	Terrain category +Y
ETerrainCategory	TerrainCat_Ym	Terrain category -Y
ELongBoolean	CustomDirectionalFactors	Custom directional factors different in directions
double	c_dir_xp	directional factors in -X direction
double	c_dir_xm	directional factors in +X direction

double	c_dir_yp	<i>directional factors in -y direction</i>
double	c_dir_ym	<i>directional factors in +Y direction</i>
ERoofType	RoofType	<i>Roof type</i>
EFlatRoofEdgeType	FlatRoofEdgeType	<i>Used only if RoofType = rtFlat</i>
double	FlatRoofEdgeParam	<i>parapet height(h_p), radius (r) or pitch(alfa) depends on FlatRoofEdgeType</i>
ELongBoolean	TorsionalEffect	<i>If lbTrue then additional load cases will be created for torsional winds</i>
double	u_xp	<i>μ factor related to the area of openings in +X direction</i>
double	u_xm	<i>μ factor related to the area of openings in -X direction</i>
double	u_yp	<i>μ factor related to the area of openings in +Y direction</i>
double	u_ym	<i>μ factor related to the area of openings in -Y direction</i>
double	lw	<i>Importance factor (ndcEuroCode_RO only)</i>
long	Zone	<i>Number of supported zones depended on national design code where first zone name is 1, second 2, etc..</i>
		<i>Used only in listed design codes:</i>
		<i>ndcEuroCode_PL: Strefa 1, Strefa 2, Strefa 3,</i>
		<i>ndcEuroCode_GER: Zone 1, Zone 2, Zone 3, Zone 4</i>
		<i>ndcItalian: Zona 1, Zona 2, Zona 3, Zona 4, Zona 5, Zona 6, Zona 7, Zona 8, Zona 9</i>
		<i>ndcEuroCode_RO: Zona 1, Zona 2, Zona 3, Zona 4, Zona 5</i>
		<i>ndcEuroCode_CZ: Zone 1, Zone 2, Zone 3, Zone 4, Zone 5</i>
		<i>ndcEuroCode_NL: Zone I, Zone II, Zone III</i>
		<i>ndcEuroCode_B: Zone 1, Zone 2, Zone 3, Zone 4</i>

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 an error code ([lcaeNameExists](#), [errDatabaseNotReady](#), [lcaeInvalidLoadCaseType](#)).

long **AddWithGroup ([in] BSTR Name, [in] ELoadCaseType LoadCaseType, [in] long GroupId)**

Name *Name of the loadcase*
LoadCaseType *Type of the loadcase*
GroupId *Group index*

Creates load case with assigned group. Returns load case index if successful, otherwise returns an error code ([errDatabaseNotReady](#), [lcaeGroupIdOutOfBounds](#), [lcaeNameExists](#), [lcaeInvalidLoadCaseType](#), [lcaeInvalidLoadGroupType](#)).

long **CreateImperfectionCase ([in] BSTR Name, [i/o] RImperfectionParams ImperfectionParams)**

Name *This string is used for generating the imperfection load case names*
ImperfectionParams *Imperfection parameters*
Creates imperfection load case and imperfection load group if not exists. Returns load case index if successful, otherwise returns an error code ([errDatabaseNotReady](#), [lcaeNameExists](#)).

long	CreatePushOverCases ([in] BSTR Name, [i/o] RPushOverParams PushOverParams)
	<p>Name <i>This string is used for generating the pushover load case names</i></p> <p>PushOverParams <i>Pushover parameters</i></p> <p><i>Creates ungrouped pushover load cases. Returns index of the first created pushover load case if successful, otherwise returns an error code (errDatabaseNotReady, lcaeErrorCreatingPushOverCases, lcaeNoModeShapesForLoadCaseInDirectionX, lcaeNoModeShapesForLoadCaseInDirectionY).</i></p>
	<p>Important note!</p> <p>To recalculate call the IAxisVMLoads.CreateStandardPushOverLoads function after loading a spectrum.</p>
long	CreatePreStressCases ([in] BSTR Name, [in] long LoadGroup)
	<p>Name <i>This string is used for generating the prestress load case names</i></p> <p>LoadGroup <i>index of the load group</i></p> <p><i>Creates prestress load cases. If successful, returns total number of load cases, otherwise returns an error code (errDatabaseNotReady, lcaeErrorCreatingPreStressCases, lcaeGroupIdOutOfBounds).</i></p>
long	CreateSnowCases ([in] BSTR Name, [i/o] RSnowLoadParams SnowLoadParams)
	<p>Name <i>This string is used for generating the snow load case names. For example 'SNOW' string will be used for creating loadcase names SNOW UD, SNOW DX+, etc.</i></p> <p>SnowLoadParams <i>Snow load parameters</i></p> <p><i>Creates snow load cases and snow load group depending on used design code. If successful, returns the index of the first snow load case, otherwise returns an error code (errDatabaseNotReady, lcaeInvalidName or lcaeSWGmoduleNotAvailable).</i></p>
	<p>Important note!</p> <p>To calculate the snow loads also call the IAxisVMLoads.CreateSnowLoadOnLoadPanels.</p>
long	CreateStandardSeismicCases ([in] BSTR Icname)
	<p>Icname <i>This string is used for generating the seismic load case names. For example 'EQ' string will be used for creating loadcase names EQ- and EQ+.</i></p> <p><i>Creates standard seismic load cases and seismic load group depending on used design code. If successful, returns the seismic group ID of the newly created seismic group, otherwise returns an error code (errDatabaseNotReady or lcaeErrorCreatingStandardSeismicCases).</i></p>
	<p>Important note!</p> <p>To calculate the seismic loads set the SeismicParameters with SetSeismicParams_V153 function, and optionally set the horizontal and/or vertical spectrum for it (if their default values are not appropriate). Then call IAxisVMLoads.CreateStandardSeismicLoads_V153. These functions should receive the seismic group ID that was just created.</p>
long	CreateWindCases ([in] BSTR Name, [i/o] RWindLoadParams WindLoadParams)
	<p>Name <i>This string is used for generating the wind load case names. For example 'WIND' string will be used for creating loadcase names WIND X P.O., etc.</i></p> <p>WindLoadParams <i>Wind load parameters</i></p> <p><i>Creates wind load cases and wind load group depending on used design code. If successful, returns the index of the first wind load case, otherwise returns an error code (errDatabaseNotReady, lcaeInvalidName or lcaeSWGmoduleNotAvailable).</i></p>
	<p>Important note!</p> <p>To calculate the snow loads also call the IAxisVMLoads.CreateWindLoadOnLoadPanels function.</p>
long	Delete ([in] long Index)
	<p>Index <i>index of the load case to delete ($0 < Index \leq Count$)</i></p> <p><i>Deletes a load case by index.</i></p> <p><i>If successful, returns Index, otherwise returns an error code (errIndexOutOfBounds, errDatabaseNotReady).</i></p>

long	DeleteAllLoadsFromLoadCase ([in] long Index)
	<p>Index index of the load case to delete ($0 < \text{Index} \leq \text{Count}$)</p> <p>Deletes all loads from the load case by index.</p> <p>If successful, returns Index, otherwise returns an error code (errIndexOutOfBounds, errDatabaseNotReady).</p>
long	GetImperfectionParams ([in] long Index , [i/o] RImperfectionParams ImperfectionParams)
	<p>Index index of the load case</p> <p>ImperfectionParams Imperfection parameters</p> <p>Creates imperfection load case. Returns index if successful or error code (errIndexOutOfBounds, errDatabaseNotReady, IcaeInvalidLoadCaseType).</p>
long	GetPushOverParams ([i/o] RPushOverParams PushOverParams)
	<p>PushOverParams Pushover parameters</p> <p>Get pushover parameters. Returns 1 if successful or error code (errDatabaseNotReady, IcaeNoPushOverCases, IcaeNoModeShapesInDirectionX, IcaeNoModeShapesInDirectionY).</p>
long	GetSeismicParams ([i/o] RSeismicParams SeismicParams)
	<p>Warning! This function has become obsolete, was superseded by GetSeismicParams_V153</p> <p>SeismicParams Seismic parameters according to the code</p> <p>Get the seismic parameters of the first seismic group according to the design code If successful it returns the index of the first seismic load case, otherwise it returns error (errIndexOutOfBounds, IcaeNoSeismicLoadCases, IcaeSE1moduleNotAvailable).</p>
long	GetSeismicParams_V153 ([in] long GroupID , [i/o] RSeismicParams_V153 SeismicParams)
	<p>GroupID Seismic group ID</p> <p>SeismicParams Seismic parameters according to the code</p> <p>Get the seismic parameters for the given GroupID according to the design code. If successful it returns the index of the first seismic load case from the group, otherwise it returns error (errIndexOutOfBounds, IcaeNoSeismicLoadCases, IcaeSE1moduleNotAvailable, IcaeSeismicInvalidGroupID).</p>
long	GetSnowLoadParams ([i/o] RSnowLoadParams SnowLoadParams)
	<p>SnowLoadParams Snow load parameters</p> <p>Get the snow load parameters according to the design code If successful it returns the index of the first snow load case, otherwise it returns error (errDatabaseNotReady, IcaeNoSnowLoadCases or IcaeSWGmoduleNotAvailable)</p>
long	GetWindLoadParams ([i/o] RWindLoadParams WindLoadParams)
	<p>WindLoadParams Wind load parameters</p> <p>Get the wind load parameters according to the design code If successful it returns the index of the first wind load case, otherwise it returns error (errDatabaseNotReady, IcaeNoWindLoadCases or IcaeSWGmoduleNotAvailable)</p>
long	SeismicGroupIDs ([out] SAFEARRAY(long)* GroupIds)
	<p>GroupIds list of all available seismic groups. Null if there is none.</p> <p>Queries the available seismic group indexes. GroupIds will be null if there is none. Return value is a positive number for success or error code (errDatabaseNotReady).</p>
long	SeismicSpectrumH ([in] long GroupID , [out] IAxisVMSpectrum Spectrum)
	<p>GroupID group ID of a seismic group</p> <p>Spectrum the returned horizontal spectrum</p> <p>Gets the horizontal spectrum for the given GroupID. If successful it returns a positive number, otherwise it returns error (errDatabaseNotReady, IcaeNoSeismicLoadCases, IcaeSE1moduleNotAvailable, IcaeSeismicInvalidGroupID).</p>

long	SeismicSpectrumV ([in] long GroupID , [out] IAxisVMSpectrum Spectrum)
	GroupID <i>group ID of a seismic group</i>
	Spectrum <i>the returned vertical spectrum</i>
	Gets the vertical spectrum for the given GroupID. You can check its <i>Disabled</i> property to see whether it is active or not. If successful it returns a positive number, otherwise it returns error (errDatabaseNotReady , IcaeNoSeismicLoadCases , IcaeSE1moduleNotAvailable , IcaeSeismicInvalidGroupID).
long	SetImperfectionParams ([in] long Index , [i/o] RImperfectionParams ImperfectionParams)
	Index <i>index of the load case</i>
	ImperfectionParams <i>Imperfection parameters</i>
	Creates imperfection load case. Returns index if successful or error code (errIndexOutOfBounds , errDatabaseNotReady , IcaeInvalidLoadCaseType).
long	SetSeismicParams ([i/o] RSeismicParams SeismicParams)
	Warning! This function has become obsolete, was superseded by SetSeismicParams_V153
	SeismicParams <i>Seismic parameters according to the code</i>
	Set the seismic parameters of the first seismic group according to the design code. If successful it returns the index of the first seismic load case, otherwise it returns error (errIndexOutOfBounds , IcaeNoSeismicLoadCases , IcaeSE1moduleNotAvailable , IcaeErrorSettingSeismicParams , IcaeLoadCaseLoadCombinationNotFound).
long	SetSeismicParams_V153 ([i/o] RSeismicParams_V153 SeismicParams)
	SeismicParams <i>Seismic parameters according to the code. GroupID field must be set to the intended seismic group ID.</i>
	Set the seismic parameters according to the design code. If successful it returns the index of the first seismic load case from the seismic group, otherwise it returns error (errIndexOutOfBounds , IcaeNoSeismicLoadCases , IcaeSE1moduleNotAvailable , IcaeErrorSettingSeismicParams , IcaeLoadCaseLoadCombinationNotFound , IcaeSeismicInvalidGroupID).
long	SetSnowLoadParams ([i/o] RSnowLoadParams SnowLoadParams)
	SnowLoadParams <i>Snow load parameters</i>
	Set the snow load parameters according to the design code. If successful it returns the index of the first snow load case, otherwise it returns error (errDatabaseNotReady , IcaeNoSnowLoadCases or IcaeSWGmoduleNotAvailable).
long	SetWindLoadParams ([i/o] RWindLoadParams WindLoadParams)
	WindLoadParams <i>Wind load parameters</i>
	Set the wind load parameters according to the design code. If successful it returns the index of the first wind load case, otherwise it returns error (errDatabaseNotReady , IcaeNoWindLoadCases or IcaeSWGmoduleNotAvailable).

Properties

long	Count Get number of load cases in the model. Negative number is an error code (errDatabaseNotReady).
long	GroupId [long Index] • Get or set load group index of a load case by index. See IAxisVMLoadGroups . Negative number is an error code (errIndexOutOfBounds , errDatabaseNotReady).
	Index <i>index of load case</i>
ELoadCaseType	LoadCaseType [long Index] Get load case type by index
long	LoadCount [long Index] Get number of loads in a load case by index. Negative number is an error code (errIndexOutOfBounds , errDatabaseNotReady).
	Index <i>index of load case</i>
ELoadDurationClass	LoadDurationClass [long Index] • Get or set load duration class. Negative number is an error code (errIndexOutOfBounds , errDatabaseNotReady).
	Index <i>index of load case</i>

long **IndexOfUID** [long **UID**] *Get index of the load case
UID unique index of the load case*

BSTR **Name** [long **Index**] • *Get or set load case name by index.
If an existing load case name is assigned an error event is created with the error code [lcaeNameExists](#).*

long **SesmicGroupID** [long **Index**] *Gets the seismic group ID for a seismic load case. If the load case is not a seismic load case, a negative value is returned and an error event is created with the error code [lcaeSeismicInvalidGroupID](#).*

long **UID** [long **Index**] *Get unique index of the load case which mains the same while exists in the model*

long **Index** *index of load case*

IAxisVMLoadCombinations

Load combinations of the model.

Note:

You can also generate load combinations from load cases if you have assigned load groups to the load cases.

Enumerated types

```
enum ECombinationType = {
    ctOther = 0,                                other combination
    ctSLS1 = 1,                                 Characteristic SLS combination (See 6.5.3 EN 1990)
    ctSLSChar = 1,                               same as ctSLS1
    ctSLS2 = 2,                                 Frequent SLS combination (See 6.5.3 EN 1990)
    ctSLSFreq = 2,                               same as ctSLS2
    ctSLS3 = 3,                                 Quasi-permanent SLS combination (See 6.5.3 EN 1990)
    ctSLSQuasi = 3,                            same as ctSLS3
    ctULS1 = 4,                                 Fundamental ULS combination (See 6.4.3.2 EN 1990)
    ctULS = 4,                                  same as ctULS1
    ctULS2 = 5,                                 Seismic ULS combination (See 6.4.3.4 EN 1990)
    ctULSSeismic = 5,                           same as ctULS2
    ctULS3 = 6,                                 Exceptional ULS combination (See 6.4.3.3 EN 1990)
    ctULSExceptional = 6,                      same as ctULS3
    ctULSALL = 7,                               Worst of these combination types : ctULS, ctULSSeismic,
                                                ctULSExceptional considered
    ctULSab = 8,                                Worst of a or b combination type considered , see EN
                                                1990:6.10(a,b)
    ctULSa = 9,                                 ULS combination type a , see EN 1990:6.10(a). Only as output value
    ctULSb = 10,                                ULS combination type b , see EN 1990:6.10(b). Only as output value
    ctULSALLab = 11,                           Worst of these combination types : ctULSab, ctULSSeismic,
                                                ctULSExceptional considered
    ctULSA1 = 12,                               ULS combination type A1
    ctULSA2 = 13,                               ULS combination type A2
    ctULSA3 = 14,                               ULS combination type A3
    ctULSA4 = 15,                               ULS combination type A4
    ctULSA5 = 16,                               ULS combination type A5
    ctULSA6 = 17,                               ULS combination type A6
    ctULSA7 = 18,                               ULS combination type A7
    ctULSA8 = 19,                               ULS combination type A8
}
```

Load combination type.

```
enum ECombinationTypeBits = {
    ctb_Other = 0x00000000,                    other combination
    ctb_SLS1 = 0x00000001,                     Characteristic SLS combination (See 6.5.3 EN 1990)
    ctb_SLSChar = 0x00000001,                  same as ctb_SLS1
    ctb_SLS2 = 0x00000002,                     Frequent SLS combination (See 6.5.3 EN 1990)
    ctb_SLSFreq = 0x00000002,                  same as ctb_SLS2
    ctb_SLS3 = 0x00000004,                     Quasi-permanent SLS combination (See 6.5.3 EN 1990)
    ctb_SLSQuasi = 0x00000004,                 same as ctb_SLS3
    ctb_ULS1 = 0x00000008,                     Fundamental ULS combination (See 6.4.3.2 EN 1990)
    ctb_ULS = 0x00000008,                     same as ctb_ULS1
    ctb_ULS2 = 0x00000010,                     Seismic ULS combination (See 6.4.3.4 EN 1990)
```

ctb_ULSSeismic = 0x00000010,	same as ctb_ULS2
ctb_ULS3 = 0x00000020,	<i>Accidental ULS combination (See 6.4.3.3 EN 1990)</i>
ctb_ULSExceptional = 0x00000020,	same as ctb_ULS3
ctb_ULSALL = 0x00000040,	see ctULSALL
ctb_ULSab = 0x00000080,	see ctULSab
ctb_USLa = 0x00000100,	<i>not used</i>
ctb_USLb = 0x00000200,	<i>not used</i>
ctb_ULSALLab = 0x00000400,	see ctULSALLab
ctb_ULSA1 = 0x00000800,	see ctULSA1
ctb_ULSA2 = 0x00001000,	see ctULSA2
ctb_ULSA3 = 0x00002000,	see ctULSA3
ctb_ULSA4 = 0x00004000,	see ctULSA4
ctb_ULSA5 = 0x00008000,	see ctULSA5
ctb_ULSA6 = 0x00010000,	see ctULSA6
ctb_ULSA7 = 0x00020000,	see ctULSA7
ctb_ULSA8 = 0x00040000 }	see ctULSA8

Load combination types used for generating different types of combinations

Error codes

enum	ELoadCombinationsError = {	
	IcoeDifferentFactorsAndIDsCount = -100001	<i>the number of load cases and factors are different</i>
	IcoeNameExists = -100002	<i>existing load combination name</i>
	IcoeAutoGenerationFailed = -100003	<i>Auto generation failed</i>
	IcoeAutoGenerationFailedNoCriticalGroup = -100004	<i>Load groups are not defined</i>
	IcoelInvalidCombinationTypesValue = -100005 }	<i>sum of combination types is zero or invalid</i>

Records / structures

RLoadCombinationGenParameters = (
ELongBoolean ConsiderImperfections ;	<i>consider imperfection load cases</i>
ELongBoolean OverwriteGeneratedCombos	<i>if true, delete previously generated load combinations</i>
ELongBoolean OverWriteDuplComboSameType	<i>used only if OverwriteGeneratedCombos is True. Delete previously generated load combinations with same ECombinationType</i>
)	

Functions

long	Add ([in] BSTR Name , [in] ECombinationType CombinationType , [in] SAFEARRAY(double)* Factors , [in] SAFEARRAY(long)* LoadCaseIds)	
	Name	<i>name of the load combination</i>
	CombinationType	<i>combination type</i>
	Factors	<i>combination factors</i>
	LoadCaseIds	<i>combination load case indexes according to IAxisVMLoadCases</i>

Adds a new load combination to the model.

*If successful, returns the index of the new combination, otherwise returns an error code (**IcoeDifferentFactorsAndIDsCount**, **IcoeNameExists**, **errDatabaseNotReady**).*

long	Add_vb (Visual Basic compatible function of Add)
------	---

long	Delete ([in] long Index)
	Index <i>index of the load combination to delete ($0 < \text{Index} \leq \text{Count}$)</i>
	<i>Deletes a load combination. If successful, returns Index, otherwise returns an error code (errIndexOutOfBounds, errDatabaseNotReady).</i>
long	GenerateAutoCombinations ([in] long CombinationTypes, [in] ELongBoolean ShowForms, [i/o] RLoadCombinationGenParameters LoadCombinationGenParameters)
	CombinationTypes <i>sum of ECombinationTypeBits values, except ctb_Other (e.g. $\text{ctb_SLS1+ctb_ULS1}=0x09$ therefore characteristic SLS and fundamental ULS combination will be generated)</i>
	ShowForms <i>Show window (with error messages) while generating combinations</i>
	LoadCombinationGenParameters <i>Parameters for generating load combinations</i>
	<i>Generates specified load combinations. If successful, returns number of generated combinations, otherwise returns an error code (IcoeAutoGenerationFailed, IcoeAutoGenerationFailedNoCriticalGroup, IcoelInvalidCombinationTypesValue, errDatabaseNotReady).</i>
long	GetCombination ([in] long Index, [out] SAFEARRAY(double)* Factors, [out] SAFEARRAY(long)* LoadCaselds)
	Index <i>index of the load combination ($0 < \text{Index} \leq \text{Count}$)</i>
	Factors <i>load combination factors</i>
	LoadCaselds <i>combination load case indexes according to IAxisVMLoadCases</i>
	<i>Retrieves a load combination. If successful, returns Index, otherwise returns an error code (errIndexOutOfBounds, errDatabaseNotReady).</i>
long	GetValidCombinationTypes ([out] SAFEARRAY(ECombinationType)* CombinationTypes)
	CombinationTypes <i>array of valid combination types</i>
	<i>Retrieves critical load combinations which can be used for reading results. If successful, returns 1, otherwise returns an error code (errDatabaseNotReady).</i>
long	SetCombination ([in] long Index, [in] SAFEARRAY(double)* Factors, [in] SAFEARRAY(long)* LoadCaselds)
	Index <i>index of the load combination ($0 < \text{Index} \leq \text{Count}$)</i>
	Factors <i>load combination factors</i>
	LoadCaselds <i>combination load case indexes according to IAxisVMLoadCases</i>
	<i>Modifies an existing load combination. If successful, returns Index, otherwise returns an error code (IeDifferentFactorsandIDsCount, errIndexOutOfBounds, errDatabaseNotReady).</i>
long	SetCombination_vb <i>(Visual Basic compatible function of SetCombination)</i>

Properties

ECombinationType	CombinationType [<long> Index] <i>Get type of a combination</i></long>
long	Count <i>Get number of load combinations in the model Negative number is an error code (errDatabaseNotReady).</i>
BSTR	Comment [<long> Index] • <i>Get or set comments for combination. Index index of load combination</i></long>

long	IndexOfUID [long UID] <i>Get index of the load combination UID unique index of the load combination</i>
BSTR	Name [long Index] • <i>Get or set name of a combination If an existing load combination name is assigned an error event is created with the error code LcoeNameExists.</i>
long	UID [long Index] <i>Get unique index of the load combination which remains the same while exists in the model Index index of load combination</i>

IAxisVMLoadGroups

Load groups in the model.

Enumerated types

enum	EGroupCombinationType = {	
	gctOld = 0x0,	<i>(not used)</i>
	gctExclusive = 0x1,	<i>(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.</i> <i>(for incidental load groups): only one load case of the group can be included into the critical combination.</i>
	gctAdditive = 0x2	<i>(for permanent and prestress load groups): all load cases will be taken into account simultaneously when determining the critical combination.</i> <i>(for incidental load groups): multiple load cases can be included into the critical combination.</i>

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

enum	ELoadGroupType = {	
	IgtPermanent = 0x0,	<i>permanent load group</i>
	IgtIncidental = 0x1,	<i>incidental load group</i>
	IgtExceptional = 0x2,	<i>exceptional load group</i>
	IgtSeismic = 0x3,	<i>seismic load group, output only</i>
	IgtPrestress = 0x4,	<i>prestress load group</i>
	IgtMoving = 0x5,	<i>moving load group</i>
	IgtImperfection = 0x6,	<i>imperfection load group, output only</i>
	IgtSnow = 0x7 ,	<i>snow load group, output only</i>
	IgtSnowExcept = 0x8 ,	<i>exceptional snow load group, output only</i>
	IgtWind = 0x9 ,	<i>wind load group, output only</i>
	IgtManualSeismic = 0xA ,	<i>manual seismic load group</i>
	IgtManualPreStress = 0xB,	<i>manual prestress load group</i>
	IgtFire = 0xC }	<i>fire load group</i>
		<i>Load group types.</i>

Error codes

enum	ELoadGroupsError = {	
	IgePropertyNotValidForThisType = -100001,	<i>property is not compatible with the load group type</i>
	IgeNameExists = -100002 ,	<i>group with the same name already exists</i>
	IgeInvalidType = -100003 ,	<i>use dedicated Create.. functions in IAxisVMLoadCases interface</i>
	}	

Functions

long	Add ([in] BSTR Name, [in] ELoadGroupType LoadGroupType, [in] ELongBoolean SimultExc, [in] EGroupCombinationType CombinationType)	
	Name	<i>name of the load group</i>
	LoadGroupType	<i>load group type</i>
	SimultExc	<i>(if LoadGroupType = IgtIncidental) set if load cases of the group can be simultaneous with load cases of exceptional groups</i>
	CombinationType	<i>Behaviour of load cases of groups when determining the critical combination.</i>

Adds a new load group to the model.

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

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

Index *index of the load group to delete ($0 < Index \leq Count$)*

Deletes a load group.

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

Properties

long **Count** *Get number of load groups in the model.*

Negative number is an error code ([errDatabaseNotReady](#)).

[AxisVMLoadGroup](#)* **Item** [long **Index**] *Get load group interface by index*

IAxisVMLoadGroup

An AxisVM load group interface.

Properties

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

[**EGroupCombinationType**](#) **CombinationType** • Get or set behaviour of load cases of groups when determining the critical combination

double **DynFact** • (MSz: μ) Get or set dynamic factor for incidental load groups

double **GammalInf** • (EC: γ_{GL} , NEN: $\gamma_{f,GL}$) Get or set lower partial factor for permanent load groups

double **GammaSup** • Get or set:

permanent load groups: (EC: γ_{GU} , NEN: $\gamma_{f,Gu}$) upper partial factor

incidental load groups: (EC: γ_Q , NEN: $\gamma_{f,q}$) partial factor

double **GammaFsw** • (NEN: γ_2) Get or set safety factor for self load

double **Gammal** • Get or set seismic safety factor (γ_l)

double **Ksi** • Get or set ξ reduction factor for unfavourable permanent loads

[**ELoadGroupType**](#) **LoadGroupType** • Get or set load group type

BSTR **Name** • Get or set name of the load group

double **PermLoadRatio** • (NEN: r_p) permanent load ratio

double **Psi** • (NEN: ψ) Get or set ψ load combination factor

double **Psi0** • (EC: ψ_0) Get or set ψ_0 load combination factor

double **Psi1** • (EC: ψ_1) Get or set ψ_1 load combination factor

double **Psi2** • (EC: ψ_2) Get or set ψ_2 load combination factor

double **PsiT** • (NEN: ψ_t) Get or set ψ_t load combination factor

double **SfactInf** • (MSz: γ_A) Get or set lower safety factor for permanent load groups

double **SfactUp** • (MSz: γ_F) Get or set upper safety factor for permanent load groups
(MSz: γ) safety factor for incidental load groups /

double **SimFact** • (MSz: α) Get or set simultaneity factor for incidental load groups

[**ELongBoolean**](#) **SimultExc** • (for incidental load groups) Get or set load cases of the group can be simultaneous with load cases of exceptional groups

IAxisVMLoadPanels

Load panels of the model used for generated wind and snow loads.

If property returning this interface is null (nil) then the extension module SWG is not available.

Error codes

```
enum ELoadPanelsError = {
    lopelInvalidContourParams = -100001,           Size of the array does not match total number of edges
    lopeLineIndexListEmpty = -100002,               Array of line indexes is empty
    lopeLineIndexOutOfBounds = -100003,              Line index is out of bounds
    lopelInvalidContour = -100004,                   Contour is invalid, e.g. not closed, has only 2 points, etc.
    lopeSameLoadPanelExists = -100005,              same load panel already exists
    lopeMemberIndexListEmpty = -100006,              Array of member indexes is empty
    ,
    lopeMemberIndexOutOfBounds = -100007,            Member index is out of bounds
    lopelInvalidContourType = -100008,               Type of the contour is invalid
    lopeNodeIndexListEmpty = -100009,               Array of node indexes is empty
    lopeDomainIndexListEmpty = -100010,              Array of domain indexes is empty
}
```

Functions

long	AddFromDomain ([in] long DomainID)	DomainID Domain index Adds a new load panel to the model. If successful, returns the index of the new load panel, otherwise returns an error code (errDatabaseNotReady or errIndexOutOfBoundsException).
long	AddFromLineIDs ([in] SAFEARRAY (long) LineIDs)	LineIDs Index of lines defining the contour of the load panel Adds a new load panel to the model. If successful, returns the index of the new load panel, otherwise returns an error code (errDatabaseNotReady , lopeLineIndexOutOfBounds , lopeLineIndexListEmpty or lopelInvalidContour).
long	AddFromMemberIDs ([in] SAFEARRAY (long) MemberIDs)	MemberIDs Index of members defining the contour of the load panel Adds a new load panel to the model. If successful, returns the index of the new load panel, otherwise returns an error code (errDatabaseNotReady , lopeMemberIndexOutOfBounds , lopeMemberIndexListEmpty or lopelInvalidContour).
long	AddFromPolygon ([in] AxisVMLines3d * ContourPoly)	ContourPoly Generic contour polygon of the load panel Adds a new load panel to the model. If successful, returns the index of the new load panel, otherwise returns an error code (errDatabaseNotReady , lopeSameLoadPanelExists or lopelInvalidContour).
long	Clear	Clears all load panels from the model. If successful, returns number of deleted panels, otherwise returns an error code (errDatabaseNotReady).
long	Delete ([in] long LoadPanelID)	LoadPanelID Load panel index Delete load panel from the model. If successful, returns index of deleted load panel, otherwise returns an error code (errDatabaseNotReady or errIndexOutOfBoundsException).

long **DeleteSelected**

Delete selected load panel from the model. If successful, returns index of deleted load panel, otherwise returns an error code ([errDatabaseNotReady](#)).

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

Select selection state

If Select is True, selects all load panels.

If Select is False, deselects all load panels.

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

NOTE: Call [Refresh](#) function afterwards if not called between functions [BeginUpdate](#) and [EndUpdate](#)

Properties

long **Count** *Get number of load panels in the model*

[IAxisVMLoadPanel](#) * **Item [long Index]** *Get load panel interface by index*

Index *index of the load panel*

[ELongBoolean](#) **Selected [long Index]** • *Get or set the selection status of a load panel*

Index *index of the load panel*

NOTE: Call [Refresh](#) function afterwards if not called between functions [BeginUpdate](#) and [EndUpdate](#)

long **SelCount** *Get number of selected load panels in the model*

long **UID [long Index]** *Get unique index of the load panel which remains the same while exists in the model*

Index *index of the load panel*

IAxisVMLoadPanel

Load panel of the model used for generated wind and snow loads.

Enumerated types

enum	ELoadPanelContourType = {	
	lpctUserDefined =	User defined (generic) type of the load panel contour, not associated with lines, members, domains.
	0x0,	
	lpctAssociated =	The load panel contour is associated with lines, members, domains.
	0x1 }	
		Type of load panel contour
enum	ELoadPanelEdgeType = {	
	lpetNone = 0x0,	Nothing on the edge
	lpetParapet = 0x1,	Parapet on the edge set other parameters
	lpetWall = 0x2 }	Wall on the edge set other parameters
		Type of load panel edge

Records / structures

<u>ELoadPanelEdgeType</u>	RLoadPanelEdgeParams = (
	LoadPanelEdgeType	Type of load panel edge
Double	h	Height of wall or parapet
Double	Alpha	Angle of the roof above the abutting wall
Double	b_1	Width of the taller construction
)		

Functions

long	GetAllEdgeParameters ([i/o] SAFEARRAY (RLoadPanelEdgeParams)* AllEdgeParameters)	
	AllEdgeParameters	Array containing all edge parameters of the load panel
		Get all edge parameters of the load panel. Returns number of edges if successful, otherwise returns an error code (errDatabaseNotReady or lopeInvalidContour)
long	GetContourLinelds ([out] SAFEARRAY(long)* Linelds)	
	Linelds	Index array (long) with line indexes
		Returns number of lines in load panel's contour, otherwise returns an error code (errDatabaseNotReady or lopeInvalidContourType).
long	GetContourPolygon ([out] IAxisVMLines3d * ContourPolygon)	
	ContourPolygon	Interface with contour's coordinates
		Returns number of edges of the load panel, otherwise returns an error code (errDatabaseNotReady or lopeInvalidContourType).
long	GetEdgeParameters ([in] long Index , [i/o] RLoadPanelEdgeParams EdgeParameters)	
	Index	Index of the edge
	EdgeParameters	Edge parameters of the load panel
		Get edge parameters of one load panel edge. Returns edge index if successful, otherwise returns an error code (errDatabaseNotReady or errIndexOutOfBounds)
long	GetDomains ([out] SAFEARRAY(long)* DomainIds)	
	DomainIds	Index array (long) with domain indexes where load from panel is applied
		Returns DomainIds array length, otherwise returns an error code (errDatabaseNotReady).
long	GetLines ([out] SAFEARRAY(long)* Linelds)	
	Linelds	Index array (long) with line indexes where load from panel is applied
		Returns Linelds array length, otherwise returns an error code (errDatabaseNotReady).

long	GetNodes ([out] SAFEARRAY(long)* Nodelds)
	Nodelds Index array (long) with node indexes where load from panel is applied
<i>Returns Nodelds array length, otherwise returns an error code (errDatabaseNotReady).</i>	
long	SetAllEdgeParameters ([i/o] SAFEARRAY (RLoadPanelEdgeParams)* AllEdgeParameters)
	AllEdgeParameters Array containing all edge parameters of the load panel
	Set all edge parameters of the load panel. Returns number of edges if successful, otherwise returns an error code (errDatabaseNotReady or lopelInvalidContourParams)
long	SetEdgeParameters ([in] long Index, [i/o] RLoadPanelEdgeParams EdgeParameters)
	Index Index of the edge
	EdgeParameters Edge parameters of the load panel
	Set edge parameters of one load panel edge. Returns edge index if successful, otherwise returns an error code (errDatabaseNotReady , errIndexOutOfBoundsException)
long	SetNodes ([i/o] SAFEARRAY(long)* Nodelds)
	Nodelds Index array (long) with node indexes where load from panel is applied
	Returns Nodelds array length, otherwise returns an error code (errDatabaseNotReady or ELoadPanelsError).
long	SetLines ([i/o] SAFEARRAY(long)* Linelds)
	Linelds Index array (long) with line indexes where load from panel is applied
	Returns Linelds array length, otherwise returns an error code (errDatabaseNotReady or ELoadPanelsError).
long	SetDomains ([i/o] SAFEARRAY(long)* DomainIds)
	DomainIds Index array (long) with domain indexes where load from panel is applied
	Returns DomainIds array length, otherwise returns an error code (errDatabaseNotReady or ELoadPanelsError).

Properties

<u>ELongBoolean</u>	Auto • Get or set whether the load panel will apply loads automatically. <i>lbTrue: load will be applied automatically to all nodes and lines within the contour</i> <i>lbFalse: load will be applied only to set nodes and lines(See SetNodes and SetLines functions)</i>
<u>ELoadPanelContourType</u>	ContourType Get type of the load panel
long	EdgeCount Get number of edges of the load panel
<u>EBoolean</u>	SelectedEdge [long Index] • Get or set the selection status of a load panel's edge <i>Index index of the load panel's edge</i> <i>NOTE: Call Refresh function afterwards if not called between functions BeginUpdate and EndUpdate</i>

IAxisVMLoads

Loads of the model

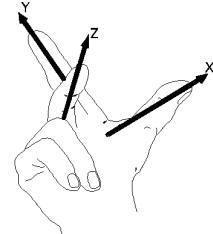
NOTE:

If load is applied to rib, then the point of application of defined loads are transferred, explained [here](#).

Enumerated types

```
enum EAxis = {
    Ax = 0x00,
    Ay = 0x01,
    Az = 0x02,
```

x direction
y direction
z direction



```
aXX = 0x03,
aYY = 0x04,
aZZ = 0x05 }
```

about x axis
about y axis
about z axis



Directions.

```
enum EBBeamRibDistributionType = {
```

```
brdtLength = 0x00,           by length
brdtProjected = 0x01 }       projected
```

Beam/rib load distribution type.

```
enum EDistributionType = {
```

```
dtGlobal = 0x00,
```

Global

```
dtLocal = 0x01,
```

Local

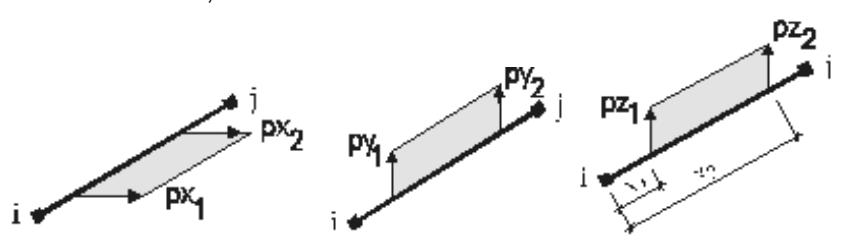
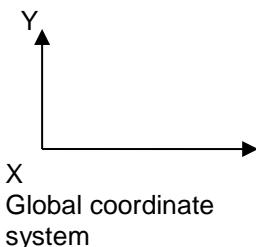
```
dtProjected = 0x02
```

projected

```
dtEdgeLocal = 0x03 }
```

dtEdgeLocal

Mesh-independent surface load distribution type.

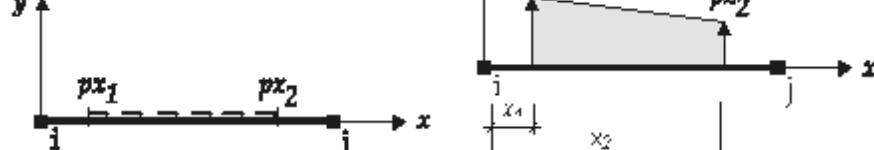


dtProjected = 0x02

projected

dtEdgeLocal = 0x03 }

dtEdgeLocal



dtProjected = 0x02

projected

dtEdgeLocal = 0x03 }

dtEdgeLocal

Mesh-independent surface load behaviour.

```
enum ELoadDistributionType = {
```

```
ldtConst = 0x00,
ldtLinear = 0x01 }
```

constant load intensity

linear load intensity

Mesh-independent surface load behaviour.

enum	ELoadType = {	
	ItNodalForce = 0x00,	nodal force, see record RLoadNodalForce
	ItBeamConcentrated = 0x01,	concentrated load on beams, see record RLoadBeamConcentrated
	ItBeamDistributed = 0x02,	distributed load on beams, see record RLoadBeamDistributed
	ItBeamThermal = 0x03,	thermal load on beams, see record RLoadBeamThermal
	ItBeamStress = 0x04,	tension/compression load on beams, see record RLoadBeamStress
	ItBeamFault = 0x05,	„fault in length“ load on beams, see record RLoadBeamFault
	ItBeamSelfWeight = 0x07,	self weight of beam
	ItTrussThermal = 0x08,	thermal load on trusses, see record RLoadTrussThermal
	ItTrussStress = 0x09,	tension/compression load on trusses, see record RLoadTrussStress
	ItTrussFault = 0x0A,	„fault in length“ load on trusses, see record RLoadTrussFault
	ItTrussSelfWeight = 0x0B,	self weight of truss
	ItSurfaceSelfWeight = 0x0C,	self weight of surface element
	ItSurfaceDistributed = 0x0D,	distributed load on surface elements, see record RLoadSurfaceDistributed
	ItSurfaceEdge = 0x0E,	edge load on surface elements, see record RLoadSurfaceEdge
	ItSurfaceThermal = 0x0F,	thermal load on surface elements, see record RLoadSurfaceThermal
	ItSurfaceStress = 0x10,	(not used)
	ItBeamInfluence = 0x11,	influence line load on beams, see record RLoadBeamInfluence
	ItDomainSelfWeight = 0x12,	self weight of domain
	ItDomainConstant = 0x13,	constant area load on domain, see record RLoadDomainConstant
	ItDomainEdge = 0x14,	(not used)
	ItDomainThermal = 0x15,	thermal load on domains, see record RLoadDomainThermal
	ItDomainStress = 0x16,	(not used)
	ItRibThermal = 0x17,	thermal load on ribs, see record RLoadRibThermal
	ItRibSelfWeight = 0x18,	self weight of rib
	ItRibConcentrated = 0x19,	concentrated load on ribs, see record RLoadRibConcentrated
	ItRibDistributed = 0x1A,	distributed load on ribs, see record RLoadRibDistributed
	ItSupportDisplacement = 0x1B,	support displacement, see record RLoadSupportDisplacement
	ItDomainConcentrated = 0x1C,	concentrated load on domains, see record RLoadDomainConcentrated
	ItSurfaceConcentrated = 0x1E,	concentrated load on surface elements, see record RLoadSurfaceConcentrated
	ItDomainPolyArea = 0x21,	distributed polygon area load on domains, see record RLoadDomainPolyArea
	ItDomainLinear = 0x22,	distributed area load on domains, see record RLoadDomainLinear
	ItDomainFluid = 0x24,	fluid load on domains, see record RLoadDomainFluid
	ItSurfaceFluid = 0x25,	fluid load on surface elements, see record RLoadSurfaceFluid
	ItLoadDomainPolyLine = 0x26,	polygon load on domain, see record RLoadDomainPolyLine , to read all items use GetDomainPolyLineItems
	ItSurfaceToBeam = 0x27,	surface load distributed over beams, see record RLoadSurfaceToBeam

ItDomainPolyAssoc = 0x28,	associative edge load on domains, see record RLoadDomainPolyAssoc , Access to associated lines through functions GetLines / SetLines
ItSurfaceToBeamAssoc = 0x29,	associative surface load distributed over beams, see record RLoadSurfaceToBeamAssoc
ItDynamicNodalForce = 0x2A	dynamic nodal force, see record RLoadDynamic
ItDynamicNodalAcceleration = 0x2B,	dynamic nodal acceleration, see record RLoadDynamic
ItDynamicNodalSupportAcceleration = 0x2C,	dynamic nodal support acceleration, see record RLoadDynamic
ItBeamMemberConcentrated = 0x2D,	concentrated force on beam structural member, see record RLoadBeamMemberConcentrated
ItBeamMemberDistributed = 0x2E,	distributed force on beam structural member, see record RLoadBeamMemberDistributed
ItRibMemberConcentrated = 0x2F,	concentrated force on rib structural member, see record RLoadRibMemberConcentrated
ItRibMemberDistributed = 0x30,	distributed force on rib structural member, see record RLoadRibMemberDistributed
ItNone = 0xFFFFFFFF }	none of the above

Load types.

enum **ESurfaceDomainDistributionType** = {
sddtSurface = 0x00,
sddtProjected = 0x01 }

Surface/domain load distribution type.

enum **ESystem** = {
sysGlobal = 0x00,
sysLocal = 0x01,
sysReference = 0x02 }

Coordinate system of load components.

enum **ELoadDomainPolyLineItemType** = {

ldpitDomain = 0x00,	<i>on domain</i>
ldpitSurface = 0x01,	<i>on surface</i>
ldpitBlankLine = 0x02,	<i>on blank (virtual) line</i>
ldpitLoadPanel = 0x03 }	<i>on load panel</i>

Type of the loaded element

Error codes

enum **ELoadsError** = {

leInvalidLineType = -100001	<i>load type is not compatible with the line type</i>
leErrorAddingLoad = -100002	<i>error when adding load</i>
leErrorSettingLoad = -100003	<i>error when setting load</i>
leInvalidLoadType = -100004	<i>invalid operation for this load type</i>
leNotValidLineTypeForThisLoad = -100005	<i>invalid line type when distributing surface load on lines</i>
leErrorSettingLines = -100006	<i>cannot set line list when distributing surface load on lines</i>
leErrorSettingPoly = -100007	<i>cannot set the load polygon</i>
leLoadCaseIndexOutOfBounds = -100008	<i>load record contains an invalid load case index</i>
leErrorSettingLoadCaseId = -100009	<i>load case index cannot be set</i>
leDomainIndexOutOfBounds = -100010	<i>load record contains an invalid domain index</i>
leMemberIndexOutOfBounds = -100011	<i>load record contains an invalid member index</i>
leThereAreNoSeismicStoreys = -100012	<i>CreateStandardSeismicLoads can return this error when the user requires torsion forces but the seismic storeys are not set</i>
loeNotDynamicLoadCase = -100013	<i>if the load case is not dynamic</i>
loeReferenceIndexOutOfBounds = -100014	<i>Reference index is out of boundaries</i>
loeErrorCreatingPushOverLoads = -100015	<i>Error while creating pushover loads</i>
loeInvalidLoadCaseType = -100016	<i>Load case type not valid</i>
loeInvalidRootType = -100017	<i>Roof type not supported for this load</i>
loeLoadPanelIndexListEmpty = -100018	<i>Array with load panel indexes is empty</i>
loeNoPushOverLoadCase = -100019	<i>Create pushover load cases before calling the function returning this error</i>
loeSE1moduleNotAvailable = -100020	<i>extension module SE1 is not available</i>

```

loeSE2moduleNotAvailable = -100021
loeDYNmoduleNotAvailable = -100022
loeSWGmoduleNotAvailable = -100023
loeNoSnowLoadCase = -100024
loeNoWindLoadCase = -100025
loeNoSeismicLoadCase = -100026,
loePointIsOutOfLoadPanel = -100027
loeZeroLoadValueOnLoadPanel = -
100028
loeDerivedSurfaceLoadsNotConverted = -
100029
}

```

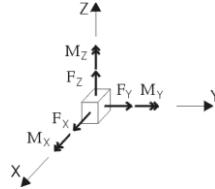
extension module SE2 is not available
extension module DYN is not available
extension module SWG is not available
Create snow load cases before calling the function returning this error
Create wind load cases before calling the function returning this error
Create seismic load cases before calling the function returning this error
the point is out of the load panel
load with zero value has been defined
derived surface loads have not been converted

Records / structures

Nodal loads

RLoadNodalForce = (

long	LoadCaseld	load case index ($0 < \text{LoadCaseld} \leq \text{AxisVMLoadCases.Count}$)
long	Nodeld	node index ($0 < \text{Index} \leq \text{AxisVMNodes.Count}$)
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 $0 < \text{Referenceld} \leq \text{AxisVMReferences.Count}$, the load direction is set by the reference. Referenceld is the index of reference according to AxisVMReferences .
)		



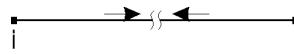
Truss loads

RLoadTrussFault = (

long	LoadCaseld	load case index ($0 < \text{LoadCaseld} \leq \text{AxisVMLoadCases.Count}$)
long	Lineld	line index ($0 < \text{Index} \leq \text{AxisVMLines.Count}$)
double	DL	fault in length
)		

RLoadTrussStress = (

long	LoadCaseld	load case index ($0 < \text{LoadCaseld} \leq \text{AxisVMLoadCases.Count}$)
long	Lineld	line index ($0 < \text{Index} \leq \text{AxisVMLines.Count}$)
double	P	tension/compression if $P > 0$, there is a tension at the endpoints if $P < 0$, there is a compression at the endpoints +P
)		



RLoadTrussThermal = (

long	LoadCaseld	load case index ($0 < \text{LoadCaseld} \leq \text{AxisVMLoadCases.Count}$)
long	Lineld	line index ($0 < \text{Index} \leq \text{AxisVMLines.Count}$)
double	Tref	reference temperature
double	T0	actual truss temperature
)		

Beam loads

RLoadBeamConcentrated = (

long	LoadCaseld	load case index ($0 < \text{LoadCaseld} \leq \text{AxisVMLoadCases.Count}$)
long	Lineld	line index ($0 < \text{Index} \leq \text{AxisVMLines.Count}$)
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 = (

long	LoadCaseld	load case index ($0 < \text{LoadCaseld} \leq \text{AxisVMLoadCases.Count}$)
long	Lineld	line index ($0 < \text{Index} \leq \text{AxisVMLines.Count}$)
double	qx1, qy1, qz1	x, y, z force components [kN/m] at the 1 st point
double	mx1, my1, mz1	moment components about the x, y, z axis [kNm/m] at the 1 st point
double	qx2, qy2, qz2	x, y, z force components [kN/m] at the 2 nd point
double	mx2, my2, mz2	moment components about the x, y, z axis [kNm/m] at the 2 nd 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
EBeamRibDistributionType	double	Position2	position of the 2nd point with the same sign convention
EBoolean		DistributionType	distributed by length or projected
)	trapezoid load
		RLoadBeamFault = (
long	long	LoadCaseId	load case index ($0 < \text{LoadCaseId} \leq \text{AxisVMLoadCases.Count}$)
		LinId	line index ($0 < \text{Index} \leq \text{AxisVMLines.Count}$)
		DL	fault in length
)	See Fault in Length (Fabrication Error) in AxisVM manual
		RLoadBeamInfluence = (
long	long	LoadCaseId	load case index ($0 < \text{LoadCaseId} \leq \text{AxisVMLoadCases.Count}$)
		LinId	line index ($0 < \text{Index} \leq \text{AxisVMLines.Count}$)
		ex, ey, ez	relative displacements of the influence line load (-1, 0, 1)
		Fx, Fy, Fz	relative rotations of the influence line load (-1, 0, 1)
		Position	if Position ≥ 0 , the beam influence [m], if Position < 0 , the absolute value is beam influence /line length
)	
		RLoadBeamStress = (
long	long	LoadCaseId	load case index ($0 < \text{LoadCaseId} \leq \text{AxisVMLoadCases.Count}$)
		LinId	line index ($0 < \text{Index} \leq \text{AxisVMLines.Count}$)
		P	tension/compression force if Force > 0 , tension is applied if Force < 0 , compression is applied
)	+P
		RLoadBeamThermal = (
long	long	LoadCaseId	load case index ($0 < \text{LoadCaseId} \leq \text{AxisVMLoadCases.Count}$)
		LinId	line index ($0 < \text{Index} \leq \text{AxisVMLines.Count}$)
		Tref	reference temperature
		Ttop	top cord temperature (in the Axis direction)
		Tbot	bottom cord temperature (in the Axis direction)
		Axis	direction of temperature variation (in local y or z direction only)
)	
		RLoadSurfaceToBeam = (
long	EDistributionType	LoadCaseId	load case index ($0 < \text{LoadCaseId} \leq \text{AxisVMLoadCases.Count}$)
		DistributionType	type of the distributed load (global / local / projected)
		Px, Py, Pz	surface load intensity [kN/m^2]
)	
		RLoadSurfaceToBeamAssoc = (
long	EDistributionType	LoadCaseId	load case index ($0 < \text{LoadCaseId} \leq \text{AxisVMLoadCases.Count}$)
		DistributionType	a distributed load típusa (global / local / projected)
		Px, Py, Pz	surface load intensity [kN/m^2]
)	
Rib loads			
		RLoadRibConcentrated = (
long	long	LoadCaseId	load case index ($0 < \text{LoadCaseId} \leq \text{AxisVMLoadCases.Count}$)
		LinId	line index ($0 < \text{Index} \leq \text{AxisVMLines.Count}$)
		Fgx, Fgy, Fgz	x, y, z force components [kN]
		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
coordinate system of load components
if SystemGLR = sysReference, only Fgx and Mgx are valid
)

RLoadRibDistributed =
long LoadCaseId
long LineId
double qx1, qy1, qz1
double mx1, my1, mz1
double qx2, qy2, qz2
double mx2, my2, mz2
ESystem double SystemGLR
Position1
load case index (0 < LoadCaseId ≤ AxisVMLoadCases.Count)
line index (0 < Index ≤ AxisVMLines.Count)
x, y, z force components [kN/m] at the 1st point
moment components about the x, y, z axis [kNm/m] at the 1st point
x, y, z force components [kN/m] at the 2nd point
moment components about the x, y, z axis [kNm/m] at the 2nd point
coordinate system of load components
position of the 1st point
if Position ≥ 0, the load position [m],
if Position < 0, the absolute value is load position/line length
position of the 2nd point with the same sign convention
distributed by length or projected
trapezoid load

EBeamRibDistributionType
ELongBoolean
)

RLoadRibThermal =
long LoadCaseId
long LineId
Tref
Ttop
Tbot
Axis
)
load case index (0 < LoadCaseId ≤ AxisVMLoadCases.Count)
line index (0 < Index ≤ AxisVMLines.Count)
reference temperature
top cord temperature (in the Axis direction)
bottom cord temperature (in the Axis direction)
direction of temperature variation (in local y or z direction only)

Structural member loads

RLoadBeamMemberConcentrated =
long LoadCaseId
long MemberId
Fgx, Fgy, Fgz
Mgx, Mgy, Mgz
Position
)
load case index (0 < LoadCaseId ≤ AxisVMLoadCases.Count)
member index
x, y, z force components [kN]
moment components about the x, y, z axis [kNm]
if Position ≥ 0, the load position [m],
if Position < 0, the absolute value is load position/member length
coordinate system of load components
if SystemGLR = sysReference, only Fgx and Mgx are valid

RLoadBeamMemberDistributed =
long LoadCaseId
long MemberId
qx1, qy1, qz1
mx1, my1, mz1
double qx2, qy2, qz2
double mx2, my2, mz2
ESystem double SystemGLR
Position1
load case index (0 < LoadCaseId ≤ AxisVMLoadCases.Count)
member index
x, y, z force components [kN/m] at the 1st point
moment components about the x, y, z axis [kNm/m] at the 1st point
x, y, z force components [kN/m] at the 2nd point
moment components about the x, y, z axis [kNm/m] at the 2nd point
coordinate system of load components
position of the 1st point
if Position ≥ 0, the load position [m],
if Position < 0, the absolute value is load position/member length
position of the 2nd point with the same sign convention
distributed by length or projected
trapezoid load

EBeamRibDistributionType
ELongBoolean
)

RLoadRibMemberConcentrated =
long LoadCaseId
long MemberId
Fgx, Fgy, Fgz
Mgx, Mgy, Mgz
Position
)
load case index (0 < LoadCaseId ≤ AxisVMLoadCases.Count)
member index
x, y, z force components [kN]
moment components about the x, y, z axis [kNm]
if Position ≥ 0, the load position [m],
if Position < 0, the absolute value is load position/member length
coordinate system of load components
if SystemGLR = sysReference, only Fgx and Mgx are valid
)

```

```

RLoadRibMemberDistributed = (
    long LoadCasId           load case index (0 < LoadCasId ≤ AxisVMLoadCases.Count)
    long MemberId            member index
    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
    double Position2          if Position ≥ 0, the load position [m], if Position < 0, the absolute value is load position/member length
    EBeamRibDistributionType DistributionType position of the 2nd point with the same sign convention
    ELongBoolean              distributed by length or projected trapezoid load
)

```

Surface loads

```

RLoadSurfaceConcentrated = (
    long LoadCasId           load case index (0 < LoadCasId ≤ AxisVMLoadCases.Count)
    long SurfacId            surface index (0 < Index ≤ AxisVMSurfaces.Count)
    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
    long ReferencId          If set to 0, load components are in global directions.
    ESystem SystemGLR        If 0 < ReferencId ≤ AxisVMReferences.Count, the load direction is set by the reference. ReferencId is the index of reference according to AxisVMReferences.
)

```

```

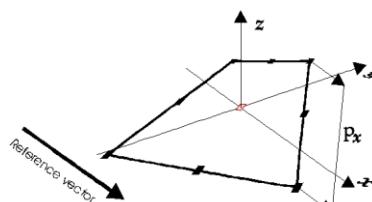
RLoadSurfaceDistributed = (
    long LoadCasId           load case index (0 < LoadCasId ≤ AxisVMLoadCases.Count)
    long SurfacId            surface index (0 < Index ≤ AxisVMSurfaces.Count)
    double qx, qy, qz         x, y, z force components [kN/m2]
    ESystem SystemGLR        coordinate system of load components (local or global only)
    double DistributionType   type of the distributed load
)

```

```

RLoadSurfaceEdge = (
    long LoadCasId           load case index (0 < LoadCasId ≤ AxisVMLoadCases.Count)
    long SurfacId            surface index (0 < Index ≤ AxisVMSurfaces.Count)
    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 System1, System2, load coordinate systems on edges (local or global only)
    System3, System4
    ELongBoolean EdgeLoaded1, True, if a load is applied on the edge connecting corner nodes
    EdgeLoaded2, (For quadrilateral elements: 1 → 2, 2 → 3, 3 → 4, 4 → 1)
    EdgeLoaded3, (For triangular elements: 1 → 2, 2 → 3, 3 → 1)
    EdgeLoaded4, For triangular elements the fourth component is ignored.
    ESurfaceDomainDistributionType DistributionType1, edge load in local x direction
)

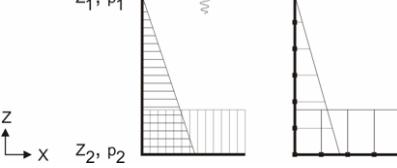
```



RLoadSurfaceFluid = (

long	LoadCaseld	load case index ($0 < \text{LoadCaseld} \leq \text{AxisVMSurfaceLoadCases.Count}$)
long	Surfaceld	surface index ($0 < \text{Index} \leq \text{AxisVMSurfaces.Count}$)
<u>EAxis</u>	Direction	direction of fluid load variation, x,y or z direction only
double	Coord1, Coord2	coordinate of the 1st and 2nd point
double	P1, P2	fluid load intensity at the 1st and 2nd point [kN/m^2]

)



RLoadSurfaceThermal = (

long	LoadCaseld	load case index ($0 < \text{LoadCaseld} \leq \text{AxisVMSurfaceLoadCases.Count}$)
long	Surfaceld	surface index ($0 < \text{Index} \leq \text{AxisVMSurfaces.Count}$)
<u>Tref</u>	Tref	reference temperature
double	Ttop	top temperature according to the local z direction
double	Tbot	bottom temperature according to the local z direction

)

Domain loads

RLoadDomainPolyArea = (

long	LoadCaseld	load case index ($0 < \text{LoadCaseld} \leq \text{AxisVMSurfaceLoadCases.Count}$)
<u>EDistributionType</u>	DistributionType	type of the distributed load (global / local / projected)
<u>ELoadDistributionType</u>	LoadDistributionType	type of the distributed load (constant / linear intensity)
<u>EAxis</u>	Component	load component
double	P1, P2, P3	load intensity at the load reference points [kN/m^2], if LoadDistributionType is constant the P1 is used for intensity
double	x1, y1, z1	coordinates of the 1st load reference point
double	x2, y2, z2	0 for LoadDistributionType = ldtConst
double	x3, y3, z3	coordinates of the 2nd load reference point
<u>ELongBoolean</u>	WindowLoad	0 for LoadDistributionType = ldtConst
		coordinates of the 3rd load reference point
		0 for LoadDistributionType = ldtConst
		window load (load falling on an opening is distributed along the edges)

)

RLoadDomainLinear = (

long	LoadCaseld	load case index ($0 < \text{LoadCaseld} \leq \text{AxisVMSurfaceLoadCases.Count}$)
long	DomainId	domain index ($0 < \text{Index} \leq \text{AxisVMDomains.Count}$)
<u>EDistributionType</u>	DistributionType	type of the distributed load (global / local / projected)
<u>ELoadDistributionType</u>	LoadDistributionType	type of the distributed load (constant / linear intensity)
<u>EAxis</u>	Component	load component
double	P1, P2, P3	load intensity at the load reference points [kN/m^2], if LoadDistributionType is constant the P1 is used for intensity
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
<u>ELongBoolean</u>	WindowLoad	window load (load falling on an opening is distributed along the edges)

)

RLoadDomainConcentrated = (

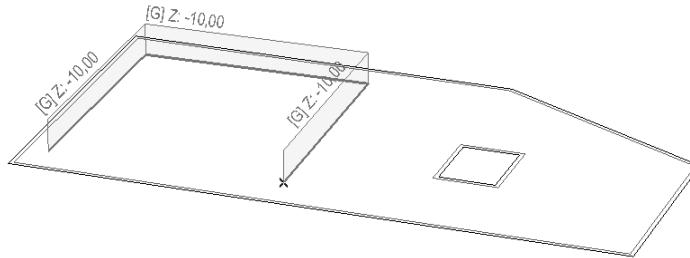
long	LoadCaseld	load case index ($0 < \text{LoadCaseld} \leq \text{AxisVMSurfaceLoadCases.Count}$)
long	DomainId	domain index ($0 < \text{Index} \leq \text{AxisVMDomains.Count}$)
double	Fx, Fy, Fz	x, y, z force components [kN]
double	Mx, My, Mz	moment components about the x, y, z axis [kNm]
double	x, y, z	global load position
long	Referenceld	If set to 0, load components are in global directions. If $0 < \text{Referenceld} \leq \text{AxisVMReferences.Count}$, the load direction is set by the reference. Referenceld is the index of reference according to AxisVMReferences .
<u>ESystem</u>	SystemGLR	load coordinate system

)

RLoadDomainConstant = (
 long
 long
 double
ESurfaceDomainDistributionType
EDistributionType
ESystem
)
SystemGLR
)

RLoadDomainFluid = (
 long
 long
EAxis
 double
 double
 long
)
LoadCasId
DomainId
Direction
Coord1, Coord2
P1, P2
DomainId
)

RLoadDomainPolyLine = (
 long
 double
 double
 double
 double
EDistributionType
 double
)
LoadCasId
px1, px2
py1, py2
pz1, pz2
pm1, pm2
DistributionType
Nx, Ny, Nz
)



RLoadDomainPolyAssoc = (
 long
 double
 double
 double
 double
EDistributionType
 double
)
LoadCasId
px1, px2
py1, py2
pz1, pz2
pm1, pm2
DistributionType
Nx, Ny, Nz
)

RLoadDomainPolyLineItem = (
ELoadDomainPolyLineItemType
 double
 double
 double
 double
 long
RPoint3d
RPoint3d
ELineGeomType
RCircleArcGeomData
)
ItemType
px1, px2
py1, py2
pz1, pz2
pm1, pm2
ElementId
StartPoint
EndPoint
LineGeomType
CircleArcGeomData
)

```

RLoadDomainThermal = (
    long LoadCaseId           load case index (0 < LoadCaseId < AxisVMLoadCases.Count)
    long DomainId             domain index (0 < Index < AxisVMDomains.Count)
    double Tref                reference temperature
    double Tsup               top temperature according to the local z direction
    double Tinf               bottom temperature according to the local z direction
)

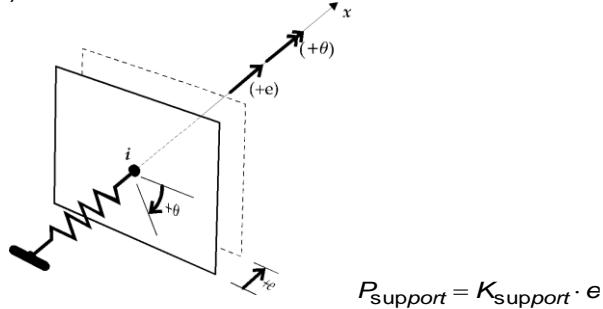
```

Support loads

```

RLoadSupportDisplacement = (
    long LoadCaseId           load case index ( $0 < \text{LoadCaseId} \leq \text{AxisVMLoadCases}.\text{Count}$ )
    long SupportId            nodal support index ( $0 < \text{Index} \leq \text{AxisVMNodalSupports}.\text{Count}$ )
    double ex, ey, ez          support displacement in x, y, z directions
    double fx, fy, fz          support rotation about the x, y, z axis
)

```



<u>ELoadType</u>	RLoadDynamic = (
	DynamicLoadType	<i>type of dynamic load, see here</i>
long	LoadCaseId	<i>load case index</i>
long	NodeId	<i>node index</i>
double	Fx	<i>nodal force in x direction</i>
double	Fy	<i>nodal force in y direction</i>
double	Fz	<i>nodal force in z direction</i>
double	Mx	<i>nodal moment about x direction</i>
double	My	<i>nodal moment about y direction</i>
double	Mz	<i>nodal moment about z direction</i>
long	Referenceld	<i>index of the reference</i>
long	FxFunctionId	<i>IAxisVMDynamicLoadFunctions index for force in x direction</i>
long	FyFunctionId	<i>IAxisVMDynamicLoadFunctions index for force in y direction</i>
long	FzFunctionId	<i>IAxisVMDynamicLoadFunctions index for force in z direction</i>
long	MxFunctionId	<i>IAxisVMDynamicLoadFunctions index for moment about x direction</i>
long	MyFunctionId	<i>IAxisVMDynamicLoadFunctions index for moment about y direction</i>
long	MzFunctionId	<i>IAxisVMDynamicLoadFunctions index for moment about z direction</i>
)		

Load panel loads

	RLoadPanelPolyArea = (
long EDistributionType	LoadCaseId	<i>load case index (0 < LoadCaseId ≤ AxisVMLoadCases.Count)</i>
ELoadDistributionType	DistributionType	<i>type of the distributed load (global / local / projected)</i>
EAxis	LoadDistributionType	<i>type of the distributed load (constant / linear intensity)</i>
double	Component	<i>load component (only aX, aY and aZ are accepted)</i>
	P1, P2, P3	<i>load intensity at the load reference points [kN/m²], if LoadDistributionType is constant the P1 is used for intensity</i>
double	x1, y1, z1	<i>global coordinates of the 1st load reference point</i>
double	x2, y2, z2	<i>global coordinates of the 2nd load reference point</i>
double	x3, y3, z3	<i>global coordinates of the 3rd load reference point</i>
ELongBoolean	WindowLoad	<i>window load (load falling on an opening is distributed along the edges)</i>
)	

```

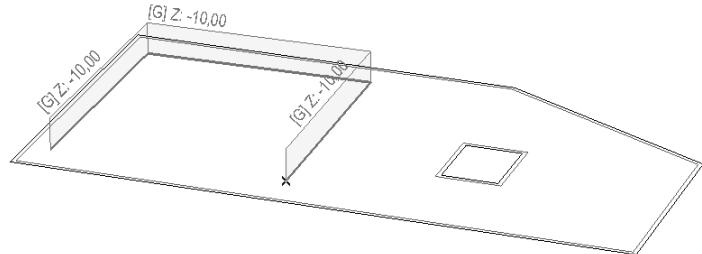
RLoadPanelLinear = (
    long LoadCaseId           load case index (0 < LoadCaseId ≤ AxisVMLoadCases.Count)
    long LoadPanelId          load panel index (0 < Index ≤ AxisVMLoadPanels.Count)
    EDistributionType DistributionType type of the distributed load (global / local / projected)
    ELoadDistributionType LoadDistributionType type of the distributed load (constant / linear intensity)
    EAxis Component          load component (only aX, aY and aZ are accepted)
    double P1, P2, P3          load intensity at the load reference points [kN/m2], if LoadDistributionType is constant the P1 is used for intensity
    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
)

```

```

RLoadPanelPolyLine = (
    long LoadCaseId           load case index (0 < LoadCaseId ≤ AxisVMLoadCases.Count)
    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 local x axis of polyline [kNm/m]
    EDistributionType DistributionType type of distributed load (global / local / projected)
    double Nx, Ny, Nz          normal plane of the load
)

```



Functions

Use **Add...** functions to create a new load based on a load record. Call **GetLoad** to read load data. Call **SetLoad** to overwrite write load data (load case index and element index cannot be changed) . 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 (GetLoad)

Delphi:

1. Call the GetLoad function of the interface.
2. Call the **SafeArrayAccessssData** COM function to get the pointer to the array data.

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

3.Move the bytes from the safe array to the destination record.

4. After usage free the pointer by calling **SafeArrayUnAccessssData**.

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

5. When you don't need it any longer, free the SAFEARRAY by calling **SafeArrayDestroy**.

Example code :

VB.NET:

1. Call the GetLoad function of the interface.

```
ReturnValue = AxLoads.GetLoad(LoadIndex, LoadData)  
where:  
Dim LoadData As Array = Nothing
```

2. Convert array to byte and typecast the array to the appropriate load record (struct) using functions ConvertToBytes and ByteArrayToStruct

```
Dim Load As RLoadRibMemberDistributed = ByteArrayToStruct(ConvertToBytes(LoadData),  
GetType(RLoadRibMemberDistributed))
```

used VB functions:

```
Private Function ConvertToBytes(ByVal MyArray As Array)  
    Dim AllBytes As New List(Of Byte)()  
    For Each obj As Object In MyArray  
        AllBytes.Add(obj)  
    Next  
    Return AllBytes.ToArray()  
End Function  
  
Private Function ByteArrayToStruct(ByVal btData As Byte(), ByVal StructType As Type)  
    Dim iStructSize As Integer = System.Runtime.InteropServices.Marshal.SizeOf(StructType)  
    If iStructSize <> btData.Length Then  
        Return Nothing  
    End If  
    Dim Buffer As IntPtr = System.Runtime.InteropServices.Marshal.AllocHGlobal(iStructSize)  
    System.Runtime.InteropServices.Marshal.Copy(btData, 0, Buffer, iStructSize)  
    Dim RetStruct As Object = System.Runtime.InteropServices.Marshal.PtrToStructure(Buffer,  
    StructType)  
    System.Runtime.InteropServices.Marshal.FreeHGlobal(Buffer)  
    Return RetStruct  
End Function
```

VBA (MS Office):

1. Call the GetLoad function of the interface.

```
RetVal = AxLoads.GetLoad(LoadIndex, LoadData)
ArrSize = UBound(LoadData)
```

where:

```
Dim LoadData() As Byte
Dim ArrSize As Long
Dim LoadNodalForce as RLoadNodalForce
```

2. Copy the content from the byte array pointer to the pointer of the load record (struct) using function

CopyMemory

```
CopyMemory ByVal VarPtr(LoadNodalForce), ByVal VarPtr(LoadData(1)), ArrSize
```

CopyMemory function is declared as follow:

```
Public Declare Sub CopyMemory Lib "kernel32" Alias "RtlMoveMemory" _
(Destination As Any, Source As Any, ByVal Length As Long)
```

How to modify load data (SetLoad)

Delphi:

1. Call **SafeArrayCreateEx** COM function to create a one dimensional array of VT_UI1 with the length of the record's SizeOf:

```
SAFEARRAY SafeArrayCreateEx(
    VARTYPE vt
    unsigned long cDims
    SAFEARRAYBOUND* rgsabound
    PVOID pvExtra );
```

*vt = VT_UI1
cDims = 1
rgsabound
unsigned long cElements = sizeof(Record)
long lLbound = 1
ppRecInfo obtained from the previous function*

2. Call **SafeArrayAccesssData** COM function to get the pointer to the array data.

```
HRESULT SafeArrayAccesssData(
    SAFEARRAY* psa
    void HUGEP** ppvData);
```

*psa is the return value of SafeArrayCreateEx
ppvData is a pointer to the array data*

3. Move the bytes from the source record to the safe array.

4. After usage free the pointer by calling **SafeArrayUnAccesssData**.

```
HRESULT SafeArrayUnAccesssData(
    SAFEARRAY* psa);
```

psa same as above

5. Call the [SetLoad](#) method of the interface to write the modified data.

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

VB.NET:

1. Declare variables and set values

```
Dim LoadData As Array = Nothing
Dim Load As RLoadRibMemberDistributed
Load.qz1 = 10
```

2. Convert struct (record) to byte array with function StructToByteArray then convert byte array to array with function ByteArrayToArray

```
LoadData = ByteArrayToArray(StructToByteArray(CObj(Load), GetType(RLoadRibMemberDistributed)))
```

3. Call the SetLoad function of the interface.

```
RetVal = AxLoads.SetLoad(LoadIndex, LoadData)
```

used VB functions:

```
Private Function StructToByteArray(ByVal strData As Object, ByVal StructType As Type)
    Dim iStructSize As Integer = System.Runtime.InteropServices.Marshal.SizeOf(StructType)
    Dim btData As Byte()
    ReDim btData(iStructSize)
    Dim Buffer As IntPtr = System.Runtime.InteropServices.Marshal.AllocHGlobal(iStructSize)
    System.Runtime.InteropServices.Marshal.StructureToPtr(strData, Buffer, False)
    System.Runtime.InteropServices.Marshal.Copy(Buffer, btData, 0, iStructSize)
    System.Runtime.InteropServices.Marshal.FreeHGlobal(Buffer)
    Return btData
End Function

Private Function ByteArrayToArray(ByVal btData As Byte())
    Dim lengths() As Integer = {btData.Length} 'this is for one-dimensional array with i elements
    Dim lowerBounds() As Integer = {1} 'this specifies lower bound=1
    Dim arr As Array = Array.CreateInstance(GetType(Byte), lengths, lowerBounds)
    For i As Long = 1 To btData.Length - 1
        arr.SetValue(btData(i - 1), i)
    Next
    Return arr
End Function
```

VBA (MS Office):

1. Set array size of the byte array and fill the LoadNodalForce load record (struct)

```
Dim LoadNodalForce as RLoadNodalForce
Dim ArrSize As Long
ArrSize = Len(LoadNodalForce)
Dim LoadData() As Byte
ReDim LoadData(1 To ArrSize)
```

2. Copy the content of the load record (struct) to the pointer of the byte array using function CopyMemory

```
CopyMemory ByVal VarPtr(LoadData(1)) ByVal VarPtr(LoadNodalForce), ArrSize
```

3. Call the SetLoad_vb function of the interface.

```
AxisLoads.SetLoad_vb(LoadIndex, LoadData)
```

CopyMemory function is declared as follow:

```
Public Declare Sub CopyMemory Lib "kernel32" Alias "RtlMoveMemory" -
(Destination As Any, Source As Any, ByVal Length As Long)
```

IMPORTANT NOTE:

LoadCaseId and NodeId, LineId, SurfaceId, DomainId, SupportId record fields will not be overwritten by SetLoad so these fields cannot be changed.

Same loads deleted

If the model has two or more loads which meet **all** of the following criterias :

- same type
- applied at the same element (node, beam, domain)
- applied at the same location (start, end position, location, etc.)
- are in the same load case
- each intensity (temperature, deformation, etc.) of their components are the same
- coordinate system is the same (where applicable)
- distribution type is the same (where applicable)
- all the other properties are the same as well eg. Trapezoid, Reference index etc. (where applicable)

then only one of them will be allowed, all the others will be deleted before analysis

Add Nodal loads

long **AddNodalForce** ([i/o] [RLoadNodalForce](#) Data)
Data load data

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

IMPORTANT NOTE: If the added load meets [this](#), then it will be deleted before analysis!

Add Truss loads

long **AddTrussFault** ([i/o] [RLoadTrussFault](#) Data)
Data load data

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

IMPORTANT NOTE: If the added load meets [this](#), then it will be deleted before analysis!

long **AddTrussSelfWeight** ([in] long **Lineld**, [in] long **LoadCaseld**)
Lineld line index (truss)
(0 < Index ≤ [AxisVMLines.Count](#))
LoadCaseld load case index
(0 < Index ≤ [AxisVMLoadCases.Count](#))

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

long **AddTrussStress** ([i/o] [RLoadTrussStress](#) Data)
Data load data

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

IMPORTANT NOTE: If the added load meets [this](#), then it will be deleted before analysis!

long **AddTrussThermal** ([i/o] [RLoadTrussThermal](#) Data)
Data load data

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

IMPORTANT NOTE: If the added load meets [this](#), then it will be deleted before analysis!

Add Beam loads

long **AddBeamConcentrated** ([i/o] [RLoadBeamConcentrated](#) Data)

Data load data

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

IMPORTANT NOTE:

If the added load meets [this](#), then it will be deleted before analysis!

long **AddBeamDistributed** ([i/o] [RLoadBeamDistributed](#) Data)

Data load data

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

IMPORTANT NOTE:

If the added load meets [this](#), then it will be deleted before analysis!

long **AddBeamFault** ([i/o] [RLoadBeamFault](#) Data)

Data load data

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

IMPORTANT NOTE:

If the added load meets [this](#), then it will be deleted before analysis!

long **AddBeamInfluence** ([i/o] [RLoadBeamInfluence](#) Data)

Data load data

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

IMPORTANT NOTE:

If the added load meets [this](#), then it will be deleted before analysis!

long **AddBeamSelfWeight** ([in] long **LinId**, [in] long **LoadCaseId**)

LinId line index (beam)

($0 < Index \leq \text{AxisVMLines.Count}$)

LoadCaseId load case index

($0 < Index \leq \text{AxisVMLoadCases.Count}$)

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

long **AddBeamStress** ([i/o] [RLoadBeamStress](#) Data)

Data load data

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

IMPORTANT NOTE:

If the added load meets [this](#), then it will be deleted before analysis!

long **AddBeamThermal** ([i/o] [RLoadBeamThermal](#) Data)

Data load data

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

IMPORTANT NOTE:

If the added load meets [this](#), then it will be deleted before analysis!

long	AddSurfaceToBeam ([in] AxisVMLines3d * ContourPoly , [in] ELongBoolean AutoFindLines , [in] SAFEARRAY(long)* Linelds , [i/o] RLoadSurfaceToBeam Data)
	<p>ContourPoly closed polygon of the surface load to distribute</p> <p>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.</p> <p>Linelds line indexes 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</p> <p>Data load data</p>
	<p>Distributes a homogenous surface load defined by a closed polygon over line elements. If successful, returns load index, otherwise returns an error code (leNotValidLineTypeForThisLoad, leErrorAddingLoad, errIndexOutOfBounds, errDatabaseNotReady).</p> <p>IMPORTANT NOTE: If the added load meets this, then it will be deleted before analysis!</p>
long	AddSurfaceToBeam_vb (Visual Basic compatible function of AddSurfaceToBeam)
long	AddSurfaceToBeamAssoc ([in] SAFEARRAY(long)* ContourLinelds , [in] ELongBoolean AutoFindLines , [in] SAFEARRAY(long)* Linelds , [i/o] RLoadSurfaceToBeamAssoc Data)
	<p>ContourLinelds line indexes of contour lines of the closed polygon</p> <p>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.</p> <p>Linelds line indexes 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</p> <p>Data load data</p>
	<p>Adds derived 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 an error code (leNotValidLineTypeForThisLoad, leErrorAddingLoad, errIndexOutOfBounds, errDatabaseNotReady).</p> <p>IMPORTANT NOTE: If the added load meets this, then it will be deleted before analysis!</p>
long	AddSurfaceToBeamAssoc_vb (Visual Basic compatible function of AddSurfaceToBeamAssoc)

Add Rib loads

If load is applied to a rib, the point of application of defined loads are transferred, explained [here](#).

long	AddRibConcentrated ([i/o] RLoadRibConcentrated Data)
	<p>Data load data</p>
	<p>Creates a concentrated load on a rib. If successful, returns load index, otherwise returns an error code (leInvalidLineType, errIndexOutOfBounds, errDatabaseNotReady).</p> <p>IMPORTANT NOTE: If the added load meets this, then it will be deleted before analysis!</p>

long	AddRibDistributed ([i/o] RLoadRibDistributed Data)
	Data load data
	Creates a distributed load on a rib. If successful, returns load index, otherwise returns an error code (leInvalidLineType , errIndexOutOfBounds , errDatabaseNotReady).
	IMPORTANT NOTE:
	If the added load meets this , then it will be deleted before analysis!
long	AddRibSelfWeight ([in] long LineId , [in] long LoadCaseId)
	LineId line index (rib) ($0 < Index \leq \text{AxisVMLines.Count}$)
	LoadCaseId load case index ($0 < Index \leq \text{AxisVMLoadCases.Count}$)
	Creates a self weight load for a rib. If successful, returns load index, otherwise returns an error code (leInvalidLineType , errIndexOutOfBounds , errDatabaseNotReady).
long	AddRibThermal ([i/o] RLoadRibThermal Data)
	Data load data
	Creates a thermal load on a rib. If successful, returns load index, otherwise returns an error code (leInvalidLineType , errIndexOutOfBounds).
	IMPORTANT NOTE:
	If the added load meets this , then it will be deleted before analysis!

Add Structural member loads

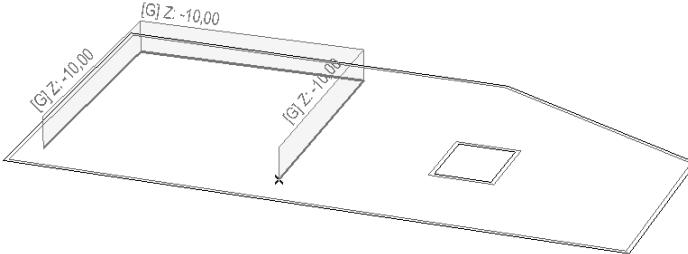
long	AddBeamMemberConcentrated ([i/o] RLoadBeamMemberConcentrated Data)
	Data load data
	Creates a concentrated load on a beam structural member. If successful, returns load index, otherwise returns an error code (loInvalidLoadCaseType , leInvalidLineType , errIndexOutOfBounds , errDatabaseNotReady).
	IMPORTANT NOTE:
	If the added load meets this , then it will be deleted before analysis!
long	AddBeamMemberDistributed ([i/o] RLoadBeamMemberDistributed Data)
	Data load data
	Creates a distributed load on a beam structural member. If successful, returns load index, otherwise returns an error code (loInvalidLoadCaseType , leInvalidLineType , errIndexOutOfBounds , errDatabaseNotReady).
	IMPORTANT NOTE:
	If the added load meets this , then it will be deleted before analysis!
long	AddRibMemberConcentrated ([i/o] RLoadRibMemberConcentrated Data)
	Data load data
	Creates a concentrated load on a rib structural member. If successful, returns load index, otherwise returns an error code (loInvalidLoadCaseType , leInvalidLineType , errIndexOutOfBounds , errDatabaseNotReady).
	IMPORTANT NOTE:
	If the added load meets this , then it will be deleted before analysis!
long	AddRibMemberDistributed ([i/o] RLoadRibMemberDistributed Data)
	Data load data
	Creates a distributed load on a rib structural member. If successful, returns load index, otherwise returns an error code (loInvalidLoadCaseType , leInvalidLineType , errIndexOutOfBounds , errDatabaseNotReady).
	IMPORTANT NOTE:
	If the added load meets this , then it will be deleted before analysis!

Add Surface loads

long	AddSurfaceConcentrated ([i/o] RLoadSurfaceConcentrated Data)
	Data load data
	<i>Creates a concentrated load on a surface element. If successful, returns load index, otherwise returns an error code (errIndexOutOfBounds, errDatabaseNotReady, leErrorAddingLoad).</i>
	IMPORTANT NOTE: <i>If the added load meets this, then it will be deleted before analysis!</i>
long	AddSurfaceDistributed ([i/o] RLoadSurfaceDistributed Data)
	Data load data
	<i>Creates a distributed load on a surface element. If successful, returns load index, otherwise returns an error code (errIndexOutOfBounds, errDatabaseNotReady).</i>
	IMPORTANT NOTE: <i>If the added load meets this, then it will be deleted before analysis!</i>
long	AddSurfaceEdge ([i/o] RLoadSurfaceEdge Data)
	Data load data
	<i>Creates an edge load on a surface element. If successful, returns load index, otherwise returns an error code (errIndexOutOfBounds, errDatabaseNotReady).</i>
	IMPORTANT NOTE: <i>If the added load meets this, then it will be deleted before analysis!</i>
long	AddSurfaceFluid ([i/o] RLoadSurfaceFluid Data)
	Data load data
	<i>Creates a fluid load on a surface element. If successful, returns load index, otherwise returns an error code (leErrorAddingLoad, errIndexOutOfBounds, errDatabaseNotReady).</i>
	IMPORTANT NOTE: <i>If the added load meets this, then it will be deleted before analysis!</i>
long	AddSurfaceSelfWeight ([in] long Surfaceld , [in] long LoadCaseld)
	Surfaceld surface element index ($0 < \text{Index} \leq \text{AxisVMSurfaces}.\text{Count}$)
	LoadCaseld load case index ($0 < \text{Index} \leq \text{AxisVMLoadCases}.\text{Count}$)
	<i>Creates a self weight load on a surface element. If successful, returns load index, otherwise returns an error code (errIndexOutOfBounds, errDatabaseNotReady).</i>
long	AddSurfaceThermal ([i/o] RLoadSurfaceThermal Data)
	Data load data
	<i>Creates a thermal load on a surface element. If successful, returns load index, otherwise returns an error code (errIndexOutOfBounds, errDatabaseNotReady).</i>
	IMPORTANT NOTE: <i>If the added load meets this, then it will be deleted before analysis!</i>

Add Domain loads

long	AddDomainPolyArea ([in] AxisVMLines3d * ContourPoly , [i/o] RLoadDomainPolyArea Data)
	ContourPoly a closed polygon
	Data load data
	<i>Creates a mesh-independent distributed area load on the part of domain defined by closed polygon. If successful, returns load index, otherwise returns an error code (leErrorAddingLoad, errIndexOutOfBounds, errDatabaseNotReady).</i>
	IMPORTANT NOTE: <i>If the added load meets this, then it will be deleted before analysis!</i>

long	AddDomainLinear ([i/o] RLoadDomainLinear Data)
	Data load data
	Creates an associative distributed load over the entire domain. If successful, returns load index, otherwise returns an error code (leErrorAddingLoad , errIndexOutOfBounds , errDatabaseNotReady).
	IMPORTANT NOTE: If the added load meets this , then it will be deleted before analysis!
long	AddDomainConcentrated ([i/o] RLoadDomainConcentrated Data)
	Data load data
	Creates a mesh-independent concentrated load on a domain. If successful, returns load index, otherwise returns an error code (errIndexOutOfBounds , errDatabaseNotReady).
	IMPORTANT NOTE: If the added load meets this , then it will be deleted before analysis!
long	AddDomainConstant ([i/o] RLoadDomainConstant Data)
	Data load data
	Creates a distributed load on a domain. If successful, returns load index, otherwise returns an error code (errIndexOutOfBounds , errDatabaseNotReady).
	IMPORTANT NOTE: If the added load meets this , then it will be deleted before analysis!
long	AddDomainFluid ([i/o] RLoadDomainFluid Data)
	Data load data
	Creates a fluid load on a domain. If successful, returns load index, otherwise returns an error code (leErrorAddingLoad , errIndexOutOfBounds , errDatabaseNotReady).
	IMPORTANT NOTE: If the added load meets this , then it will be deleted before analysis!
long	AddDomainPolyLine ([in] AxisVMLines3d * LoadPoly, [i/o] RLoadDomainPolyLine Data)
	LoadPoly the load polygon
	Data load data
	Creates a mesh-independent polyline load on a part of domain defined by load polygon. If successful, returns load index, otherwise returns an error code (leErrorAddingLoad , errIndexOutOfBounds , errDatabaseNotReady).
	IMPORTANT NOTE: If the added load meets this , then it will be deleted before analysis!
	
long	AddDomainPolyAssoc ([i/o] RLoadDomainPolyAssoc Data)
	Data load data
	Creates an associative line load on a domain edge. Load position changes with the edge of domain. If successful, returns load index, otherwise returns an error code (leErrorAddingLoad , errIndexOutOfBounds , errDatabaseNotReady).
	IMPORTANT NOTE: If the added load meets this , then it will be deleted before analysis!

long	AddDomainSelfWeight ([in] long DomainId , [in] long LoadCaseId)
	DomainId domain index ($0 < \text{Index} \leq \text{AxisVMDomains.Count}$)
	LoadCaseId load case index ($0 < \text{Index} \leq \text{AxisVMLoadCases.Count}$)
<i>Creates a self weight on a domain. If successful, returns load index, otherwise returns an error code (errIndexOutOfBounds, errDatabaseNotReady).</i>	
long	AddDomainThermal ([i/o] RLoadDomainThermal Data)
	Data load data
<i>Creates a thermal load on a domain. If successful, returns load index, otherwise returns an error code (errIndexOutOfBounds, errDatabaseNotReady).</i>	
IMPORTANT NOTE: If the added load meets this , then it will be deleted before analysis!	

Add Support loads

long	AddSupportDisplacement ([i/o] RLoadSupportDisplacement Data)
	Data load data
<i>Creates a support displacement. If successful, returns load index, otherwise returns an error code (errIndexOutOfBounds, errDatabaseNotReady).</i>	
IMPORTANT NOTE: If the added load meets this , then it will be deleted before analysis!	

Add Dynamic loads

long	AddDynamic ([i/o] RLoadDynamic * Data)
	Data Dynamic load parameters
<i>Adds dynamic load. If successful, returns Index, otherwise returns an error code (errIndexOutOfBounds, errDatabaseNotReady, loeReferenceIndexOutOfBounds, loeInvalidLoadCaseType, loeNotDynamicLoadCase, leLoadCaseIndexOutOfBounds, leInvalidLoadType or loeDYNmoduleNotAvailable).</i>	
IMPORTANT NOTE: If the added load meets this , then it will be deleted before analysis!	

Add Load panel loads

long	AddLoadPanelPolyArea ([in] AxisVMLines3d * ContourPoly , [i/o] RLoadPanelPolyArea Data)
	ContourPoly a closed polygon
	Data load data
<i>Creates a mesh-independent distributed area load defined by closed polygon on the part of load panel. If successful, returns load index, otherwise returns an error code (see EGeneralError or ELoadsError).</i>	
IMPORTANT NOTE: If the added load meets this , then it will be deleted before analysis!	
long	AddLoadPanelConcentrated ([i/o] RLoadPanelConcentrated * Data)
	Data parameters of concentrated load on a load panel
<i>Adds concentrated load on a load panel. If successful, returns Index, otherwise returns an error code (see EGeneralError or ELoadsError).</i>	
IMPORTANT NOTE: If the added load meets this , then it will be deleted before analysis!	

long **AddLoadPanelLinear** ([i/o] [RLoadPanelLinear](#) * **Data**)

Data parameters of linear load on a load panel

Adds linear load on a load panel. If successful, returns Index, otherwise returns an error code (see [EGeneralError](#) or [ELoadsError](#)).

IMPORTANT NOTE: If the added load meets [this](#), then it will be deleted before analysis!

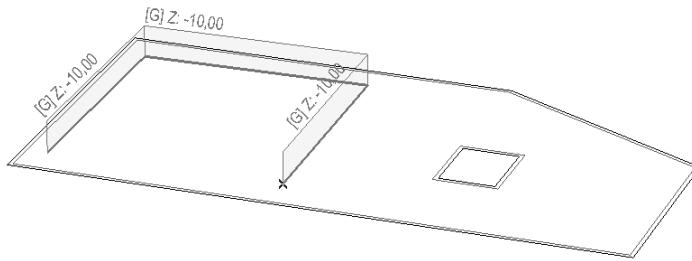
long **AddLoadPolyLine** ([in] [AxisVMLines3d](#) * **LoadPoly**, [i/o] [RLoadPanelPolyLine](#) **Data**)

LoadPoly the load polygon

Data load data

Creates a mesh-independent polyline load defined by load polygon on a part of load panel. If successful, returns load index, otherwise returns an error code (see [EGeneralError](#) or [ELoadsError](#)).

IMPORTANT NOTE: If the added load meets [this](#), then it will be deleted before analysis!



Other functions

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

Index index of the derived surface load

It converts derived surface load over beams/ribs/trusses.

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

long **ConvertSelectedDerivedSurfaceLoad**

It converts selected derived surface load over beams/ribs/trusses.

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

long **CreateSnowLoadOnLoadPanels** ([in] [ERoofType](#) **RoofType**,

[in] SAFEARRAY(long) **LoadPanelIDs**)

RoofType Load panel as roof of type, only rtBarrel and others

LoadPanelIDs Load panel indexes

Generates snow loads and loadcases on specified load panels. Can generate If successful, index of the first snow load case, otherwise returns an error code ([errIndexOutOfBounds](#), [errDatabaseNotReady](#), [loeSWGmoduleNotAvailable](#), [loeLoadPanelIndexListEmpty](#) or [loeNoSnowLoadCase](#)).

long **CreateStandardSeismicLoads**

Warning! This function has become obsolete, was superseded by [CreateStandardSeismicLoads_V153](#)

Recreates the sesimic loads and load cases using vibration results, seismic parameters and the spectrum. (See [IAxisVMSpectrum](#)). If successful, returns index of the first seismic load case, otherwise returns an error code ([errDatabaseNotReady](#), [loeSE1moduleNotAvailable](#), [loeNoSeismicLoadCase](#) or [loeThereAreNoSeismicStoreys](#)).

long **CreateStandardSeismicLoads_V153** ([in] long **GroupID**)

GroupID seismic group ID

Recreates the sesimic loads and load cases using vibration results, seismic parameters and the spectrum (See [IAxisVMSpectrum](#)), corresponding to seismic group GroupID. The other seismic groups remain unaffected. If successful, returns index of the first seismic load case, otherwise returns an error code ([errDatabaseNotReady](#), [loeSE1moduleNotAvailable](#), [loeNoSeismicLoadCase](#) , [lcaeSeismicInvalidGroupID](#), [loeThereAreNoSeismicStoreys](#)).

long **CreateStandardPushOverLoads**

Creates pushover loads using the spectrum. It will delete unused load cases. If successful, returns index of the first pushover load case, otherwise returns an error code ([errDatabaseNotReady](#), [loeSE2moduleNotAvailable](#), [loeNoPushOverLoadCase](#) or [loeErrorCreatingPushOverLoads](#)).

long **CreateWindLoadOnLoadPanels** ([in] SAFEARRAY(long) **LoadPanelIDs**)

LoadPanelIDs Load panel indexes

Generates wind loads and loadcases on specified load panels. If successful, returns Index, otherwise returns an error code ([errIndexOutOfBounds](#), [errDatabaseNotReady](#), [loeSWGmoduleNotAvailable](#), [loeLoadPanelIndexListEmpty](#) or [loeNoWindLoadCase](#)).

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

long **DeleteSnowLoadFromLoadPanels** ([in] SAFEARRAY(long) **LoadPanelIDs**)

LoadPanelIDs Load panel indexes

*Delete snow loads from the load panels in array.
If successful, returns Index, otherwise returns an error code ([errIndexOutOfBounds](#),
[errDatabaseNotReady](#) or [loeLoadPanelIndexIsEmpty](#)).*

long **DeleteSnowLoadFromAllLoadPanels**

*Delete snow loads from all load panels where it was applied.
If successful, returns Index, otherwise returns an error code ([errDatabaseNotReady](#)).*

long	DeleteWindLoadFromLoadPanels ([in] SAFEARRAY(long) LoadPanelIDs)	
	LoadPanelIDs Load panel indexes	
	<i>Delete wind loads from the load panels in array.</i>	
	<i>If successful, returns Index, otherwise returns an error code (errIndexOutOfBounds, errDatabaseNotReady or JoeLoadPanelIndexIsEmpty).</i>	
long	DeleteWindLoadFromAllLoadPanels	
	<i>Delete wind loads from all load panels where it was applied.</i>	
	<i>If successful, returns Index, otherwise returns an error code (errDatabaseNotReady).</i>	
long	GetDomains_Surfaces_LoadPanels ([in] long Index , [out] SAFEARRAY(long)* DomainIds , [out] SAFEARRAY(long)* SurfacingIds , [out] SAFEARRAY(long)* LoadPanelIDs)	
	Index load index ($0 < Index \leq Count$)	
	DomainIds index of domains where load is applied	
	SurfacingIds index of surfaces where load is applied	
	LoadPanelIDs index of load panels where load is applied	
	<i>Get indexes of domains, surfaces and load panels where load is applied. Can be used with load types ItDomainPolyAssoc and ItDomainPolyArea. If successful, returns index, otherwise returns an error code (leInvalidLoadType, errIndexOutOfBounds, errDatabaseNotReady)</i>	
long	GetDomainPolyLineItems ([in] long Index , [out] SAFEARRAY(RLoadDomainPolyLineItem)* Items)	
	Index load index ($0 < Index \leq Count$)	
	Items items (parts) of the load	
	<i>Get all load data of the load type ItLoadDomainPolyLine. If successful, returns number of items, otherwise returns an error code (leInvalidLoadType, errIndexOutOfBounds, errDatabaseNotReady)</i>	
long	GetLoadPanelsOfSnowLoad ([out] SAFEARRAY(long)* PitchedLoadPanelIDs , [out] SAFEARRAY(long)* BarrelLoadPanelIDs)	
	PitchedLoadPanelIDs Load panel indexes on pitched roof	
	BarrelLoadPanelIDs Load panel indexes on barrel roof	
	<i>Get load panel indexes where snow load was applied. If successful, returns total sum of array length of load panels, otherwise returns an error code (errDatabaseNotReady)</i>	
long	GetLoadPanelsOfWindLoad ([out] SAFEARRAY(long)* LoadPanelIDs)	
	LoadPanelIDs Load panel indexes on pitched roof	
	<i>Get load panel indexes where wind load was applied. If successful, returns length of array of load panels, otherwise returns an error code (errDatabaseNotReady)</i>	
long	GetLines ([in] long Index , [out] SAFEARRAY(long)* Linelds)	
	Index load index ($0 < Index \leq Count$)	
	Linelds lines participating in the distribution process according to AxisVMLines	
	<i>(only for ItSurfaceToBeam or ItSurfaceToBeamAssoc) Obtains the line indexes participating in the distribution process. If successful, returns Index, otherwise returns an error code (leInvalidLoadType, errIndexOutOfBounds, errDatabaseNotReady)</i>	
long	GetLoad ([in] long Index , [out] SAFEARRAY* LoadData)	
	Index load index ($0 < Index \leq Count$)	
	LoadData a pointer to the load record in SAFEARRAY format	
	<i>Reads a load record. If successful, returns Index, otherwise returns an error code (errIndexOutOfBounds, errDatabaseNotReady, or positive values).</i>	

long	GetPoly ([in] long Index , [out] AxisVMLines3d* Poly)	
	Index	<i>load index ($0 < Index \leq Count$)</i>
	Poly	<i>the load polygon. For ItDomainPolyArea and ItSurfaceToBeam types it is a closed polygon, for ItLoadDomainPolyLine it is a polyline.</i>
	<i>Reads the load polygon of load types: ItDomainPolyArea, ItLoadDomainPolyLine and ItSurfaceToBeam types only. If successful, returns Index, otherwise returns an error code (leInvalidLoadType, errIndexOutOfBounds, errDatabaseNotReady)</i>	
long	SetLines ([in] long Index , [in] SAFEARRAY(long)* Linelds)	
	Index	<i>load index ($0 < Index \leq Count$)</i>
	Linelds	<i>lines participating in the distribution process according to AxisVMLines (only for ItSurfaceToBeam or ItSurfaceToBeamAssoc) Set the line indexes participating in the distribution process. If successful, returns Index, otherwise returns an error code (leInvalidLoadType, leErrorSettingLines, errIndexOutOfBounds, errDatabaseNotReady)</i>
long	SetLines_vb (Visual Basic compatible function of SetLines)	
long	SetLoad ([in] long Index , [in] SAFEARRAY* LoadData)	
	Index	<i>load index ($0 < Index \leq Count$)</i>
	LoadData	<i>a pointer to the load record in SAFEARRAY format</i>
	<i>Sets a load record. Does not change the indexes of the elements. If successful, returns Index, otherwise returns an error code (can be positive). See ELoadsError.</i>	
long	SetLoad_vb (Visual Basic compatible function of SetLoad)	
long	SetPoly ([in] long Index , [in] AxisVMLines3d* Poly)	
	Index	<i>load index ($0 < Index \leq Count$)</i>
	Poly	<i>the load polygon. For ItDomainPolyArea and ItSurfaceToBeam types it is a closed polygon, for ItLoadDomainPolyLine it is a polyline.</i>
	<i>Set the load polygon only for ItDomainPolyArea, ItLoadDomainPolyLine, ItSurfaceToBeam types. If successful, returns Index, otherwise returns an error code (leInvalidLoadType, leErrorSettingPoly, errIndexOutOfBounds, errDatabaseNotReady)</i>	

Properties

long	Count <i>Get number of loads in the model</i>
long	ElementId [long Index] <i>Get index of the element associated to a load (node, line, member, surface, domain, etc.).</i>
long	LoadCaseId [long Index] • <i>Get or set load case index of a load</i>
ELoadType	LoadType [long Index] <i>Get type of a load.</i>

IAxisVLogicalParts



Interface used to access logical parts in the model. See Parts in AxisVM.

Enumerated types

```
enum EArchitecturalLineElementType = {
    aletColumn = 0,      Architectural type is column, only for lines and structural members.
    aletBeam = 1,        Architectural type is beam, only for lines and structural members.
    aletDiagonal = 2 }   Architectural type is diagonal, only for lines and structural members.
                        Architectural type of line element

enum EDomainElementType = {
    detMembrane =       Membrane type of domains
    0,
    detPlate = 1,       Plate type of domains
    detShell = 2 }       Shell type of domains
                        Structural types of domains

enum EArchitecturalDomainElementType = {
    adetWall= 0,         Architectural type is wall, only for domains
    adetSlab = 1,        Architectural type is slab, only for domains
    adetRamp = 2 }        Architectural type is ramp (other), only for domains
                        Architectural types of domains
```

Records / structures

REnabledLogicalParts = (
<u>ELongBoolean</u> By_Material	Enable logical parts based on material
<u>ELongBoolean</u> By_CrossSection	Enable logical parts based on cross-section
<u>ELongBoolean</u> By_CrossSection_LineType	Enable logical parts based on cross-section and line type
<u>ELongBoolean</u> By_CrossSection_ArchitecturalLineElementType	Enable logical parts based on cross-section and architectural line type
<u>ELongBoolean</u> By_DomainType_Thickness	Enable logical parts based on domain type and thickness
<u>ELongBoolean</u> By_ArchitecturalDomainElementType_Thickness	Enable logical parts based on architectural domain type and thickness
<u>ELongBoolean</u> By_Stories	Enable logical parts based on stories
<u>ELongBoolean</u> By_StructuralGridLines)	Enable logical parts based on structural grid lines

Error codes

```
enum ELogicalPartsError: = {
    lpeMaterialIdOutOfBounds = -100001          material index is invalid
    lpeCrossSectionIdOutOfBounds = -100002        cross-section index is invalid
    lpeInvalidLineType = -100003                  line type is invalid
    lpeNotFound = -100004                         nothing found based on search parameters
    lpeStoreyIdOutOfBounds = -100005              storey index is invalid
    lpeStructuralGridLineUIDOutOfBounds = -100006 } Structural grid line unique index is invalid
```

Functions

```
long GetBy_ArchitecturalDomainElementType_Thickness (
    [in] EArchitecturalDomainElementType ArchitecturalDomainElementType, [in] Double Thickness,
    [in] ELongBoolean SelectParts, [in] ESelectMode SelectMode,
    [out] SAFEARRAY(RPartItem)* PartItems)

ArchitecturalDomainElementType  Architectural types of domains
                                Thickness  thickness of the searched domains
                                SelectParts if lbTrue then found parts will be selected also
                                SelectMode Considered if SelectParts =lbTrue, selection mode
                                PartItems custom part items

Get part items based on arch. domain type and thickness. Returns PartUID if successful, otherwise
returns an error code (errDatabaseNotReady, lpeNotFound)
```

NOTE: If SelectParts = lbTrue the call [Refresh](#) function afterwards if not called between functions [BeginUpdate](#) and [EndUpdate](#)

long **GetBy_ArchitecturalDomainElementType_Thickness_Storey** (
[in] [EArchitecturalDomainElementType](#) **ArchitecturalDomainElementType**, [in] Double **Thickness**,
[in] long **StoreyId**, [in] [EBoolean](#) **SelectParts**, [in] [ESelectMode](#) **SelectMode**,
[out] SAFEARRAY([RPartItem](#))* **PartItems**)

ArchitecturalDomainElementType *Architectural types of domains*
Thickness *thickness of the searched domains*
StoreyId *storey index*
SelectParts *if lbTrue then found parts will be selected also*
SelectMode *Considered if SelectParts =lbTrue, selection mode*
PartItems *custom part items*

Get part items based on arch. domain type, thickness and storey. Returns number of part items if successful, otherwise returns an error code ([errDatabaseNotReady](#), [IpeNotFound](#) or [IpeStoreyIdOutOfBounds](#))

NOTE: If SelectParts = lbTrue the call [Refresh](#) function afterwards if not called between functions [BeginUpdate](#) and [EndUpdate](#)

long **GetBy_CrossSection** ([in] long **CrossSectionId**, [in] long **CrossSectionId2**,
[in] [EBoolean](#) **SelectParts**,
[in] [ESelectMode](#) **SelectMode**, [out] SAFEARRAY([RPartItem](#))* **PartItems**)

CrossSectionId *cross-section index of cross-section at the beginning of the element*
CrossSectionId2 *cross-section index of cross-section at the end of the element*
SelectParts *if lbTrue then found parts will be selected also*
SelectMode *Considered if SelectParts =lbTrue, selection mode*
PartItems *custom part items*

Get part items based on cross-section. Returns PartUID if successful, otherwise returns an error code ([errDatabaseNotReady](#), [IpeNotFound](#) or [IpeCrossSectionIdOutOfBounds](#))

NOTE: If SelectParts = lbTrue the call [Refresh](#) function afterwards if not called between functions [BeginUpdate](#) and [EndUpdate](#)

long **GetBy_CrossSection_ArchitecturalLineElementType** ([in] long **CrossSectionId**,
[in] long **CrossSectionId2**,
[in] [EArchitecturalLineElementType](#) **ArchitecturalLineElementType**, [in] [EBoolean](#) **SelectParts**,
[in] [ESelectMode](#) **SelectMode**, [out] SAFEARRAY([RPartItem](#))* **PartItems**)

CrossSectionId *cross-section index of cross-section at the beginning of the element*
CrossSectionId2 *cross-section index of cross-section at the end of the element*
ArchitecturalLineElementType *Architectural type of line element*
SelectParts *if lbTrue then found parts will be selected also*
SelectMode *Considered if SelectParts =lbTrue, selection mode*
PartItems *custom part items*

Get part items based on cross-section and arch. type of line. Returns PartUID if successful, otherwise returns an error code ([errDatabaseNotReady](#), [IpeNotFound](#), [IpeCrossSectionIdOutOfBounds](#))

NOTE: If SelectParts = lbTrue the call [Refresh](#) function afterwards if not called between functions [BeginUpdate](#) and [EndUpdate](#)

long	GetBy_CrossSection_ArchitecturalLineElementType_Storey ([in] long CrossSectionId , [in] long CrossSectionId2 , [in] EArchitecturalLineElementType ArchitecturalLineElementType , [in] long StoreyId , [in] ELongBoolean SelectParts , [in] ESelectMode SelectMode , [out] SAFEARRAY(RPartItem)* PartItems)
	CrossSectionId cross-section index of cross-section at the beginning of the element
	CrossSectionId2 cross-section index of cross-section at the end of the element
	ArchitecturalLineElementType Architectural type of line element
	StoreyId storey index
	SelectParts if lbTrue then found parts will be selected also
	SelectMode Considered if SelectParts =lbTrue, selection mode
	PartItems custom part items
	Get part items based on cross-section, arch. type of line and storey. Returns PartUID if successful, otherwise returns an error code (errDatabaseNotReady , IpeNotFound , IpeCrossSectionIdOutOfBounds)
	<i>NOTE: If SelectParts = lbTrue the call Refresh function afterwards if not called between functions BeginUpdate and EndUpdate</i>
long	GetBy_CrossSection_LineType ([in] long CrossSectionId , [in] long CrossSectionId2 , [in] ELineType LineType , [in] ELongBoolean SelectParts , [in] ESelectMode SelectMode , [out] SAFEARRAY(RPartItem)* PartItems)
	CrossSectionId cross-section index of cross-section at the beginning of the element
	CrossSectionId2 cross-section index of cross-section at the end of the element
	LineType Structural type of the line
	SelectParts if lbTrue then found parts will be selected also
	SelectMode Considered if SelectParts =lbTrue, selection mode
	PartItems custom part items
	Get part items based on cross-section and line type. Returns PartUID if successful, otherwise returns an error code (errDatabaseNotReady , IpeNotFound , IpeCrossSectionIdOutOfBounds or IpeInvalidLineType)
	<i>NOTE: If SelectParts = lbTrue the call Refresh function afterwards if not called between functions BeginUpdate and EndUpdate</i>
long	GetBy_CrossSection_LineType_Storey ([in] long CrossSectionId , [in] long CrossSectionId2 , [in] ELineType LineType , [in] long StoreyId , [in] ELongBoolean SelectParts , [in] ESelectMode SelectMode , [out] SAFEARRAY(RPartItem)* PartItems)
	CrossSectionId cross-section index of cross-section at the beginning of the element
	CrossSectionId2 cross-section index of cross-section at the end of the element
	LineType Structural type of the line
	StoreyId storey index
	SelectParts if lbTrue then found parts will be selected also
	SelectMode Considered if SelectParts =lbTrue, selection mode
	PartItems custom part items
	Get part items based on cross-section, line type and storey. Returns PartUID if successful, otherwise returns an error code (errDatabaseNotReady , IpeNotFound , IpeCrossSectionIdOutOfBounds or IpeInvalidLineType)
	<i>NOTE: If SelectParts = lbTrue the call Refresh function afterwards if not called between functions BeginUpdate and EndUpdate</i>

long	GetBy_DomainElementType_Thickness ([in] EDomainElementType DomainElementType, [in] Double Thickness, [in] EBoolean SelectParts, [in] ESelectionMode SelectMode, [out] SAFEARRAY(RPartItem)* PartItems)
	<p>DomainElementType Structural types of domains Thickness thickness of the searched domains SelectParts if <i>lbTrue</i> then found parts will be selected also SelectMode Considered if <i>SelectParts</i> =<i>lbTrue</i>, selection mode PartItems custom part items</p>
	<i>Get part items based on struct. type of domain and thickness. Returns number of part items if successful, otherwise returns an error code (errDatabaseNotReady, IpeNotFound)</i>
	<i>NOTE: If <i>SelectParts</i> = <i>lbTrue</i> the call Refresh function afterwards if not called between functions BeginUpdate and EndUpdate</i>
long	GetBy_DomainElementType_Thickness_Storey ([in] EDomainElementType DomainElementType, [in] Double Thickness, [in] long StoreyId, [in] EBoolean SelectParts, [in] ESelectionMode SelectMode, [out] SAFEARRAY(RPartItem)* PartItems)
	<p>DomainElementType Structural types of domains Thickness thickness of the searched domains StoreyId storey index SelectParts if <i>lbTrue</i> then found parts will be selected also SelectMode Considered if <i>SelectParts</i> =<i>lbTrue</i>, selection mode PartItems custom part items</p>
	<i>Get part items based on struct. type of domain, thickness and storey. Returns PartUID if successful, otherwise returns an error code (errDatabaseNotReady, IpeNotFound or IpeStoreyIdOutOfBounds)</i>
	<i>NOTE: If <i>SelectParts</i> = <i>lbTrue</i> the call Refresh function afterwards if not called between functions BeginUpdate and EndUpdate</i>
long	GetBy_Material ([in] long MaterialId, [in] EBoolean SelectParts, [in] ESelectionMode SelectMode, [out] SAFEARRAY(RPartItem)* PartItems)
	<p>MaterialId index of the material SelectParts if <i>lbTrue</i> then found parts will be selected also SelectMode Considered if <i>SelectParts</i> =<i>lbTrue</i>, selection mode PartItems custom part items</p>
	<i>Get part items based on material. Returns PartUID if successful, otherwise returns an error code (errDatabaseNotReady, IpeNotFound, IpeMaterialIdOutOfBounds)</i>
	<i>NOTE: If <i>SelectParts</i> = <i>lbTrue</i> the call Refresh function afterwards if not called between functions BeginUpdate and EndUpdate</i>
long	GetBy_StructuralGridLineUID ([in] long StructuralGridLineUID, [in] EBoolean SelectParts, [in] ESelectionMode SelectMode, [i/o] SAFEARRAY(RPartItem)* PartItems)
	<p>StructuralGridLineUID unique index of the structural grid's line SelectParts if <i>lbTrue</i> then found parts will be selected also SelectMode Considered if <i>SelectParts</i> =<i>lbTrue</i>, selection mode PartItems custom part items</p>
	<i>Get part items based on StructuralGridUID. Returns PartUID if successful, otherwise returns an error code (errDatabaseNotReady, IpeNotFound, IpeMaterialIdOutOfBounds)</i>
long	GetPartItemsByUID ([in] long PartUID, [i/o] SAFEARRAY(RPartItem)* PartItems)
	<p>PartUID unique index of the part PartItems custom part items</p>
	<i>Get logical part items by PartUID. Returns PartUID if successful, otherwise returns an error code (errDatabaseNotReady)</i>

long	GetEnabledLogicalParts ([i/o] REnabledLogicalParts EnabledLogicalParts)
	EnabledLogicalParts <i>Enabled logical parts</i> Get enabled logical pars switch. Returns 1 if successful, otherwise returns an error code (errDatabaseNotReady)
long	SetEnabledLogicalParts ([i/o] REnabledLogicalParts EnabledLogicalParts)
	EnabledLogicalParts <i>Enabled logical parts</i> Set enabled logical pars switch. Returns 1 if successful, otherwise returns an error code (errDatabaseNotReady)
long	SaveDefaultEnabledLogicalParts
	Save selected enabled logical parts switch as default. Returns 1 if successful, otherwise returns an error code (errDatabaseNotReady)

Properties

ELongBoolean	IsLogicalPart [long PartUID] • <i>Is lbTrue if it is a logical part</i> PartUID <i>Unique index of the logical part</i>
BSTR	Name [long PartUID] • <i>Gets the name of the logical part</i> PartUID <i>Unique index of the logical part</i>
BSTR	FullName [long PartUID] • <i>Gets the full name of the logical part</i> PartUID <i>Unique index of the logical part</i>

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_Alfax = -100008	$\alpha_x < 0$
	meNegative_Alfty = -100009	$\alpha_y < 0$
	meNegative_Alfaz = -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
	meNonPositive_Fu = -100019	$f_{yd} \leq 0$ NEN steel
	meNonPositive_Fy40 = -100020	$f_u \leq 0$ EC, I, DIN, SIA steel
	meNonPositive_Fu40 = -100021	$f_{yt} \leq 0$ NEN steel
	meNonPositive_R = -100022	$f_{y*} \leq 0$ EC, I, DIN, SIA steel
	meNonPositive_Rc = -100023	$f_{yd*} \leq 0$ NEN steel
	meNonPositive_SigmabH = -100024	$f_{u*} \leq 0$ EC, I, DIN, SIA steel
	meNonPositive_SigmahH = -100025	$f_{yt*} \leq 0$ NEN steel
	meNonPositive_Fit = -100026	$R \leq 0$ STAS steel
	meNonPositive_Fck = -100027	$R_c \leq 0$ STAS steel
	meNonPositive_GammaC = -100028	$\sigma_{bh} \leq 0$ MSz concrete
	meNonPositive_Alfacc = -100029;	$R_{bc} \leq 0$ STAS concrete
	meNonPositive_Fck_cube = -100030	$\sigma_{hh} \leq 0$ MSz concrete
	meInvalid_MaterialType = -100031	$R_{bi} \leq 0$ STAS concrete
	meInvalid_NationalDesignCode = -100032	$\phi \leq 0$ STAS, NEN concrete
	meNameAlreadyExists = -100033	$\Phi_t \leq 0$ EC, I, DIN, SIA concrete
	meNonPositive_E005 = -100034,	$f_{ck, cube} \leq 0$ DIN concrete
	meNonPositive_Gmean = -100035,	material type is invalid
	meNonPositive_fmk = -100036,	national design code is invalid
	meNonPositive_ft0k = -100037,	
	meNonPositive_ft90k = -100038,	
	meNonPositive_fc0k = -100039,	material name already exists
	meNonPositive_fc90kz = -100040,	
	meNonPositive_fvkz = -100041,	
	meNonPositive_GammaM = -100042,	
	meNonPositive_fc90ky = -100043,	
	meNonPositive_fvky = -100044,	
	meNonPositive_s = -100045,	
	meErrorAdding = -100046 }	

Functions

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

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 FillColour, [in] unsigned long ContourColour, [in] double Ex, [in] double Ey,
[in] double Ez, [in] double Nux, [in] double Nuy, [in] double Nuz, [in] double Alfax, [in] double Alfay,
[in] double Alfaz, [in] double Rho, ...)

NationalDesignName name of the national design code
MaterialDesignName name of the material code
Name name of the material
FillColour fill colour used in rendered view, any number or [EmaterialColour](#), see [Colour](#)
ContourColour outline colour used in rendered view, any number or [EmaterialColour](#), see [Colour](#)
Ex, Ey, Ez Young's modulus of elasticity in local directions [kN/m^2]
Nux, Nuy, Nuz ν Poisson's ratio in local directions $0 \leq \nu \leq 0,5$
Alfax, Alfay, Alfaz α thermal expansion coefficient in local directions [$1/\text{^\circ C}$]
Rho ρ density [kg/m^3]

Adds a new material to the model. If successful, returns the material index, otherwise returns an 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 FillColour,
[in] unsigned long ContourColour, [in] double Ex, [in] double Ey, [in] double Ez,
[in] double Nux, [in] double Nuy, [in] double Nuz, [in] double Alfaz,
[in] double Alfay, [in] double Alfaz, [in] double Rho)

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

Adds an aluminium material to the model.

long **AddAluminium_vb** (Visual Basic compatible function of [AddAluminium](#))

long **AddBrick** ([in] BSTR NationalDesignName,
[in] BSTR MaterialDesignName, [in] BSTR Name, [in] unsigned long FillColour,
[in] unsigned long ContourColour, [in] double Ex, [in] double Ey, [in] double Ez,
[in] double Nux, [in] double Nuy, [in] double Nuz, [in] double Alfaz,
[in] double Alfay, [in] double Alfaz, [in] double Rho)

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

Adds a brick material to the model.

long **AddBrick_vb** (Visual Basic compatible function of [AddBrick](#))

long **AddConcrete_Dutch_NEN** ([in] BSTR NationalDesignName,
[in] BSTR MaterialDesignName, [in] BSTR Name, [in] unsigned long FillColour,
[in] unsigned long ContourColour, [in] double Ex, [in] double Ey, [in] double Ez,
[in] double Nux, [in] double Nuy, [in] double Nuz, [in] double Alfaz,
[in] double Alfay, [in] double Alfaz, [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/m^2]

Fit ϕ creeping factor

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

long **AddConcrete_Dutch_NEN_vb** (Visual Basic compatible function of [AddConcrete_Dutch_NEN](#))

long **AddConcrete_Eurocode** ([in] BSTR NationalDesignName,
[in] BSTR MaterialDesignName, [in] BSTR Name, [in] unsigned long FillColour,
[in] unsigned long ContourColour, [in] double Ex, [in] double Ey, [in] double Ez,
[in] double Nux, [in] double Nuy, [in] double Nuz, [in] double Alfax,
[in] double Alfay, [in] double Alfaz, [in] double Rho, [in] double Fck,
[in] double GammaC, [in] double Alfacc, [in] double Fit)

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

Fck f_{ck} characteristic compressive cylinder strength at 28 days [kN/m^2]
GammaC γ_c safety factor of the concrete
Alfacc 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_Eurocode_vb** (Visual Basic compatible function of [AddConcrete_Eurocode](#))

long **AddConcrete_German_DIN1045_1** ([in] BSTR NationalDesignName,
[in] BSTR MaterialDesignName, [in] BSTR Name, [in] unsigned long FillColour,
[in] unsigned long ContourColour, [in] double Ex, [in] double Ey, [in] double Ez,
[in] double Nux, [in] double Nuy, [in] double Nuz, [in] double Alfax,
[in] double Alfay, [in] double Alfaz, [in] double Rho, [in] double Fck,
[in] double Fck_cube, [in] double GammaC, [in] double Alfacc, [in] double Fit)

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

Fck f_{ck} characteristic compressive cylinder strength at 28 days [kN/m^2]
Fck_cube $f_{ck, cube}$ characteristic compressive cylinder strength of cube
[kN/m^2]
GammaC γ_c safety factor of the concrete
Alfacc 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_German_DIN1045_1_vb** (Visual Basic compatible function of [AddConcrete_German_DIN1045_1](#))

long **AddConcrete_Hungarian_MSz** ([in] BSTR NationalDesignName,
[in] BSTR MaterialDesignName, [in] BSTR Name, [in] unsigned long FillColour,
[in] unsigned long ContourColour, [in] double Ex, [in] double Ey, [in] double Ez,
[in] double Nux, [in] double Nuy, [in] double Nuz, [in] double Alfax,
[in] double Alfay, [in] double Alfaz, [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/m^2]
SigmahH σ_{hH} resistance for tension [kN/m^2]
Fit ϕ creeping factor

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

long **AddConcrete_Hungarian_MSz_vb** (Visual Basic compatible function of [AddConcrete_Hungarian_MSz](#))

long **AddConcrete_Italian** ([in] BSTR NationalDesignName,
[in] BSTR MaterialDesignName, [in] BSTR Name, [in] unsigned long FillColour,
[in] unsigned long ContourColour, [in] double Ex, [in] double Ey, [in] double Ez,
[in] double Nux, [in] double Nuy, [in] double Nuz, [in] double Alfax,
[in] double Alfay, [in] double Alfaz, [in] double Rho, [in] double Fck,
[in] double GammaC, [in] double Alfacc, [in] double Fit)

See [AddConcrete_EuroCode](#)

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

long [AddConcrete_Italian_vb](#) (Visual Basic compatible function of [AddConcrete_Italian](#))

long [AddConcrete_Romanian_STAS](#) ([in] BSTR NationalDesignName,
[in] BSTR MaterialDesignName, [in] BSTR Name, [in] unsigned long FillColour,
[in] unsigned long ContourColour, [in] double Ex, [in] double Ey, [in] double Ez,
[in] double Nux, [in] double Nuy, [in] double Nuz, [in] double Alfax,
[in] double Alfay, [in] double Alfaz, [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/m^2]

SigmahH R_{bi} resistance for tension [kN/m^2]

Fit ϕ creeping factor

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

long [AddConcrete_Romanian_STAS_vb](#) (Visual Basic compatible function of [AddConcrete_Romanian_STAS](#))

long [AddConcrete_Swiss_SIA26x](#) ([in] BSTR NationalDesignName,
[in] BSTR MaterialDesignName, [in] BSTR Name, [in] unsigned long FillColour,
[in] unsigned long ContourColour, [in] double Ex, [in] double Ey, [in] double Ez,
[in] double Nux, [in] double Nuy, [in] double Nuz, [in] double Alfax, [in] double Alfay,
[in] double Alfaz, [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/m^2]

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 [AddConcrete_Swiss_SIA26x_vb](#) (Visual Basic compatible function of [AddConcrete_Swiss_SIA26x](#))

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 successful, returns the material index, otherwise returns an error code.

long [AddFromCatalogFile](#) ([in] BSTR CatalogFileName, [in] BSTR MaterialName)

CatalogFileName name of the catalog file

(e.g.: 'c:\Program Files\AxisVM9\MAT_EC_ENG.mat')

MaterialName name of the material

Adds a material from the catalog file to the model.

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

long	AddFromDialog ([in] SAFEARRAY(ENationalDesignCode) NationalDesignCode , [in] SAFEARRAY(EMaterialType) MaterialTypes)
	<p>NationalDesignCodes <i>List of national design codes (except ndcOther) of whose materials will be shown in the dialog.</i> <i>If nil (null) then materials of current design code are shown.</i></p> <p>MaterialTypes <i>List of material types which will be shown in the dialog</i> <i>If nil (null) then all types of materials are shown.</i></p> <p><i>Opens a dialog for adding material. If successful, returns index of the material, otherwise an error (meErrorAdding).</i></p>
long	AddFromDialog_vb (Visual Basic compatible function of AddFromDialog)
long	AddSteel_Dutch_NEN ([in] BSTR NationalDesignName , [in] BSTR MaterialDesignName , [in] BSTR Name , [in] unsigned long FillColour , [in] unsigned long ContourColour , [in] double Ex , [in] double Ey , [in] double Ez , [in] double Nux , [in] double Nuy , [in] double Nuz , [in] double Alfax , [in] double Alfay , [in] double Alfaz , [in] double Rho , [in] double Fy , [in] double Fu , [in] double Fy40 , [in] double Fu40) <p> Fy f_yd yield stress [kN/m^2] Fu f_{yt} ultimate stress [kN/m^2] Fy40 f_{yd}^* yield stress <i>if thickness is between 40 and 100 mm [kN/m^2]</i> Fu40 f_{yt}^* ultimate stress <i>if thickness is between 40 and 100 mm [kN/m^2]</i> </p> <p><i>Adds a steel according to the Dutch code to the model.</i></p>
long	AddSteel_Dutch_NEN_vb (Visual Basic compatible function of AddSteel_Dutch_NEN)
long	AddSteel_EuroCode ([in] BSTR NationalDesignName , [in] BSTR MaterialDesignName , [in] BSTR Name , [in] unsigned long FillColour , [in] unsigned long ContourColour , [in] double Ex , [in] double Ey , [in] double Ez , [in] double Nux , [in] double Nuy , [in] double Nuz , [in] double Alfax , [in] double Alfay , [in] double Alfaz , [in] double Rho , [in] double Fy , [in] double Fu , [in] double Fy40 , [in] double Fu40) <p><i>For the beginning of the parameter list see Add...</i></p> <p> Fy f_y yield stress [kN/m^2] Fu f_u ultimate stress [kN/m^2] Fy40 f_y^* yield stress <i>if thickness is between 40 and 100 mm [kN/m^2]</i> Fu40 f_u^* ultimate stress <i>if thickness is between 40 and 100 mm [kN/m^2]</i> </p> <p><i>Adds a steel according to Eurocode to the model.</i></p>
long	AddSteel_EuroCode_vb (Visual Basic compatible function of AddSteel_EuroCode)
long	AddSteel_German_DIN1045_1 ([in] BSTR NationalDesignName , [in] BSTR MaterialDesignName , [in] BSTR Name , [in] unsigned long FillColour , [in] unsigned long ContourColour , [in] double Ex , [in] double Ey , [in] double Ez , [in] double Nux , [in] double Nuy , [in] double Nuz , [in] double Alfax , [in] double Alfay , [in] double Alfaz , [in] double Rho , [in] double Fy , [in] double Fu , [in] double Fy40 , [in] double Fu40) <p><i>See AddSteel_EuroCode</i></p> <p><i>Adds a steel according to DIN 1045-1 to the model.</i></p>
long	AddSteel_German_DIN1045_1_vb (Visual Basic compatible function of AddSteel_German_DIN1045_1)

long **AddSteel_Hungarian_MSz** ([in] BSTR NationalDesignName,
[in] BSTR MaterialDesignName, [in] BSTR Name, [in] unsigned long FillColour,
[in] unsigned long ContourColour, [in] double Ex, [in] double Ey, [in] double Ez,
[in] double Nux, [in] double Nuy, [in] double Nuz, [in] double Alfaz,
[in] double Alfay, [in] double Alfaz, [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/m^2]

SigmapH σ_{pH} cylinder-jacket resistance [kN/m^2]

Ry R_y ultimate stress [kN/m^2]

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

long **AddSteel_Hungarian_MSz_vb** (Visual Basic compatible function of [AddSteel_Hungarian_MSz](#))

long **AddSteel_Italian** ([in] BSTR NationalDesignName,
[in] BSTR MaterialDesignName, [in] BSTR Name, [in] unsigned long FillColour,
[in] unsigned long ContourColour, [in] double Ex, [in] double Ey, [in] double Ez,
[in] double Nux, [in] double Nuy, [in] double Nuz, [in] double Alfaz,
[in] double Alfay, [in] double Alfaz, [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_Italian_vb** (Visual Basic compatible function of [AddSteel_Italian](#))

long **AddSteel_Romanian_STAS** ([in] BSTR NationalDesignName,
[in] BSTR MaterialDesignName, [in] BSTR Name, [in] unsigned long FillColour,
[in] unsigned long ContourColour, [in] double Ex, [in] double Ey, [in] double Ez,
[in] double Nux, [in] double Nuy, [in] double Nuz, [in] double Alfaz,
[in] double Alfay, [in] double Alfaz, [in] double Rho, [in] double R, [in] double Rc)

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

R R ultimate stress [kN/m^2]

Rc R_c resistance [kN/m^2]

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

long **AddSteel_Romanian_STAS_vb** (Visual Basic compatible function of [AddSteel_Romanian](#))

long **AddSteel_Swiss_SIA26x** ([in] BSTR NationalDesignName,
[in] BSTR MaterialDesignName, [in] BSTR Name, [in] unsigned long FillColour,
[in] unsigned long ContourColour, [in] double Ex, [in] double Ey, [in] double Ez,
[in] double Nux, [in] double Nuy, [in] double Nuz, [in] double Alfaz,
[in] double Alfay, [in] double Alfaz, [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 **AddSteel_Swiss_SIA26x_vb** (Visual Basic compatible function of [AddSteel_Swiss_SIA26x](#))

long **AddTimber** ([in] BSTR NationalDesignName,
[in] BSTR MaterialDesignName, [in] BSTR Name, [in] unsigned long FillColour,
[in] unsigned long ContourColour, [in] double Ex, [in] double Ey, [in] double Ez,
[in] double Nux, [in] double Nuy, [in] double Nuz, [in] double Alfaz,
[in] double Alfay, [in] double Alfaz, [in] double Rho)

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

Adds a timber material to the model.

long **AddTimber_vb** (Visual Basic compatible function of [AddTimber](#))

long	Clear
<i>Deletes all materials of the model. If successful, returns the number of deleted materials, otherwise an error code.</i>	
long	Delete ([in] long Index)
Index <i>index of the material to delete ($0 < \text{Index} \leq \text{Count}$)</i>	
<i>Deletes a material from the model If successful, returns Index, otherwise an error code.</i>	
long	IndexOf ([in] BSTR Name)
Name <i>name of the material</i>	
<i>Finds a material by its name. If successful, returns the index of the material, otherwise an error code (errDatabaseNotReady or errNotFound).</i>	

Properties

long	Count
<i>Number of materials in the model.</i>	
long	IndexOfUID [long UID] <i>Get index of the material</i>
UID <i>unique index of the material</i>	
<u>AxisVMMaterial</u> *	Item [long Index]
<i>Material interface by index. $1 \leq \text{Index} \leq \text{Count}$.</i>	
long	UID [long Index] <i>Get unique index of the material which remains the same while exists in the model</i>
Index <i>index of the material</i>	

IAxisVMMaterial

Material interface in the model or catalog.

Enumerated types

enum	EMaterialType = {	
	mtOther = 0x0,	<i>Other</i>
	mtSteel = 0x1,	<i>Structural steel</i>
	mtConcrete = 0x2,	<i>Concrete</i>
	mtTimber = 0x3,	<i>Timber</i>
	mtAluminium = 0x4,	<i>Aluminium</i>
	mtBrick = 0x5 }	<i>Brick</i>
	<i>Material type.</i>	
enum	EMaterialColour = {	
	mclOther = 44732672,	<i>Other</i>
	mclOtherContour = 0,	<i>Other contour</i>
	mclSteel = 39134390,	<i>Structural steel</i>
	mclSteelContour = 39124992,	<i>Structural steel contour</i>
	mclConcrete = 44742326,	<i>Concrete</i>
	mclConcreteContour = 33554432,	<i>Concrete contour</i>
	mclTimber = 33582518,	<i>Timber</i>
	mclTimberContour = 33563720,	<i>Timber contour</i>
	mclAluminium = 50313837,	<i>Aluminium</i>
	mclAluminiumContour = 39143496,	<i>Aluminium contour</i>
	mclBrick = 8031416,	<i>Brick</i>
	mclBrickContour = 3424867 }	<i>Brick contour</i>
	<i>Material colour.</i>	
enum	ETimberType = {	
	ttHardwood = 0x0,	<i>Hardwood</i>
	ttGLULAM = 0x1,	<i>GLULAM</i>
	ttLVL = 0x2,	<i>LVL</i> = Laminated Veneer Lumber
	ttSoftWood = 0x3,	<i>SoftWood</i>
	ttOther = 0x4 }	<i>Other</i>
	<i>Types of timber</i>	

Properties

Please note: All properties are read and write if not noted otherwise

double	Alfax • <i>α thermal expansion coefficient in local x direction [1/°C]</i>
double	Alfay • <i>α thermal expansion coefficient in local y direction [1/°C]</i>
double	Alfaz • <i>α thermal expansion coefficient in local z direction [1/°C]</i>
unsigned long	ContourColour • <i>outline(edge) colour used in rendered view (see IAxisVMMaterials)</i>
long	ContourColour_vb • Visual Basic compatible property of ContourColour
double	Ex • <i>Young's modulus of elasticity in local x direction [kN/m²]</i>
double	Ey • <i>Young's modulus of elasticity in local y direction [kN/m²]</i>
double	Ez • <i>Young's modulus of elasticity in local z direction [kN/m²]</i>
unsigned long	FillColour • <i>fill colour used in rendered view (see IAxisVMMaterials) , any number or EmaterialColour</i>
unsigned long	FillColour_vb • Visual Basic compatible property of FillColour
BSTR	MaterialDesignName • <i>name of the material code</i>
	EMaterialType • <i>type of the material</i>

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/m^3]
long	UID Get unique index of the material which remains the same while exists in the model (read only property)

Design parameters of steel materials

EC, I, DIN, SIA, NEN

double	Fu • ultimate stress (EC, DIN, I, SIA: f_u) (NEN: f_{yt}) [kN/m^2]
double	Fu40 • ultimate stress if thickness is between 40 and 100 mm (EC, DIN, I, SIA: f_u^*) (NEN: f_{yt}^*) [kN/m^2]
double	Fy • yield stress (EC, DIN, I, SIA: f_y) (NEN: f_{yd}) [kN/m^2]
double	Fy40 • yield stress if thickness is between 40 and 100 mm (EC, DIN, I, SIA: f_y^*) (NEN: f_{yd}^*) [kN/m^2]

MSz

double	Ry • ultimate stress (R_y) [kN/m^2]
double	SigmaH • resistance (σ_H) [kN/m^2]
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	Alfacc • 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_{hh} , STAS: R_{bi}) [kN/m^2]

Design parameters of timber materials

double	E005 • 5% modulus of elasticity parallel to grain (x) [kN/m^2]
double	fmk • Characteristic bending strength [kN/m^2]
double	ft0k • Characteristic tensile strength parallel to grain [kN/m^2]
double	ft90k • Characteristic tensile strength perpendicular to grain [kN/m^2]
double	fc0k • Characteristic compression strength parallel to grain [kN/m^2]
double	fc90kz • Characteristic compression strength perpendicular to grain (z) [kN/m^2]
double	fvkz • Characteristic shear strength (z) [kN/m^2]
double	fvky • Characteristic shear strength (y) [kN/m^2]

double **fc90ky** • *Characteristic compression strength perpendicular to grain (y) [kN/m²]*
double **GammaM** • *Partial factor of the material*
double **GMean** • *Mean shear modulus [kN/m²]*
double **s** • *Size effect exponent (for LVL materials)*
[ETimberType](#) • *Type of timber*

IAxisVMMathTexts

Text (strings) for generating formulas, formatted tables, embedded images etc. are called MathText. Clients can access MathText(s) of the model through MathTexts property in [IAxisVMMModel](#).

MathText is generally a pair of strings with some properties saved to the axs file and should be updated whenever events requiring updating (view / print of the report) texts are called so all events of [IAxisVMMathTextsEvents](#) must be handled to ensure that the MathText and it's title is always updated when events are triggered.

Error codes

```
enum EMathTextError = {
    mteMathTextUIDAndAPI_NameDontMatch = -100001,
    mteAPI_NameNotFound = -100002,
    mteMathTextUIDnotValid = -100003
}
```

*MathText was added with different API_Name
No math texts found with this API_Name
Math texts with this unique index is not found*

Functions

long	AddToReport ([in] long MathTextUID , [in] BSTR API_Name)	
	MathTextUID <i>Unique index of math text</i>	
	API_Name <i>Name of the COM client, used for identification of the client</i>	
	<i>Add the math text to the end of report. If successful, returns unique index of math text, otherwise an error code (mteMathTextUIDnotValid, mteMathTextUIDAndAPI_NameDontMatch, errDatabaseNotReady).</i>	
long	Delete ([in] long MathTextUID , [in] BSTR API_Name)	
	MathTextUID <i>Unique index of math text</i>	
	API_Name <i>Name of the COM client, used for identification of the client</i>	
	<i>Delete the math text. If successful, returns unique index of math text, otherwise an error code (mteMathTextUIDnotValid, mteMathTextUIDAndAPI_NameDontMatch, errDatabaseNotReady).</i>	
long	GetUIDs ([in] BSTR API_Name , [out] SAFEARRAY(long) UIDs)	
	API_Name <i>Name of the client which is saved to the model file.</i>	
	UIDs <i>Unique indexes of math text which belong to API_Name</i>	
	<i>Get math text UIDs by API_Name . If successful, returns unique index of math text, otherwise an error code (mteAPI_NameNotFound, errDatabaseNotReady).</i>	
long	GetMathText ([in] long MathTextUID , [in] BSTR API_Name , [out] BSTR MathTextTitle , [out] BSTR MathText , [out] ELongBoolean EnableIfNotPresent)	
	MathTextUID <i>Unique index of math text</i>	
	API_Name <i>Name of the COM client, used for identification of the client</i>	
	MathTextTitle <i>Title of math text which is shown in report's table of contents, can be empty and set when ValidMathText event is handled by the client</i>	
	MathText <i>The actual math text, can be empty and set during FillMathText event of this IAxisVMMathTextsEvents. This text is used even if COM client (addon / plugin) which have created the math text is not loaded or not responding.</i>	
	EnableIfNotPresent <i>If lbTrue, the math text is also always listed in the report even if the COM client (addon / plugin) which have created the math text is not loaded or not responding. IAxisVMMathTextsEvents events are not triggered for this MathText. Set to lbFalse if you want to update with events.</i>	
	<i>Get math text data by MathTextUID and API_Name. If successful, returns unique index of math text, otherwise an error code (mteAPI_NameNotFound, errDatabaseNotReady, mteMathTextUIDnotValid, mteMathTextUIDAndAPI_NameDontMatch).</i>	

long	New ([in] BSTR API_Name, [in] BSTR MathTextTitle, [in] BSTR MathText, [in] ELongBoolean EnableIfNotPresent)
	<p>MathTextUID <i>Unique index of math text</i></p> <p>API_Name <i>Name of the COM client, used for identification of the client</i></p> <p>MathTextTitle <i>Title of math text which is shown in report's table of contents, can be empty and set when ValidMathText event is handled by the client</i></p> <p>MathText <i>The actual math text, can be empty and set during FillMathText event of this IAxisVMMathTextsEvents. This text is used even if COM client (addon / plugin) which have created the math text is not loaded or not responding.</i></p> <p>EnableIfNotPresent <i>If lbTrue, the math text is also always listed in the report even if the COM client (addon / plugin) which have created the math text is not loaded or not responding. IAxisVMMathTextsEvents events are not triggered for this MathText. Set to lbFalse if you want to update with events.</i></p>
	<i>Add new math text associated with the API_Name string. If successful, returns unique index of the new math text, otherwise an error code (errDatabaseNotReady, errInvalidName).</i>
long	SetMathText ([in] long MathTextUID, [in] BSTR API_Name, [in] BSTR MathTextTitle, [in] BSTR MathText, [in] ELongBoolean EnableIfNotPresent)
	<p>MathTextUID <i>Unique index of math text</i></p> <p>API_Name <i>Name of the COM client, used for identification of the client</i></p> <p>MathTextTitle <i>Title of math text which is shown in report's table of contents, can be empty and set when ValidMathText event is handled by the client</i></p> <p>MathText <i>The actual math text, can be empty and set during FillMathText event of this IAxisVMMathTextsEvents. This text is used even if COM client (addon / plugin) which have created the math text is not loaded or not responding.</i></p> <p>EnableIfNotPresent <i>If lbTrue, the math text is also always listed in the report even if the COM client (addon / plugin) which have created the math text is not loaded or not responding. IAxisVMMathTextsEvents events are not triggered for this MathText. Set to lbFalse if you want to update with events.</i></p>
	<i>Set math text data by MathTextUID and API_Name. If successful, returns unique index of math text, otherwise an error code (mteAPI_NameNotFound, errDatabaseNotReady, mteMathTextUIDnotValid, mteMathTextUIDandAPI_NameDontMatch).</i>
long	ShowInWindow ([in] long MathTextUID, [in] BSTR API_Name, [in] BSTR Caption, [in] RWindowPosition Position, [in] ELongBoolean Substitution)
	<p>MathTextUID <i>Unique index of math text</i></p> <p>API_Name <i>Name of the COM client, used for identification of the client</i></p> <p>Caption <i>Caption of the form showing the math text</i></p> <p>Position <i>Size and the position of the form showing the math text</i></p> <p>Substitution <i>If lbTrue then the variables in math text are also shown.</i></p>
	<i>Show the math text in a new AxisVM window. If the math text is added to the report, returns unique index of math text, otherwise returns 0 or an error code (mteAPI_NameNotFound, errDatabaseNotReady, mteMathTextUIDnotValid, mteMathTextUIDandAPI_NameDontMatch).</i>

IAxisVMMembers

Structural members (line elements) of the model.

The structural member is a series of line elements with the same local coordination system. Structural members can be defined as truss, beam or rib.

When rib is defined please note that the point of application of defined loads are transferred, explained [here](#).

Error codes

enum EMembersError = {	
mbeEmptyLineList = -100001,	<i>line list is empty</i>
mbePropertyNotValidForThisLineType = -100002,	<i>invalid property for the given line type</i>
mbeNotBeam = -100003,	<i>the element is not a beam</i>
mbeNotRib = -100004,	<i>the element is not a rib</i>
mbeIllegalServiceClassValue = -100005,	<i>service class value of the timber is incorrect</i>
mbeDomainIndexOutOfBounds = -100006,	<i>provided DomainID was incorrect using functions: IAxisVMLine.DefineAsRibWithAutoExcentricity and IAxisVMLine.DefineAsTimberRibWithAutoExcentricity</i>
mbeStoreyIdOutOfBounds = -100007,	<i>storey index invalid</i>
mbeMustBeBeamOrRibOrTuss = -100008,	<i>member must be a beam, rib or truss</i>
mbeInvalidMemberType = -100009,	<i>invalid member type</i>
mbeReinforcementParametersNotExist = -100010,	<i>reinforcement parameters not exist</i>
mbeInvalidColumnRebarsId = -100011,	<i>invalid column rebar index</i>
mbeInvalidConcreteMaterialId = -100012,	<i>invalid concrete material index</i>
mbeInvalidRebarSteelGradeId = -100013,	<i>invalid rebar steel grade index</i>
mbeNotTruss = -100014,	<i>the element is not a truss</i>
mbeInvalidRelease = -100015,	<i>other release error</i>
mbeInvalidFunctionIdOfRelease = -100016,	<i>invalid function index related to the DOF</i>
mbeReleaseInitAndLimitMustBe0 = -100017,	<i>Initial rotational stiffness and moment resistance of the release must be 0</i>
mbeFunctionIdMustBe0 = -100018,	<i>FunctionId of the release must be 0</i>
mbeMembersNotContinuous = -100019,	<i>Members do not compose a continuous member</i>
mbeStartEndCrossSectionTypeIncompatible = -100020,	
mbeInvalidRCCheckingParameters = -100021,	<i>at least one of the RC checking parameters is invalid</i>
mbeRCShrinkageEpsMustBePositive = -100022,	<i>shrinkage strain must be positive</i>
mbeStirrupParametersAreInvalid = -100023,	<i>stirrup parameters are invalid</i>
mbeShearCrackAngleIsInvalid = -100024,	<i>defined shear crack angle is too low/high</i>
mbeInvalidSteelMaterialId = -100025 } ,	<i>invalid steel grade index</i>

Functions

long **Add** ([in] SAFEARRAY(long) **Linelds**)

Linelds Consecutive line indexes defining the structural member.

Defines a new structural member. It inherits its properties from the constituent beam or rib elements. If angle between lines in consecutive order is bigger than specific angle, then it defines member for each line.

If successful, returns number of defined members, otherwise returns an error code ([errDatabaseNotReady](#), [errIndexOutOfBounds](#), [mbeEmptyLineList](#)).

long **Add_vb** (Visual Basic compatible function of [Add](#))

long **AssembleSelectedMembers ()**

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

long **BulkGetMembers** ([in] SAFEARRAY(long) **MemberIds**, [out] SAFEARRAY ([RLineAttr](#)) * **MemberAttrs**)

MemberIds list of member indexes to query. Each index must satisfy: $(1 \leq \text{Index} \leq \text{AxisVMMembers.Count})$

MemberAttrs list of member attribute records.

Queries the properties of several members in one step. If successful, returns number of queried members, otherwise returns an error code ([errDatabaseNotReady](#), [errIndexOutOfBounds](#), [errInternalException](#), [errOutOfMemory](#)).

long **BulkSetMembers** ([in] SAFEARRAY(long) **MemberIds**, [in] SAFEARRAY ([RLineAttr](#)) **MemberAttrs**)

MemberIds list of member indexes to modify. Each index must satisfy: $(1 \leq \text{Index} \leq \text{AxisVMMembers.Count})$

MemberAttrs *list of member attribute records.*

Modifies the properties of several already existing members in one step. If successful, returns number of modified members, otherwise returns an error code ([errDatabaseNotReady](#), [errIndexOutOfBounds](#), [errInternalException](#), [errOutOfMemory](#), [mbeStartEndCrossSectionTypeIncompatible](#), [mbellIllegalServiceClassValue](#), [mbePropertyNotValidForThisLineType](#)) .

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

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

Reverts local x direction. If end node indexes are i and j and local x direction points from i to j ChangeLocalDirection makes it point from j to i . If successful, returns the member index, otherwise returns an error code ([errDatabaseNotReady](#), [errIndexOutOfBounds](#)).

long **Clear**

Deletes all existing structural members. Their constituent line elements are not deleted. If successful, returns zero, otherwise returns an error code ([errDatabaseNotReady](#)).

long	Delete ([in] long Index)
	Index member index ($0 < \text{Index} \leq \text{Count}$)
	<i>Deletes an existing structural member. Breaks apart consecutive lines. Its constituent line elements are not deleted nor the assigned cross-sections and materials. If successful, returns zero, otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBounds).</i>
long	DeleteMesh ([in] long Index)
	Index member index ($0 < \text{Index} \leq \text{Count}$)
	<i>If successful, returns member index, otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBounds)</i>
long	DeleteMeshFromSelectedItems
	<i>If successful, returns number of selected members, otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBounds)</i>
long	DeleteNameOfAllBeams
	<i>Deletes previously added default name for all beams. If successful returns 1, otherwise returns an error code (errDatabaseNotReady).</i>
long	DeleteNameOfAllRibs
	<i>Deletes previously added default name for all ribs. If successful returns 1, otherwise returns an error code (errDatabaseNotReady).</i>
long	DeleteNameOfAllTrusses
	<i>Deletes previously added default name for all trusses. If successful returns 1, otherwise returns an error code (errDatabaseNotReady).</i>
long	DeleteSelected
	<i>Deletes selected structural members. Their constituent line elements are not deleted. If successful, returns zero, otherwise returns an error code (errDatabaseNotReady).</i>
long	GenerateMeshWithParamsOnSelectedItems ([i/o] RMemberMeshParameters * MeshParameters)
	MeshParameters Mesh parameters
	<i>If successful, returns number of selected members, otherwise returns an error code (errDatabaseNotReady)</i>
long	GetContinuousMemberIds ([i/o] SAFEARRAY(long)* MemberIds , [in] double MaxAngleDifferenceX , [in] double MaxAngleDifferenceZ , [out] SAFEARRAY (long) * ContinousMemberIds)
	MemberIds array of member indices
	MaxAngleDifferenceX maximum angle difference between angles of local X axes of adjacent members
	MaxAngleDifferenceZ maximum angle difference between angles of local Z axes of adjacent members
	ContinousMemberIds array of member indices that are continuous
	<i>Get indices of continuous member indices. If successful It returns the number of continuous members, otherwise it returns an error code (errDatabaseNotReady, errIndexOutOfBounds, mbeEmptyLineList, mbeMembersNotContinuous).</i>
long	GetSelectedItemIds ([out] SAFEARRAY (long) * ItemIds)
	ItemIds list of selected members
	<i>If successful, returns the number of selected members otherwise returns an error code</i>
long	GetSelectedColumnIds ([out] SAFEARRAY (long) * MemberIds)
	MemberIds list of selected vertical members
	<i>If successful, returns the number of selected vertical members, otherwise returns an error code(errDatabaseNotReady).</i>

long	GetSelectedBeamIds ([out] SAFEARRAY (long) * MemberIds)
	<p>MemberIds <i>list of selected horizontal members</i> <i>If successful, returns the number of selected horizontal members otherwise returns an error code (errDatabaseNotReady).</i></p>
long	GetSelectedOtherIds ([out] SAFEARRAY (long) * MemberIds)
	<p>MemberIds <i>list of selected neither vertical or horizontal members</i> <i>If successful returns the number of selected neither vertical or horizontal members, otherwise returns an error code (errDatabaseNotReady).</i></p>
long	IndexOf ([in] SAFEARRAY(long) Linelds)
	<p>Linelds <i>Consecutive line indexes defining the structural member. Must contain all line indexes of the member otherwise function returns 0.</i> <i>Retrieves the index of an existing structural member by the line indexes of its constituents. If successful, returns the member index, otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBounds, mbeEmptyLineList [Linelds array is empty]).</i></p>
long	IndexOf _vb (Visual Basic compatible function of IndexOf)
long	RenameSelectedBeams ([in] long NewBase , [in] BSTR FormatStr)
	<p>NewBase <i>Start number for renaming, must be a positive number</i> FormatStr <i>Prefix string of the new name + "_" eg "Beam_"</i> <i>Rename selected members with membertype.of beam.</i> <i>Example: NewBase=100 and FormatStr= 'new_' then names are: 'new100', 'new101',...etc.</i> <i>If successful returns 1, otherwise returns an error code (errDatabaseNotReady).</i></p>
long	RenameSelectedRibs ([in] long NewBase , [in] BSTR FormatStr)
	<p>NewBase <i>Start number for renaming, must be a positive number</i> FormatStr <i>Prefix string of the new name + "_" eg "Rib_"</i> <i>Rename selected members with membertype.of rib</i> <i>Example: NewBase=100 and FormatStr= 'new_' then names are: 'new100', 'new101',...etc.</i> <i>If successful returns 1, otherwise returns an error code (errDatabaseNotReady).</i></p>
long	RenameSelectedTrusses ([in] long NewBase , [in] BSTR FormatStr)
	<p>NewBase <i>Start number for renaming, must be a positive number</i> FormatStr <i>Prefix string of the new name + "_" eg "Truss_"</i> <i>Rename selected members with membertype.of truss</i> <i>Example: NewBase=100 and FormatStr= 'new_' then names are: 'new100', 'new101',...etc.</i> <i>If successful returns 1, otherwise returns an error code (errDatabaseNotReady).</i></p>
long	SelectAll ([in] ELongBoolean Select)
	<p>Select <i>selection state</i> <i>If Select is True, selects all members.</i> <i>If Select is False, deselects all members.</i> <i>If successful, returns the number of selected members, otherwise returns an error code (errDatabaseNotReady).</i> <i>NOTE: Call Refresh function afterwards if not called between functions BeginUpdate and EndUpdate</i></p>
long	SelectAllColumns ([in] ELongBoolean Select)
	<p>Select <i>selection state</i> <i>If Select is True, selects all vertical members.</i> <i>If Select is False, deselects all vertical members.</i> <i>If successful, returns the number of selected vertical members, otherwise returns an error code (errDatabaseNotReady).</i> <i>NOTE: Call Refresh function afterwards if not called between functions BeginUpdate and EndUpdate</i></p>

long	SelectAllBeams ([in] ELongBoolean Select)
	Select selection state
	<i>If Select is True, selects all horizontal members</i>
	<i>If Select is False, deselects all horizontal members</i>
	<i>If successful, returns the number of selected horizontal members, otherwise returns an error code (errDatabaseNotReady).</i>
	NOTE: Call Refresh function afterwards if not called between functions BeginUpdate and EndUpdate
long	SelectAllOthers ([in] ELongBoolean Select)
	Select selection state
	<i>If Select is True, selects all neither vertical or horizontal members).</i>
	<i>If Select is False, deselects all neither vertical or horizontal members .</i>
	<i>If successful, returns the number of selected neither vertical or horizontal members, otherwise returns an error code (errDatabaseNotReady).</i>
	NOTE: Call Refresh function afterwards if not called between functions BeginUpdate and EndUpdate
long	SelectAllColumnsAtStorey ([in] ELongBoolean Select, [in] long StoreyId)
	Select selection state
	StoreyId storey index
	<i>If Select is True, selects all vertical members at storey.</i>
	<i>If Select is False, deselects all vertical members at storey.</i>
	<i>If successful, returns the number of selected vertical members at storey (and above if last top storey) with index StoreyId, otherwise returns an error code (errDatabaseNotReady, mbeStoreyIdOutOfBounds).</i>
	NOTE: Call Refresh function afterwards if not called between functions BeginUpdate and EndUpdate
long	SelectAllBeamsAtStorey ([in] ELongBoolean Select, [in] long StoreyId)
	Select selection state
	StoreyId storey index
	<i>If Select is True, selects all horizontal members.</i>
	<i>If Select is False, deselects all horizontal members.</i>
	<i>If successful, returns the number of selected horizontal members at storey (and above if last top storey) with index StoreyId, otherwise returns an error code (errDatabaseNotReady, mbeStoreyIdOutOfBounds).</i>
	NOTE: Call Refresh function afterwards if not called between functions BeginUpdate and EndUpdate
long	SelectAllOthersAtStorey ([in] ELongBoolean Select, [in] long StoreyId)
	Select selection state
	StoreyId storey index
	<i>If Select is True, selects all neither vertical or horizontal members at storey.</i>
	<i>If Select is False, deselects all neither vertical or horizontal members at storey.</i>
	<i>If successful, returns the number of selected neither vertical or horizontal members at storey (and above if last top storey) with index StoreyId, otherwise returns an error code (errDatabaseNotReady, mbeStoreyIdOutOfBounds).</i>
	NOTE: Call Refresh function afterwards if not called between functions BeginUpdate and EndUpdate

Properties

AxisVMAttachments*	Attachments Get the attachments interface
AxisVMAtributes*	Attributes Get the attributes interface
long	Count Get number of structural members in the model
IAxisVMMember*	Item [long Index] Get structural member interface by index
	Index index of the member
long	IndexOfUID [long UID] Get index of the structural member
	UID unique index of the structural member

<u>EBoolean</u>	LocalX_is_ij [long Index] <i>Is lbTrue if local x axis of the line is in i-j direction</i> Index <i>index of the member</i>
BSTR	Name [long Index] <i>Get name of the member</i> Index <i>index of the member</i>
<u>EBoolean</u>	Selected [long Index] • <i>Get or set the selection status of a structural member</i> Index <i>index of the member</i> <i>NOTE: Call <u>Refresh</u> function afterwards if not called between functions <u>BeginUpdate</u> and <u>EndUpdate</u></i>
long	SelCount <i>Get number of selected structural members in the model</i>
double	StiffnessReduction_A [long Index] • <i>Get or set axial stiffness factor. Cross sections area is multiplied with this value in stiffness matrix. Default value is 1. Used only for atLinearStatic and atLinearVibration analysis <u>types</u>. (only valid for STAS and Eurocode [RO] standards)</i>
double	StiffnessReduction_I [long Index] • <i>Get or set flexural stiffness factor. Cross sections inertia is multiplied with this value in stiffness matrix. Default value is 1. Used only for atLinearStatic and atLinearVibration analysis <u>types</u>. (only valid for STAS and Eurocode [RO] standards)</i>
long	UID [long Index] <i>Get unique index of the member which remains the same while exists in the model</i> Index <i>index of the member</i>

IAxisVMMember

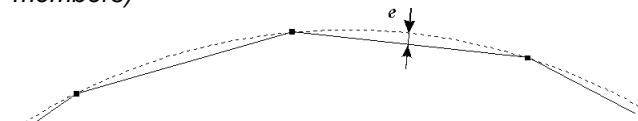
AxisVM structural member interface.

NOTE:

When rib is defined please note that the point of application of defined loads are transferred, explained [here](#).

Enumerated types

enum	EMemberMeshType = {	
	mmtMaxDeviationFromArc = 0x0,	maximum deviation from the arc (only for curved members)
	mmtMaxElementSize = 0x1,	maximum element size
	mmtSegments = 0x2,	number of segments
	mmtAngle = 0x3 }	angle of each segment (only for curved members)
	Mesh type of the member.	



Records / structures

	RMemberMeshParameters = (
EMemberMeshType	MeshType	mesh type
double	MeshParam	mesh parameter
)	

Functions

long **ChangeLocalDirection**

Reverts local x direction. If end node indexes are *i* and *j* and local x direction points from *i* to *j*, *ChangeLocalDirection* makes it point from *j* to *i*. If successful, returns the member index, otherwise returns an error code ([errDatabaseNotReady](#), [errIndexOutOfBounds](#)).

long **CreateMeshWithCoordinates ([in] SAFEARRAY(RPoint3d) * Points)**

Points Array with point coordinates used for meshing of the member

If successful, returns number of Points in the array, otherwise returns an error code ([errDatabaseNotReady](#))

long **CreateMeshWithCoordinates_vb** (Visual Basic compatible function of [CreateMeshWithCoordinates](#))

long **DefineAsBeam ([in] long MaterialIndex, [in] long StartCrossSectionIndex, [in] long EndCrossSectionIndex, [i/o] RPoint3d StartEccentricity, [i/o] RPoint3d EndEccentricity)**

MaterialIndex index of the material
($0 < \text{MaterialIndex} \leq \text{AxisVMMaterials.Count}$)

StartCrossSectionIndex CrossSection index at the beginning of the line (according to local x direction)

EndCrossSectionIndex CrossSection index at the end of the line (according to local x direction)

StartEccentricity eccentricity at the start node (**not used**)

EndEccentricity eccentricity at the end node (**not used**)

Defines structural member as a beam. For beams with a constant cross-section *StartCrossSectionIndex* = *EndCrossSectionIndex*.

If successful, returns the member index according to [IAxisVMMembers](#), otherwise returns an error code ([errDatabaseNotReady](#), [errIndexOutOfBounds](#)).

long	DefineAsRib ([in] long MaterialIndex , [in] long StartCrossSectionIndex , [in] long EndCrossSectionIndex , [i/o] RPoint3d StartEccentricity , [i/o] RPoint3d EndEccentricity)
	<p style="margin-left: 20px;">MaterialIndex <i>index of the material ($0 < \text{MaterialIndex} \leq \text{AxisVMMaterials.Count}$)</i></p> <p style="margin-left: 20px;">StartCrossSectionIndex <i>CrossSection index at the beginning of the line (according to local x direction)</i></p> <p style="margin-left: 20px;">EndCrossSectionIndex <i>CrossSection index at the end of the line (according to local x direction)</i></p> <p style="margin-left: 20px;">StartEccentricity <i>eccentricity at the start node</i></p> <p style="margin-left: 20px;">EndEccentricity <i>eccentricity at the end node</i></p>
	<p><i>Defines a structural member as a rib. For ribs with a constant cross-section StartCrossSectionIndex = EndCrossSectionIndex.</i></p> <p><i>If successful, returns the member index according to IAxisVMMembers, otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBounds).</i></p>
long	DefineAsRibWithAutoExcentricity ([in] long MaterialIndex , [in] long StartCrossSectionIndex , [in] long EndCrossSectionIndex , [in] EAutoExcentricityType AutoExcentricityType , [in] double kx , [in] long Domain1 , [in] long Domain2)
	<p style="margin-left: 20px;">MaterialIndex <i>Material Index</i></p> <p style="margin-left: 20px;">StartCrossSectionIndex <i>CrossSection index at the beginning of the line (according to local x direction)</i></p> <p style="margin-left: 20px;">EndCrossSectionIndex <i>CrossSection index at the end of the line (according to local x direction)</i></p> <p style="margin-left: 20px;">AutoExcentricityType <i>Type of the automatic eccentricity</i></p> <p style="margin-left: 20px;">kx <i>Friction resistance between rib and surface</i></p> <p style="margin-left: 20px;">Domain1 <i>Domain index (filled if connected to domain)</i></p> <p style="margin-left: 20px;">Domain2 <i>Domain index (filled if rib is between two domains)</i></p>
	<p><i>Defines a structural member as a rib, where eccentricity is calculated using domain thicknesses (Domain1,Domain2). For ribs with a constant cross-section StartCrossSectionIndex = EndCrossSectionIndex. If successful, returns member index, otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBounds, mbeDomainIndexOutOfBounds).</i></p>
long	DefineAsTimberBeam ([in] long MaterialIndex , [in] long ServiceClass , [in] double kdef , [in] long StartCrossSectionIndex , [in] long EndCrossSectionIndex , [i/o] RPoint3d StartExcentricity , [i/o] RPoint3d EndExcentricity)
	<p style="margin-left: 20px;">MaterialIndex <i>Material Index</i></p> <p style="margin-left: 20px;">ServiceClass <i>Service class of timber (valid values are 1, 2, 3)</i></p> <p style="margin-left: 20px;">kdef <i>Kdef value (deformation factor for timber members)</i></p> <p style="margin-left: 20px;">StartCrossSectionIndex <i>CrossSection index at the beginning of the line (according to local x direction)</i></p> <p style="margin-left: 20px;">EndCrossSectionIndex <i>CrossSection index at the end of the line (according to local x direction)</i></p> <p style="margin-left: 20px;">StartExcentricity <i>Eccentricity at the beginning (according to local x direction)</i></p> <p style="margin-left: 20px;">EndExcentricity <i>Eccentricity at the end (according to local x direction)</i></p>
	<p><i>If successful, returns member index, otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBounds, mbelllegalServiceClassValue).</i></p>

long	DefineAsTimberRib ([in] long MaterialIndex , [in] long ServiceClass , [in] double kdef , [in] long StartCrossSectionIndex , [in] long EndCrossSectionIndex , [i/o] RPoint3d StartExcentricity , [i/o] RPoint3d EndExcentricity)
	MaterialIndex <i>Material Index</i>
	ServiceClass <i>Service class of timber (valid values are 1, 2, 3)</i>
	kdef <i>Kdef value (deformation factor for timber members)</i>
	StartCrossSectionIndex <i>CrossSection index at the beginning of the line (according to local x direction)</i>
	EndCrossSectionIndex <i>CrossSection index at the end of the line (according to local x direction)</i>
	StartExcentricity <i>Eccentricity at the beginning (according to local x direction)</i>
	EndExcentricity <i>Eccentricity at the end (according to local x direction)</i>
	<i>Defines a structural member as a timber rib. If successful, returns member index, otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBounds, leMaterialIndexOutOfBounds, leCrossSectionIndexOutOfBounds, mbelllegalServiceClassValue).</i>
long	DefineAsTimberRibWithAutoExcentricity ([in] long MaterialIndex , [in] long ServiceClass , [in] double kdef , [in] long StartCrossSectionIndex , [in] long EndCrossSectionIndex , [in] EAutoExcentricityType AutoExcentricityType , [in] double kx , [in] long Domain1 , [in] long Domain2)
	MaterialIndex <i>Material Index</i>
	ServiceClass <i>Service class of timber (valid values are 1, 2, 3)</i>
	kdef <i>Kdef value (deformation factor for timber members)</i>
	StartCrossSectionIndex <i>CrossSection index at the beginning of the line (according to local x direction)</i>
	EndCrossSectionIndex <i>CrossSection index at the end of the line (according to local x direction)</i>
	AutoExcentricityType <i>Type of automatic eccentricity</i>
	kx <i>Friction resistance between rib and surface</i>
	Domain1 <i>Domain index (filled if connected to domain)</i>
	Domain2 <i>Domain index (filled if rib is between two domains)</i>
	<i>Defines a structural member as a timber rib, where eccentricity is calculated using domain thicknesses (Domain1, Domain2). If successful, returns member index, otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBounds, mbelllegalServiceClassValue, mbelllegalServiceClassValue).</i>
long	DefineAsTimberTruss ([in] long MaterialIndex , [in] long ServiceClass , [in] double kdef , [in] long CrossSectionIndex , [in] ELineNonLinearity TrussType , [in] double Resistance)
	MaterialIndex <i>Material Index</i>
	ServiceClass <i>Service class of timber (valid values are 1, 2, 3)</i>
	kdef <i>Kdef value (deformation factor for timber members)</i>
	CrossSectionIndex <i>CrossSection index</i>
	TrussType <i>(non)linearity type of the truss</i>
	Resistance <i>Axial resistance (absolute value) only for InTensionOnly and InCompressionOnly types of truss.</i>
	<i>Defines a member as a timber truss element.</i>
	<i>If successful, returns the member index, otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBounds, mbelllegalServiceClassValue).</i>

long	DefineAsTruss ([in] long MaterialIndex , [in] long CrossSectionIndex , [in] ELineNonLinearity TrussType , [in] double Resistance)
	<p>MaterialIndex index of the material ($0 < \text{MaterialIndex} \leq \text{AxisVMMaterials.Count}$)</p> <p>CrossSectionIndex index of the cross-section ($0 < \text{CrossSectionIndex} \leq \text{AxisVMCrossSections.Count}$)</p> <p>TrussType nonlinear behaviour</p> <p>Resistance Axial resistance (absolute value) only for <i>InTensionOnly</i> and <i>InCompressionOnly</i> types of truss</p>
	<p>Defines a member as a truss element.</p> <p>If successful, returns the member index, otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBounds,).</p>
long	DeleteColumnReinforcementParameters
	<p>If successful, returns member index, otherwise returns an error code (errDatabaseNotReady).</p>
long	DeleteMesh
	<p>If successful, returns member index, otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBounds)</p>
long	DeleteLineElement
	<p>Delete all assigned properties (cross-sections, materials, etc) from the member. If successful, returns member index, otherwise returns an error code (errDatabaseNotReady).</p>
long	GenerateMeshWithParams ([i/o] RMemberMeshParameters * MeshParameters)
	<p>MeshParameters Mesh parameters</p> <p>If successful, returns number of existing parts before new meshing, otherwise returns an error code (errDatabaseNotReady)</p>
long	GetBeamData ([out] long MaterialIndex , [out] long StartCrossSectionIndex , [out] long EndCrossSectionIndex , [i/o] RPoint3d StartEccentricity , [i/o] RPoint3d EndEccentricity)
	<p>MaterialIndex index of the material ($0 < \text{MaterialIndex} \leq \text{AxisVMMaterials.Count}$)</p> <p>StartCrossSectionIndex index of the start cross-section (according to the local x direction) ($0 < \text{StartCrossSectionIndex} \leq \text{AxisVMCrossSections.Count}$)</p> <p>EndCrossSectionIndex index of the end cross-section (according to the local x direction) ($0 < \text{EndCrossSectionIndex} \leq \text{AxisVMCrossSections.Count}$)</p> <p>StartEccentricity eccentricity at the start node (not used)</p> <p>EndEccentricity eccentricity at the end node (not used)</p>
	<p>Get properties of a beam structural member. If successful, returns the member index according to IAxisVMMembers, otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBounds, mbeNotBeam).</p>
long	GetColumnReinforcementParameters ([i/o] RColumnReinforcementParameters ColumnReinforcementParameters)
	<p>ColumnReinforcementParameters column reinforcement parameters</p> <p>If successful, returns the member index, otherwise returns an error code (errDatabaseNotReady, mbeReinforcementParametersNotExist, mbeInvalidMemberType, mbeEmptyLineList).</p>
long	GetEndReleases ([out] RRelases EndReleases)
	<p>EndReleases end releases</p> <p>Get end releases of the structural member. If successful, returns the member index according to IAxisVMMembers, otherwise returns an error code (errDatabaseNotReady).</p>

long	GetGeomData ([i/o] RLineGeomData Value) Get the geometry data for the arc (only if GeomType = <i>IgtCircleArc</i>). If successful, returns the member index according to IAxisVMMembers , otherwise returns an error code (errDatabaseNotReady , errIndexOutOfBounds , mbePropertyNotValidForThisLineType).										
double	GetLineIDAndLineSectionID ([in] EAnalysisType AnalysisType, [in] long MemberSectionId, [out] long LineID, [out] long LineSectionID) <table style="margin-left: 20px;"> <tr> <td>AnalysisType</td><td><i>material Index</i></td></tr> <tr> <td>MemberSectionId</td><td><i>section index of the member</i> <i>0 < MemberSectionId ≤ AxisVMMember. SectionsCount</i></td></tr> <tr> <td>LineID</td><td><i>line index</i></td></tr> <tr> <td>LineSectionID</td><td><i>index of the section on the line</i></td></tr> </table> Get line index and section index of the line by member section index. If successful, returns the member index according to IAxisVMMembers , otherwise returns an error code (errDatabaseNotReady , errIndexOutOfBounds , errNotFound).	AnalysisType	<i>material Index</i>	MemberSectionId	<i>section index of the member</i> <i>0 < MemberSectionId ≤ AxisVMMember. SectionsCount</i>	LineID	<i>line index</i>	LineSectionID	<i>index of the section on the line</i>		
AnalysisType	<i>material Index</i>										
MemberSectionId	<i>section index of the member</i> <i>0 < MemberSectionId ≤ AxisVMMember. SectionsCount</i>										
LineID	<i>line index</i>										
LineSectionID	<i>index of the section on the line</i>										
long	GetLines ([out] SAFEARRAY(long) * Linelids) <table style="margin-left: 20px;"> <tr> <td>Linelids</td><td><i>line indexes</i></td></tr> </table> Get the successive line indexes of the constituents. If successful, returns the array length, otherwise returns an error code (errDatabaseNotReady).	Linelids	<i>line indexes</i>								
Linelids	<i>line indexes</i>										
long	GetLinesLocX ([out] SAFEARRAY(long) * Linelids) <table style="margin-left: 20px;"> <tr> <td>Linelids</td><td><i>line indexes</i></td></tr> </table> Get the line indexes in the same order as member's local x axis. If successful, returns the array length, otherwise returns an error code (errDatabaseNotReady).	Linelids	<i>line indexes</i>								
Linelids	<i>line indexes</i>										
long	GetMeshParameters ([i/o] RMemberMeshParameters* MeshParameters) <table style="margin-left: 20px;"> <tr> <td>MeshParameters</td><td><i>Mesh parameters</i></td></tr> </table> If successful, returns member index.	MeshParameters	<i>Mesh parameters</i>								
MeshParameters	<i>Mesh parameters</i>										
long	GetRibData ([out] long MaterialIndex, [out] long StartCrossSectionIndex, [out] long EndCrossSectionIndex, [i/o] RPoint3d StartEccentricity, [i/o] RPoint3d EndEccentricity) <table style="margin-left: 20px;"> <tr> <td>MaterialIndex</td><td><i>index of the material</i> <i>(0 < MaterialIndex ≤ AxisVMMaterials.Count)</i></td></tr> <tr> <td>StartCrossSectionIndex</td><td><i>index of the start cross-section (according to the local x direction)</i> <i>(0 < StartCrossSectionIndex ≤ AxisVMCrossSections.Count)</i></td></tr> <tr> <td>EndCrossSectionIndex</td><td><i>index of the end cross-section (according to the local x direction)</i> <i>(0 < EndCrossSectionIndex ≤ AxisVMCrossSections.Count)</i></td></tr> <tr> <td>StartEccentricity</td><td><i>eccentricity at the beginning (according to the local x direction)</i></td></tr> <tr> <td>EndEccentricity</td><td><i>eccentricity at the end (according to the local x direction)</i></td></tr> </table> Get properties of a rib structural member. If successful, returns the member index according to IAxisVMMembers , otherwise returns an error code (errDatabaseNotReady , errIndexOutOfBounds , mbeNotRib).	MaterialIndex	<i>index of the material</i> <i>(0 < MaterialIndex ≤ AxisVMMaterials.Count)</i>	StartCrossSectionIndex	<i>index of the start cross-section (according to the local x direction)</i> <i>(0 < StartCrossSectionIndex ≤ AxisVMCrossSections.Count)</i>	EndCrossSectionIndex	<i>index of the end cross-section (according to the local x direction)</i> <i>(0 < EndCrossSectionIndex ≤ AxisVMCrossSections.Count)</i>	StartEccentricity	<i>eccentricity at the beginning (according to the local x direction)</i>	EndEccentricity	<i>eccentricity at the end (according to the local x direction)</i>
MaterialIndex	<i>index of the material</i> <i>(0 < MaterialIndex ≤ AxisVMMaterials.Count)</i>										
StartCrossSectionIndex	<i>index of the start cross-section (according to the local x direction)</i> <i>(0 < StartCrossSectionIndex ≤ AxisVMCrossSections.Count)</i>										
EndCrossSectionIndex	<i>index of the end cross-section (according to the local x direction)</i> <i>(0 < EndCrossSectionIndex ≤ AxisVMCrossSections.Count)</i>										
StartEccentricity	<i>eccentricity at the beginning (according to the local x direction)</i>										
EndEccentricity	<i>eccentricity at the end (according to the local x direction)</i>										
long	GetStartReleases ([out] RReleases StartReleases) <table style="margin-left: 20px;"> <tr> <td>StartReleases</td><td><i>start releases</i></td></tr> </table> Get start releases of the structural member. If successful, returns the member index according to IAxisVMMembers , otherwise returns an error code (errDatabaseNotReady).	StartReleases	<i>start releases</i>								
StartReleases	<i>start releases</i>										

long **GetTrussData** ([out] long **MaterialIndex**, [out] long **CrossSectionIndex**,
 [out] [ELineNonlinearity](#) **TrussType**, [out] double **Resistance**)

MaterialIndex	<i>index of the material (0 < MaterialIndex ≤ AxisVMMaterials.Count)</i>
CrossSectionIndex	<i>index of the cross-section (0 < CrossSectionIndex ≤ AxisVMCrossSections.Count)</i>
TrussType	<i>nonlinear behaviour of the truss</i>
Resistance	<i>if nonzero, resistance of the truss (only if Truss Type > InTensionAndCompression)</i>

If successful, returns the member index, otherwise returns an error code ([errDatabaseNotReady](#), [errIndexOutOfBounds](#), [mbeNotTruss](#)).

long **GetTimberBeamData** ([out] long **MaterialIndex**, [out] long **ServiceClass**, [out] double **kdef**,
 [out] long **StartCrossSectionIndex**, [out] long **EndCrossSectionIndex**,
 [i/o] [RPoint3d](#) **StartExcentricity**, [i/o] [RPoint3d](#) **EndExcentricity**)

MaterialIndex	<i>Material Index</i>
ServiceClass	<i>Service class of timber (valid values are 1, 2, 3)</i>
kdef	<i>Kdef value (deformation factor for timber members)</i>
StartCrossSectionIndex	<i>index of the start cross-section (according to the local x direction) (0 < StartCrossSectionIndex ≤ AxisVMCrossSections.Count)</i>
EndCrossSectionIndex	<i>index of the end cross-section (according to the local x direction) (0 < EndCrossSectionIndex ≤ AxisVMCrossSections.Count)</i>
StartExcentricity	<i>eccentricity at the beginning (according to the local x direction)</i>
EndExcentricity	<i>eccentricity at the end (according to the local x direction)</i>

If successful, returns member index, otherwise returns an error code ([errDatabaseNotReady](#), [errIndexOutOfBounds](#), [mbeNotBeam](#)).

long **GetTimberRibData** ([out] long **MaterialIndex**, [out] long **ServiceClass**, [out] double **kdef**,
 [out] long **StartCrossSectionIndex**, [out] long **EndCrossSectionIndex**,
 [i/o] [RPoint3d](#) **StartExcentricity**, [i/o] [RPoint3d](#) **EndExcentricity**)

MaterialIndex	<i>Material Index</i>
ServiceClass	<i>Service class of timber (valid values are 1, 2, 3)</i>
kdef	<i>Kdef value (deformation factor for timber members)</i>
StartCrossSectionIndex	<i>index of the start cross-section (according to the local x direction) (0 < StartCrossSectionIndex ≤ AxisVMCrossSections.Count)</i>
EndCrossSectionIndex	<i>index of the end cross-section (according to the local x direction) (0 < EndCrossSectionIndex ≤ AxisVMCrossSections.Count)</i>
StartExcentricity	<i>eccentricity at the beginning (according to the local x direction)</i>
EndExcentricity	<i>eccentricity at the end (according to the local x direction)</i>

If successful, returns member index, otherwise returns an error code ([errDatabaseNotReady](#), [errIndexOutOfBounds](#), [mbeNotRib](#)).

long	GetRibDataWithAutoExcentricity ([out] long MaterialIndex , [out] long StartCrossSectionIndex , [out] long EndCrossSectionIndex , [out] EAutoExcentricityType AutoExcentricityType , [out] double kx , [out] long Domain1 , [out] long Domain2)
	<p style="margin-left: 20px;">MaterialIndex <i>Material Index</i></p> <p style="margin-left: 20px;">StartCrossSectionIndex <i>index of the start cross-section (according to the local x direction)</i> ($0 < \text{StartCrossSectionIndex} \leq \text{AxisVMCrossSections.Count}$)</p> <p style="margin-left: 20px;">EndCrossSectionIndex <i>index of the end cross-section (according to the local x direction)</i> ($0 < \text{EndCrossSectionIndex} \leq \text{AxisVMCrossSections.Count}$)</p> <p style="margin-left: 20px;">AutoExcentricityType <i>Type of the automatic eccentricity</i></p> <p style="margin-left: 20px;">kx <i>Friction resistance between rib and surface</i></p> <p style="margin-left: 20px;">Domain1 <i>Domain index (filled if connected to domain)</i></p> <p style="margin-left: 20px;">Domain2 <i>Domain index (filled if rib is between two domains)</i></p>
	If successful, returns member index, otherwise returns an error code (errDatabaseNotReady , errIndexOutOfBounds , mbeNotRib).
long	GetTimberRibDataWithAutoExcentricity ([out] long MaterialIndex , [out] long ServiceClass , [out] double kdef , [out] long StartCrossSectionIndex , [out] long EndCrossSectionIndex , [out] EAutoExcentricityType AutoExcentricityType , [out] double kx , [out] long Domain1 , [out] long Domain2)
	<p style="margin-left: 20px;">MaterialIndex <i>material Index</i></p> <p style="margin-left: 20px;">ServiceClass <i>service class of timber (valid values are 1, 2, 3)</i></p> <p style="margin-left: 20px;">kdef <i>Kdef value (deformation factor for timber members)</i></p> <p style="margin-left: 20px;">StartCrossSectionIndex <i>index of the start cross-section (according to the local x direction)</i> ($0 < \text{StartCrossSectionIndex} \leq \text{AxisVMCrossSections.Count}$)</p> <p style="margin-left: 20px;">EndCrossSectionIndex <i>index of the end cross-section (according to the local x direction)</i> ($0 < \text{EndCrossSectionIndex} \leq \text{AxisVMCrossSections.Count}$)</p> <p style="margin-left: 20px;">AutoExcentricityType <i>type of the automatic eccentricity</i></p> <p style="margin-left: 20px;">kx <i>Friction resistance between rib and surface</i></p> <p style="margin-left: 20px;">Domain1 <i>domain index (filled if connected to domain)</i></p> <p style="margin-left: 20px;">Domain2 <i>domain index (filled if rib is between two domains)</i></p>
	If successful, returns member index, otherwise returns an error code (errDatabaseNotReady , errIndexOutOfBounds , leNotRib mbeNotRib).
long	GetTimberTrussData ([out] long MaterialIndex , [out] long ServiceClass , [out] double kdef , [out] long CrossSectionIndex , [out] ELineNonLinearity TrussType , [out] double Resistance)
	<p style="margin-left: 20px;">MaterialIndex <i>material Index</i></p> <p style="margin-left: 20px;">ServiceClass <i>service class of timber (valid values are 1, 2, 3)</i></p> <p style="margin-left: 20px;">kdef <i>Kdef value (deformation factor for timber members)</i></p> <p style="margin-left: 20px;">CrossSectionIndex <i>cross-section index</i></p> <p style="margin-left: 20px;">TrussType <i>nonlinear behaviour of the truss</i></p> <p style="margin-left: 20px;">Resistance <i>if nonzero, resistance of the truss</i> (only if Truss Type \leftrightarrow InTensionAndCompression)</p>
	If successful, returns member index, otherwise returns an error code (errDatabaseNotReady , errIndexOutOfBounds , mbeNotTruss).
long	GetTrMatrix ([i/o] RMatrix3x3 * Value)
	Value <i>the transformation matrix</i>
	Get the transformation matrix of the member. If unsuccessful, returns an error code (errDatabaseNotReady , mbeMustBeBeamOrRibOrTuss).

double	GetXofMemberSectionID ([in] EAnalysisType AnalysisType, [in] long MemberSectionId)
	AnalysisType material Index
	MemberSectionId section indexof the member
	0 < MemberSectionId ≤ AxisVMMember. SectionsCount
	<i>If successful, returns absolute x position, otherwise returns an 0.</i>
long	SetColumnReinforcementParameters ([i/o] RColumnReinforcementParameters ColumnReinforcementParameters)
	ColumnReinforcementParameters column reinforcement parameters
	<i>If successful, returns the member index, otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBounds, mbeInvalidConcreteMaterialId, mbeInvalidConcreteMaterialId, mbeInvalidRebarSteelGradeId, mbeInvalidConcreteMaterialId, mbeInvalidColumnRebarsId, mbeInvalidMemberType, mbeEmptyLineList).</i>
long	SetEndReleases ([i/o] RRleases Value)
	<i>Set the end releases.</i>
	<i>If successful, returns the member index according to IAxisVMMembers, otherwise returns an error code (errDatabaseNotReady).</i>
long	SetStartReleases ([i/o] RRleases Value)
	<i>Set the start releases.</i>
	<i>If successful, returns the member index according to IAxisVMMembers, otherwise returns an error code (errDatabaseNotReady).</i>

Properties

EArcitectElemType	ArchitectElemType • Get or set architect element type
ELongBoolean	ColumnReinforcementParametersExists True if column rein. parameters exists (read only property)
unsigned long	ContourColour • Get or set contour colour. If value is 0xFFFFFFFF then same as assigned material colour
long	ContourColour_vb • Visual Basic compatible property of ContourColour
long	EndNode Get end node of the member. For members with a local axis, this is the node at the end of the local x axis.
long	Length Get length of the structural member [m]
ELineType	MemberType Get structural member type
ELongBoolean	IsBeam True if member is horizontal (read only property)
ELongBoolean	IsColumn True if member is vertical (read only property)
ELongBoolean	IsOtherType True if member is neither horizontall or verical (read only property)
unsigned long	MaterialColour • Get or set material colour. If value is 0xFFFFFFFF then same as assigned material colour.
long	MaterialColour_vb • Visual Basic compatible property of MaterialColour
long	SectionsCount [EAnalysisType AnalysisType] Get number of sections where results can be obtained
ELongBoolean	MeshExists True if mesh exists on member (read only property)
BSTR	Name Get name of the member
long	StoreyId Get storey index of the member <i>If unsuccessful, returns an error code (errDatabaseNotReady).</i>
long	StartNode Get start node of the member. For members with a local axis, this is the node at the start of the local x axis.
double	StiffnessReduction_A [long Index] • Get or set axial stiffness factor. Cross sections area is multiplied with this value in stiffness matrix. Default value is 1.

Used only for atLinearStatic and atLinearVibration analysis [types](#). (only valid for STAS and Eurocode [RO] standards)

double **StiffnessReduction_I**[long **Index**] • Get or set flexural stiffness factor. Cross sections inertia is multiplied with this value in stiffness matrix. Default value is 1. Used only for atLinearStatic and atLinearVibration analysis [types](#). (only valid for STAS and Eurocode [RO] standards)

double **Timber_kdef** • Get or set kdef value (deformation factor) of timber member

long **Timber_ServiceClass** • Get or set service class of timber member

double **Volume** Get volume of the member

double **Weight** Get weight of the member

long **UID** Get unique index of the member which remains the same while exists in the model

IAxisVMMovingLoads

All moving loads in the model can be added and read through MovingLoads property in [IAxisVMMModel](#), but moving loads are created through ObjectCreator property in [IAxisVMAplication](#)

Error codes

enum	EMovingLoadError = {	
	mleInvalidItemType = -100001	Invalid type of item
	mleInvalidSystemValue = -100002	Invalid system value
	mleInvalidMovingLoadType = -100003	Invalid type of moving load
	mleInvalidPathOrNodes = -100004	Invalid paths or nodes
	mleInvalidNValue = -100005	Invalid number of steps
	mleInvalidNormVLength = -10006	Invalid normal vector length
	mleInvalidPathOrNormV = -100007	Invalid path or normal vector
	mleInvalidLoadCase = -100008	Invalid load case
	mleInvalidLoadGroup = -100009	Invalid load group
	mleInvalidMovingLoad = -100010 }	Invalid moving load

Enumerated types

enum	EMovingLoadType = {	
	mLtMovingLoadOnBeam= 0x00,	Moving load on beam
	mLtMovingLoadOnDomain= 0x01 }	Moving load on domain

Functions

long	AddMovingLoadOnBeam ([in] IAxisVMMovingLoadOnBeam MovingLoadOnBeam)	 MovingLoadOnBeam Interface for defining moving load on beam Add moving load on beam to model. If successful, returns moving load index, otherwise an error code (mleInvalidLoadGroup , mleInvalidLoadCase , mleInvalidMovingLoad , errDatabaseNotReady).
long	AddMovingLoadOnDomain ([in] IAxisVMMovingLoadOnDomain MovingLoadOnDomain)	 MovingLoadOnDomain Interface for defining moving load on domain Add moving load on domain to model. If successful, returns moving load index, otherwise an error code (mleInvalidLoadGroup , mleInvalidLoadCase , mleInvalidMovingLoad , errDatabaseNotReady).
long	Delete ([in] long Index)	 Index Index of the moving load Delete moving load from the model. If successful, returns index, otherwise an error code (errIndexOutOfBounds , errDatabaseNotReady).
long	GetMovingLoadType ([in] long Index [out] EMovingLoadType MovingLoadType)	 Index Index of the moving load MovingLoadType Type of the moving load Get moving load type by index. If successful, returns index, otherwise an error code (errIndexOutOfBounds , errDatabaseNotReady , mleInvalidMovingLoad).
long	GetMovingLoadOnBeam ([in] long Index, [out] IAxisVMMovingLoadOnBeam AxisVMMovingLoadOnBeam)	 Index Index of the moving load AxisVMMovingLoadOnBeam Interface for modifying moving load on beam Get moving load on beam interface. If successful, returns moving load index, otherwise an error code (errIndexOutOfBounds , errDatabaseNotReady , errCOMServerInternalError , mleInvalidMovingLoad).

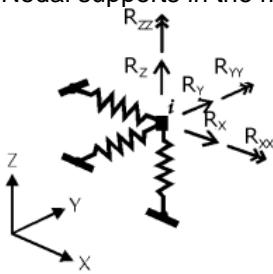
long	GetMovingLoadOnDomain ([in] long Index , [out] IAxisVMMovingLoadOnDomain AxisVMMovingLoadOnDomain)
	Index <i>Index of the moving load</i>
	AxisVMMovingLoadOnDomain <i>Interface for modifying moving load on domain</i>
	<i>Get moving load on domain interface. If successful, returns moving load index, otherwise an error code (errIndexOutOfBounds, errDatabaseNotReady, errCOMServerInternalError, mleInvalidMovingLoad).</i>
long	SetAsMovingLoadOnBeam ([in] long Index , [in] IAxisVMMovingLoadOnBeam AxisVMMovingLoadOnBeam)
	Index <i>Index of the moving load</i>
	AxisVMMovingLoadOnBeam <i>Interface for modifying moving load on beam</i>
	<i>Set as moving load on beam. If successful, returns moving load index, otherwise an error code (errIndexOutOfBounds, errDatabaseNotReady, mleInvalidMovingLoad).</i>
long	SetAsMovingLoadOnDomain ([in] long Index , [in] IAxisVMMovingLoadOnDomain AxisVMMovingLoadOnDomain)
	Index <i>Index of the moving load</i>
	AxisVMMovingLoadOnDomain <i>Interface for modifying moving load on domain</i>
	<i>Set as moving load on domain. If successful, returns moving load index, otherwise an error code (errIndexOutOfBounds, errDatabaseNotReady, mleInvalidMovingLoad).</i>

Properties

long **Count** *Get number of moving loads in the model*

IAxisVMNodalSupports

Warning! This interface, while still useable, was conceptually superseded by [IAxisVMNodesSupports](#).
Nodal supports in the model.



Error codes

enum	ESupportError = {	
	seNodeIndexOutOfBounds = -100001	node index ≤ 0 or $\geq \text{AxisVMNodes.Count}$
	seLineIndexOutOfBounds = -100002	line index ≤ 0 or $\geq \text{AxisVMLines.Count}$
	seReferenceIndexOutOfBounds = -100003	reference index ≤ 0 or $\geq \text{AxisVMReferences.Count}$
	selIncompatibleReferences = -100004 }	Incompatible x and z references for a local support

Records / structures

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

Functions

long **AddNodalBeamRelative** ([i/o] **RStiffnesses** **Stiffnesses**,
[i/o] **RNonLinearity** **NonLinearity**, [i/o] **RResistances** **Resistances**, [in] long **NodeId**,
[in] long **BeamId**)

Warning! This function has become obsolete, was superseded by *AddNodalBeamRelative_V153*

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 **AddNodalBeamRelative_V153** ([i/o] [RNodalSupportSpringParams](#) * NodalSupportSpringParams, [in] long **NodId**, [in] long **BeamId**)

NodalSupportSpringParams spring characteristics

NodId node index

(0 < Index ≤ [AxisVMNodes.Count](#))

BeamId line index (beam or rib)

(0 < Index ≤ [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](#), [errIndexOutOfBounds](#), [nsePropertyNotValidForThisType](#), [nseInvalidType](#)).

long **AddNodalEdgeRelative** ([i/o] [RStiffnesses](#) **Stiffnesses**, [i/o] [RNonLinearity](#) **NonLinearity**, [i/o] [RResistances](#) **Resistances**, [in] long **NodId**, [in] long **LinId**, [in] long **SurfacId1**, [in] long **SurfacId2**, [in] long **DomainId1**, [in] long **DomainId2**,)

Warning! This function has become obsolete, was superseded by [AddNodalEdgeRelative_V153](#)

Stiffnesses nodal support stiffnesses

NonLinearity nonlinear behaviour of support components

Resistances resistance of support components

NodId node index

(0 < Index ≤ [AxisVMNodes.Count](#))

LinId line index (edge)

(0 < Index ≤ [AxisVMLines.Count](#))

SurfacId1 first connecting surface

(0 < SurfacId1 ≤ [AxisVMSurfaces.Count](#))

SurfacId2 second connecting surface

(0 < SurfacId2 ≤ [AxisVMSurfaces.Count](#))

DomainId1 first connecting domain

(0 < DomainId1 ≤ [AxisVMDomains.Count](#))

DomainId2 second connecting domain

(0 < DomainId2 ≤ [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 indexes 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 indexes 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 an error code ([errDatabaseNotReady](#), [seNodeIndexOutOfBounds](#), [seLineIndexOutOfBounds](#)).

long **AddNodalEdgeRelative_V153** ([i/o] [RNodalSupportSpringParams](#) * NodalSupportSpringParams, [in] long **NodId**, [in] long **LinId**, [in] long **SurfacId1**, [in] long **SurfacId2**, [in] long **DomainId1**, [in] long **DomainId2**,)

NodalSupportSpringParams spring characteristics

NodId node index

(0 < Index ≤ [AxisVMNodes.Count](#))

LinId line index (edge)

(0 < Index ≤ [AxisVMLines.Count](#))

SurfacId1 first connecting surface

(0 < SurfacId1 ≤ [AxisVMSurfaces.Count](#))

SurfacId2 second connecting surface

(0 < SurfacId2 ≤ [AxisVMSurfaces.Count](#))

DomainId1 first connecting domain

(0 < DomainId1 ≤ [AxisVMDomains.Count](#))

DomainId2 second connecting domain

(0 < DomainId2 ≤ [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 indexes 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 indexes 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 an error code ([errDatabaseNotReady](#), [seNodeIndexOutOfBounds](#), [seLineIndexOutOfBounds](#), [errIndexOutOfBounds](#), [nsePropertyNotValidForThisType](#), [nseInvalidType](#)).

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

Warning! This function has become obsolete, was superseded by [AddNodalGlobal_V153](#)

Stiffnesses	<i>nodal support stiffnesses</i>
NonLinearity	<i>nonlinear behaviour of support components</i>
Resistances	<i>resistance of support components</i>
Nodeld	<i>node index</i> (0 < Index ≤ 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 **AddNodalGlobal_V153** ([i/o] [RNodalSupportSpringParams](#) * **NodalSupportSpringParams**, [in] long **Nodeld**)

NodalSupportSpringParams	<i>spring characteristics</i>
Nodeld	<i>node index</i> (0 < Index ≤ 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](#), [errIndexOutOfBounds](#), [nsePropertyNotValidForThisType](#), [nseInvalidType](#)).

long **AddNodalLocal** ([i/o] [RStiffnesses](#) **Stiffnesses**, [i/o] [RNonLinearity](#) **NonLinearity**,
[i/o] [RResistances](#) **Resistances**, [in] long **Nodeld**, [in] long **Referenceldx**, [in] long **Referenceldz**)

Warning! This function has become obsolete, was superseded by [AddNodalLocal_V153](#)

Stiffnesses	<i>nodal support stiffnesses</i>
NonLinearity	<i>nonlinear behaviour of support components</i>
Resistances	<i>resistance of support components</i>
Nodeld	<i>node index</i> (0 < Index ≤ AxisVMNodes.Count)
Referenceldx	<i>reference index for x direction</i> (0 < Index ≤ AxisVMReferences.Count)
Referenceldz	<i>reference index for z direction</i> (0 < Index ≤ AxisVMReferences.Count)

Adds a nodal support to the model, with custom *x* and *z* direction. **Referenceldx** defines the *x* direction, **Referenceldz** is used to calculate the perpendicular *z* direction to *x*. The *y* direction is derived from *x* and *z*. If successful, returns the index of the nodal support, otherwise returns an error code ([errDatabaseNotReady](#), [seNodeIndexOutOfBounds](#), [seReferenceIndexOutOfBounds](#), [seIncompatibleReferences](#)).

long **AddNodalLocal_V153** ([i/o] [RNodalSupportSpringParams](#) * **NodalSupportSpringParams**, [in] long **Nodeld**, [in] long **Referenceldx**, [in] long **Referenceldz**)

NodalSupportSpringParams	<i>spring characteristics</i>
Nodeld	<i>node index</i> (0 < Index ≤ AxisVMNodes.Count)
Referenceldx	<i>reference index for x direction</i> (0 < Index ≤ AxisVMReferences.Count)
Referenceldz	<i>reference index for z direction</i> (0 < Index ≤ AxisVMReferences.Count)

Adds a nodal support to the model, with custom *x* and *z* direction. **Referenceldx** defines the *x* direction, **Referenceldz** is used to calculate the perpendicular *z* direction to *x*. The *y* direction is derived from *x* and *z*. If successful, returns the index of the nodal support, otherwise returns an error code ([errDatabaseNotReady](#), [seNodeIndexOutOfBounds](#), [seReferenceIndexOutOfBounds](#),

[selIncompatibleReferences](#), [errIndexOutOfBounds](#), [nsePropertyNotValidForThisType](#), [nseInvalidType](#)).

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

Warning! This function has become obsolete, was superseded by [AddNodalReference_V153](#)

Stiffnesses nodal support stiffnesses
NonLinearity nonlinear behaviour of support components
Resistances resistance of support components
Nodeld node index
 (0 < Index ≤ [AxisVMNodes.Count](#))
Referenceld a reference indexe
 (0 < Index ≤ [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 **AddNodalReference_V153** ([i/o] [RNodalSupportSpringParams](#) * **NodalSupportSpringParams**, [in] long **Nodeld**, [in] long **Referenceld**)

NodalSupportSpringParams spring characteristics
Nodeld node index
 (0 < Index ≤ [AxisVMNodes.Count](#))
Referenceld a reference indexe
 (0 < Index ≤ [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](#), [errIndexOutOfBounds](#), [nsePropertyNotValidForThisType](#), [nseInvalidType](#)).

long **AddIsolator** ([i/o] [RNodalSupportSpringParams](#) * **NodalSupportSpringParams**, [in] long **Nodeld**)

NodalSupportSpringParams spring characteristics
Nodeld node index
 (0 < Index ≤ [AxisVMNodes.Count](#))

Adds a nodal seismic isolator support to the model. If successful, returns the index of the nodal support, otherwise returns an error code ([errDatabaseNotReady](#), [seNodeIndexOutOfBounds](#), [seReferenceIndexOutOfBounds](#), [errIndexOutOfBounds](#), [nsePropertyNotValidForThisType](#), [nseInvalidType](#)).

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](#)).

long **GetSelectedItemIds** ([out] SAFEARRAY (long) * **ItemIds**)

ItemIds list of selected nodal supports

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

long **GetTrMatrix** ([in] long **Index**, [i/o] [RMatrix3x3](#) **Value**)

Index nodal support index
Value Transformation matrix of the nodal support

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

long <u>EBoolean</u>	Count <i>Get number of nodal supports in the model</i> HaveStiffnessCalcParam [long Index] <i>Returns lbTrue if support has defined parameters for calculating stiffness</i> Item [long Index] <i>Get a nodal support interface</i> Selected [long Index] • <i>Get or set the selection status of a nodal support</i> <i>NOTE: Call <u>Refresh</u> function afterwards if not called between functions <u>BeginUpdate</u> and <u>EndUpdate</u></i>
long <u>AxisVMNodalSupport</u> *	SelCount <i>Get number of selected nodal supports in the model</i>

IAxisVMNodalSupport

AxisVM nodal support interface.

Enumerated types

```
enum ENodealSupportType = {
    nstNodalGlobal = 0x00,
    nstNodalBeamRelative = 0x01,
    nstNodalEdgeRelative = 0x02,
    nstNodalReference = 0x03 }
    Nodal support types
```

*nodal support in global directions
beam / rib relative nodal support
edge relative nodal support
nodal support in reference direction*

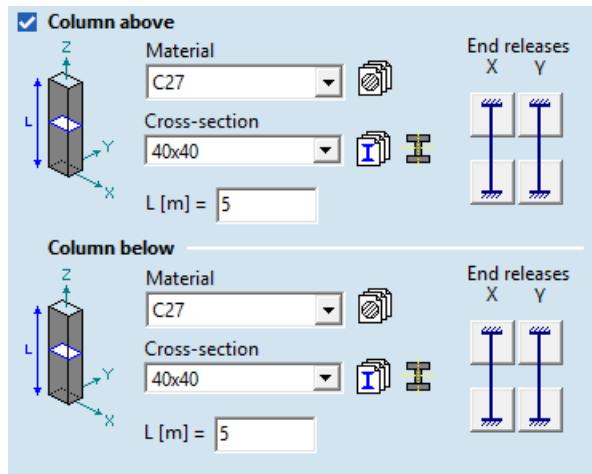
Error codes

```
enum ESupportError = {
    seNodeIndexOutOfBounds = -100001
    seLineIndexOutOfBounds = -100002
    seReferenceIndexOutOfBounds = -100003 }
```

*node index ≤ 0 or $\geq \text{AxisVMNodes.Count}$
line index ≤ 0 or $\geq \text{AxisVMLines.Count}$
reference index ≤ 0 or $\geq \text{AxisVMReferences.Count}$*

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



Node support calculation

```
RElementStiffnessParams
    long
    ElongBoolean
    ElongBoolean
RColumnStiffnessParams = (
    CalcParams
    common calculation parameters
    CrossSectionID
    index of the cross-section of the supporting element
    EndReleaseYTop
    Top of the element is released in glob. y (glob. support) or loc. z direction
    (beam / rib / edge relative support)
    EndReleaseYBottom
    Bottom of the element is released in glob. y (glob. support) or loc. z
    direction (beam / rib / edge relative support)
)
RNodalSupportStiffParams = (
    Top
    Bottom
    calculation parameters of the supporting element (column) above support
    calculation parameters of the supporting element (column) below support
)
```

Functions

long	DefineAsNodalBeamRelative ([i/o] RStiffnesses Stiffnesses , [i/o] RNonLinearity NonLinearity , [i/o] RResistances Resistances , [in] long NodeId , [in] long BeamId) Warning! This function has become obsolete Redefines a support. For parameters see IAxisVMNodalSupports.AddNodalBeamRelative .
long	DefineAsNodalEdgeRelative ([i/o] RStiffnesses Stiffnesses , [i/o] RNonLinearity NonLinearity , [i/o] RResistances Resistances , [in] long NodeId , [in] long LineId , [in] long SurfaceId1 , [in] long SurfaceId2 , [in] long DomainId1 , [in] long DomainId2 ,) Warning! This function has become obsolete Redefines a support. For parameters see IAxisVMNodalSupports.AddNodalEdgeRelative .
long	DefineAsNodalGlobal ([i/o] RStiffnesses Stiffnesses , [i/o] RNonLinearity NonLinearity , [i/o] RResistances Resistances , [in] long NodeId) Warning! This function has become obsolete Redefines a support. For parameters see IAxisVMNodalSupports.AddNodalGlobal .
long	DefineAsNodalReference ([i/o] RStiffnesses Stiffnesses , [i/o] RNonLinearity NonLinearity , [i/o] RResistances Resistances , [in] long NodeId , [in] long ReferenceId) Warning! This function has become obsolete Redefines a support. For parameters see IAxisVMNodalSupports.AddNodalReference .
long	GetFootingDimensions ([i/o] RPadFootingDimensions Value) Warning! This function has become obsolete, was superseded by GetFootingParams_V153 Get the calculated dimensions of the footing. If successful, returns the index of the line support, otherwise returns an error code (errDatabaseNotReady).
long	GetFootingParams ([i/o] RPadFootingParams Params) Warning! This function has become obsolete, was superseded by GetFootingParams_V153 Get the footing design calculation parameters of the support. If successful, returns the index of the line support, otherwise returns an error code(errDatabaseNotReady).
long	GetFootingParams_V153 ([i/o] RPadFootingParams_V153 Params) Get the specified and calculated footing parameters of the support. If successful, returns the index of the line support, otherwise returns an error code(errDatabaseNotReady).
long	GetNonLinearity ([i/o] RNonLinearity Value) Warning! This function has become obsolete Get the nonlinear behaviour of the nodal support. If unsuccessful, returns an error code(errIndexOutOfBounds , errDatabaseNotReady).
long	GetResistances ([i/o] RResistances Value) Warning! This function has become obsolete Get the resistance components of the nodal support. If unsuccessful, returns an error code(errIndexOutOfBounds , errDatabaseNotReady).

long	GetStiffnesses ([i/o] RStiffnesses Value) Warning! This function has become obsolete Get the stiffness components of the nodal support. If unsuccessful, returns an error code(errIndexOutOfBoundsException , errDatabaseNotReady).
long	GetStiffnessCalcParams ([i/o] RNodalSupportStiffParams Value) Get calculation parameters for calculating the stiffness of the nodal support. If unsuccessful, returns an error code(errIndexOutOfBoundsException , errDatabaseNotReady).
long	SetNonLinearity ([i/o] RNonLinearity Value) Warning! This function has become obsolete Set the nonlinear behaviour of the nodal support. If unsuccessful, returns an error code(errIndexOutOfBoundsException , errDatabaseNotReady).
long	SetResistances ([i/o] RResistances Value) Warning! This function has become obsolete Set the resistance components of the nodal support. If unsuccessful, returns an error code(errIndexOutOfBoundsException , errDatabaseNotReady).
long	SetStiffnesses ([i/o] RStiffnesses Value) Warning! This function has become obsolete Set the stiffness components of the nodal support. If unsuccessful, returns an error code(errIndexOutOfBoundsException , errDatabaseNotReady).
long	SetStiffnessCalcParams ([i/o] RNodalSupportStiffParams Value) Set calculation parameters for calculating the stiffness of the nodal support. If unsuccessful, returns an error code(errIndexOutOfBoundsException , errDatabaseNotReady).

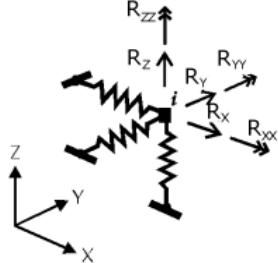
Properties

If **BeamId**, **DomainId...**, **LinId**, **Referenceld** or **Surfaceld...** property is not compatible with the support type, its value is [usePropertyNotValidForThisType](#).

long	BeamId (if <i>SupportType</i> is <i>nstNodalBeamRelative</i>) beam / rib index
EPadFootingType	FootingType Type of pad footing if it has been defined in AxisVM
ELongBoolean	HasFooting <i>lbTrue</i> if footing have been defined for the support in AxisVM
long	DomainId1 (if <i>SupportType</i> is <i>nstNodalEdgeRelative</i>) 1 st connecting domain
long	DomainId2 (if <i>SupportType</i> is <i>nstNodalEdgeRelative</i>) 2 nd connecting domain
long	LinId (if <i>SupportType</i> is <i>nstNodalEdgeRelative</i>) edge index
long	Nodeld node index
long	Referenceld (if <i>SupportType</i> is <i>nstNodalReference</i>) a reference index
RNodalSupportSpringParams	SpringParams spring characteristics
ENodalSupportType	SupportType support type
long	Surfaceld1 (if <i>SupportType</i> is <i>nstNodalEdgeRelative</i>) 1 st connecting surface
long	Surfaceld2 (if <i>SupportType</i> is <i>nstNodalEdgeRelative</i>) 2 nd connecting surface

IAxisVMNodesSupports

Interface for node supports

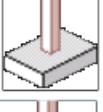
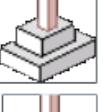
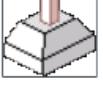


Error codes

```
enum ENodesSupportsError = {
    nsePropertyNotValidForThisType = -100001
    nseNodeIndexOutOfBounds = -100002
    nseMemberIndexOutOfBounds = -100003
    nseReferenceIndexOutOfBounds = -100004
    nselInvalidType = -100005
    nsePadFootingNotDefined = -100006
    nselInvalidMemberAndNodeCombination = -100007
    nseStiffnessCalcParamsNotDefined = -100008 }
```

LineID and ReferenceID properties can invoke Error event with this code in case it is not applicable.
node index is our of range
member index is our of range
reference index is our of range
not applicable, invalid type of support
not applicable, pad footing is not defined
not applicable, node is not attached to the member
not applicable, stiffness calculation parameters are not defined

Enumerated types

```
enum EPadFootingType = {
    pftPlate = 0x00           basic cubic footing

    pftStepped = 0x01         stepped pad footing

    pftSloped = 0x02          sloped pad footing

}
```

Type of the pad footing

```
enum EPadFootingType2 = {
    pft_NodeRectangular = 0x00
    pft_NodeCircular = 0x01
    pft_Linear = 0x02
}
```

General footing class

```
enum EPadFootingStepMeasureSource = {
    pfsmss_Interior = 0x00
    pfsmss_Edge = 0x01
}
```

Origin of the horizontal delta for the step

Records / structures

RSpringParamIndexes = (

long	x	<i>spring characteristic index for translation in direction x</i>
long	y	<i>spring characteristic index for translation in direction y</i>
long	z	<i>spring characteristic index for translation in direction z</i>
long	xx	<i>spring characteristic index for rotation around ax x</i>
long	yy	<i>spring characteristic index for rotation around ax y</i>
long	zz	<i>spring characteristic index for rotation around ax z</i>
)	
	RNodalSupportSpringParams = (
	<u>SpringParamIndexes</u>	<i>Indexes of the spring characteristics (for regular nodal supports, like global node support, domain relative node support, ...)</i>
long	IsolatorParamIndex	<i>Index of the isolator spring characteristic (only for seismic isolator node support)</i>
double	IsolatorD2	<i>Maximum design displacement at ULS [m] (only for seismic isolator node support)</i>
)	
		<i>Either SpringParamIndexes or (IsolatorParamIndex, IsolatorD2) has to be filled, depending on the type of the nodal support</i>
	RPadFootingDimensions = (
	Warning! This record has become obsolete, it was superseded by RPadFootingParams_V153 and RLinearFootingParams	
double	UpperThickness	<i>Thickness of the upper part</i>
double	LowerThickness	<i>Thickness of the lower part</i>
<u>RPoint2D</u>	UpperCornerA	<i>2D coordinates of point A, Point A is a vertex of rectangular of projected upper part</i>
<u>RPoint2D</u>	UpperCornerB	<i>2D coordinates of point B, Point B is a vertex of rectangular of projected upper part</i>
<u>RPoint2D</u>	LowerCornerA	<i>2D coordinates of point A, Point A is a vertex of rectangular of projected lower part</i>
<u>RPoint2D</u>	LowerCornerB	<i>2D coordinates of point B, Point B is a vertex of rectangular of projected lower part</i>
)	
	RPadFootingParams = (
	Warning! This record has become obsolete, it was superseded by RPadFootingParams_V153 and RLinearFootingParams	
double	HeightTop	<i>Height of upper part, h2 on form, see pic. below</i>
double	HeightBottom	<i>Height of lower part, h1 on form, see pic. below</i>
double	BaseThickness	<i>Height of the base, h_b on form, see pic. below</i>
double	bx	<i>Total width of the base, negative val. for max., see pic. below</i>
double	x1	<i>Left width of the base, negative val. for max., see pic. below</i>
double	x2	<i>Right width of the base, negative val. for max., see pic. below</i>
double	by	<i>Total length of the base, negative val. for max., node supports only, see pic. below</i>
double	y1	<i>Left length of the base, negative val. for max., node supports only, see pic. below</i>
double	y2	<i>Right length of the base, negative val. for max., node supports only, see pic. below</i>
double	dy1	<i>Left length of the base, negative val. for max., node supports only, see pic. below</i>
double	dy2	<i>Right length of the base, negative val. for max., node supports only, see pic. below</i>
long	MaterialID	<i>Material index of the concrete footing</i>
)	

Node supports:

Warning! This image is related to an obsolete record

Footing plate

b_x [mm] = 1500	b_y [mm] = 1500
x_1 max [mm] = 750	y_1 max [mm] = 750
x_2 max [mm] = 750	y_2 max [mm] = 750

Step

d [mm] = 500	t [mm] = 900
d^* [mm] = 500	h_2 [mm] = 300
dx_1 [mm] = 500	h_1 [mm] = 500
dx_2 [mm] = 500	h_b [mm] = 100
dy_1 [mm] = 500	
dy_2 [mm] = 500	

Member supports:

Warning! This image is related to an obsolete record

Footing plate

b [mm] = 2000	dx [mm] = 500
x_1 max [mm] = 1000	dx_1 [mm] = 500
x_2 max [mm] = 1000	dx_2 [mm] = 500

Step

d [mm] = 500	t [mm] = 900
d^* [mm] = 500	h_2 [mm] = 300
dx [mm] = 500	h_1 [mm] = 500
dx_1 [mm] = 500	h_b [mm] = 100
dx_2 [mm] = 500	

RRectangularFootingSpec = (see picture below)

EBoolean	FixedX1
EBoolean	FixedX2
double	x1
double	x2
EBoolean	FixedY1
EBoolean	FixedY2
double	y1
double	y2

x1 is fixed/locked
x2 is fixed/locked
value of x1 (left width from the column axis)
value of x2 (right width from the column axis)
y1 is fixed/locked
y2 is fixed/locked
value of y1 (top width from the column axis)
value of y2 (bottom width from the column axis)

interpretation of the origin of the step horizontal delta. dx_1 , dx_2 , dy_1 , dy_2 are interpreted with this rule

value of dx_1 (step left delta)

value of dx_2 (step right delta)

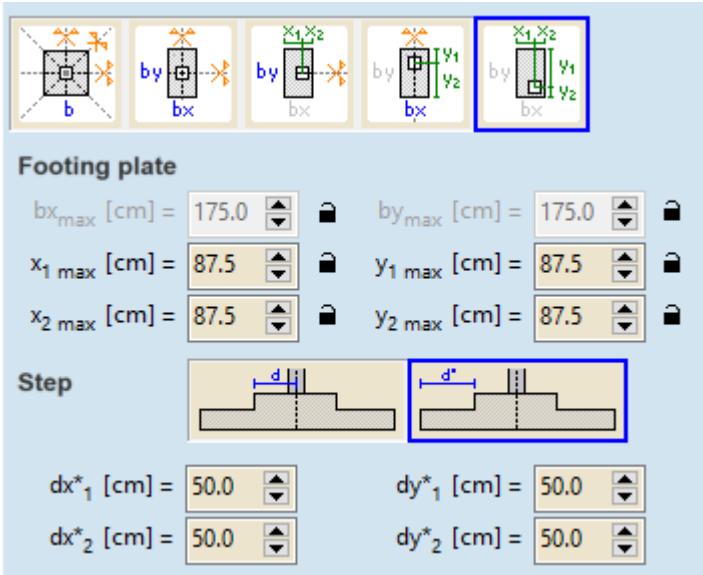
value of dy_1 (step top delta)

value of dy_2 (step bottom delta)

EPadFootingStepMeasureSource StepMeasureSource

double	dx1
double	dx2
double	dy1
double	dy2
)	

b_x and b_y are derived values, $b_x = x_1 + x_2$, $b_y = y_1 + y_2$. Their usage in the user interface can be emulated by specifying their half value in x_1 , x_2 , or y_1 , y_2 .



RRectangularFootingCalced = (

EBoolean	Calculated	True if the record contains a valid calculation
double	x_1	value of x_1 (left width from the column axis)
double	x_2	value of x_2 (right width from the column axis)
double	y_1	value of y_1 (top width from the column axis)
double	y_2	value of y_2 (bottom width from the column axis)

EPadFootingStepMeasureSource StepMeasureSource interpretation of the origin of the step horizontal delta. dx_1, dx_2, dy_1, dy_2 are interpreted with this rule

double	dx_1	value of dx_1 (step left delta)
double	dx_2	value of dx_2 (step right delta)
double	dy_1	value of dy_1 (step top delta)
double	dy_2	value of dy_2 (step bottom delta)

)

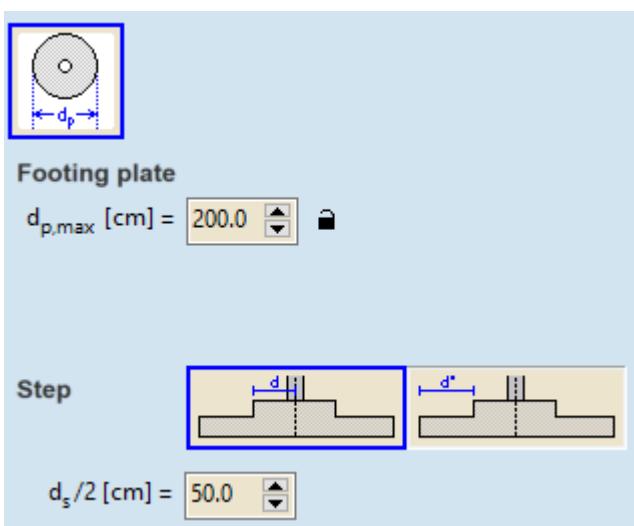
RCircularFootingSpec = (see picture below
diameter is fixed/locked

EBoolean	FixedDiam	diameter of the main element (plate or bottom step) (d_p)
double	Diam	interpretation of the origin of the step horizontal deltaR

EPadFootingStepMeasureSource StepMeasureSource value of deltaR (step delta) ($d_s/2$)

double	DeltaR
--------	--------

)



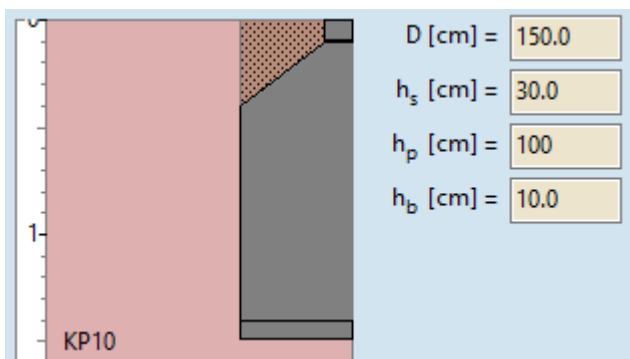
```

RCircularFootingCalced = (
    ELongBoolean Calculated True if the record contains a valid calculation
    double Diam diameter of the main element (plate or bottom step)
    EPadFootingStepMeasureSource StepMeasureSource interpretation of the origin of the step horizontal deltaR
    double DeltaR value of deltaR (step delta)
)

```

RPadFootingParams_V153 = (see picture below)

EPadFootingType2	FootingType	<i>Footing type class</i>
EPadFootingType	VerticalType	<i>Step setup</i>
long	MaterialId	<i>Material index of the concrete footing</i>
double	GroundToBottom	<i>Soil cover over the bottom plane of the footing (D)</i>
double	HeightMain	<i>Height of the main element (plate or bottom step) (h_p)</i>
double	HeightStep	<i>Step height (h_s)</i>
double	BlindThickness	<i>Thickness of the blind concrete (h_b)</i>
RRectangularFootingSpec	RectangularFootingSpec	<i>Specified parameters for FootingType = pft_NodeRectangular</i>
RRectangularFootingCalced	RectangularFootingCalced	<i>Calculated parameters for FootingType = pft_NodeRectangular</i>
RCircularFootingSpec	CircularFootingSpec	<i>Specified parameters for FootingType = pft_NodeCircular</i>
RCircularFootingCalced	CircularFootingCalced	<i>Calculated parameters for FootingType = pft_NodeCircular</i>
)	



RLinearFootingParams = (see picture above)

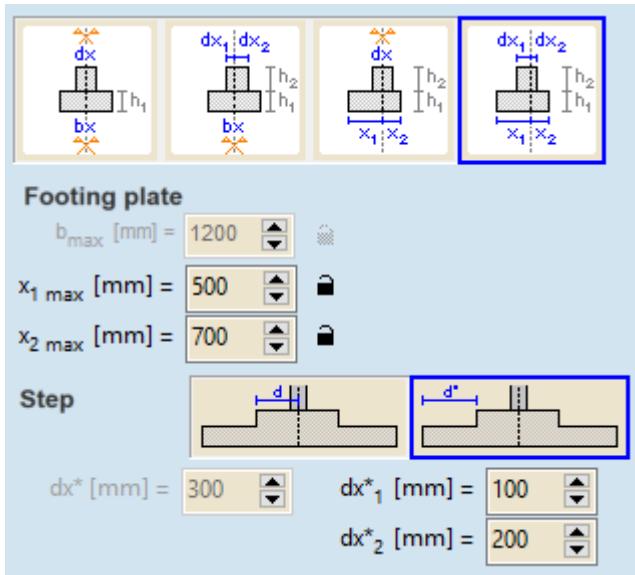
EPadFootingType	VerticalType	<i>Step setup</i>
long	MaterialId	<i>Material index of the concrete footing</i>
double	GroundToBottom	<i>Soil cover over the bottom plane of the footing (D)</i>
double	HeightMain	<i>Height of the main element (plate or bottom step) (h_p)</i>
double	HeightStep	<i>Step height (h_s)</i>
double	BlindThickness	<i>Thickness of the blind concrete (h_b)</i>
RLinearFootingSpec	FootingSpec	<i>Specified parameters for the line support</i>
RLinearFootingCalced	FootingCalced	<i>Calculated parameters for the line support</i>
)	

RLinearFootingSpec = (see picture below)

ELongBoolean	FixedX1	<i>x1 is fixed/locked</i>
ELongBoolean	FixedX2	<i>x2 is fixed/locked</i>
double	x1	<i>value of x1 (left width from the wall axis)</i>
double	x2	<i>value of x2 (right width from the wall axis)</i>

[EPadFootingStepMeasureSource](#) StepMeasureSource
double dx1
double dx2
)

interpretation of the origin of the step horizontal delta. dx1, dx2 are interpreted with this rule
value of dx1 (step left delta)
value of dx2 (step right delta)



[RLinearFootingCalced](#) = (See picture above
EBoolean Calculated
double x1
double x2
)

[EPadFootingStepMeasureSource](#) StepMeasureSource
double dx1
double dx2
)

True if the record contains a valid calculation
value of x1 (left width from the wall axis)
value of x2 (right width from the wall axis)
interpretation of the origin of the step horizontal delta. dx1, dx2 are interpreted with this rule
value of dx1 (step left delta)
value of dx2 (step right delta)

Functions

long [AddIsolator](#) ([i/o] [RNodalSupportSpringParams](#) * NodalSupportSpringParams, [in] long **NodId**)
NodalSupportSpringParams spring characteristics
NodId node index
 $(0 < \text{Index} \leq \text{AxisVMNodes.Count})$

Adds a nodal seismic isolator support to the model. If successful, returns the index of the nodal support, otherwise returns an error code ([errDatabaseNotReady](#), [seNodeIndexOutOfBounds](#), [seReferenceIndexOutOfBounds](#), [errIndexOutOfBounds](#), [nsePropertyNotValidForThisType](#), [nseInvalidType](#)).

long [AddNodalGlobal](#) ([i/o] [RStiffnesses](#) **Stiffnesses**, [i/o] [RNonLinearity](#) **NonLinearity**,
[i/o] [RResistances](#) **Resistances**, [in] long **NodId**)

Warning! This function has become obsolete, was superseded by AddNodalGlobal_V153

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

Adds a node support. Components are defined in the global directions.

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

long	AddNodalGlobal_V153 ([i/o] RNodalSupportSpringParams * NodalSupportSpringParams, [in] long Nodeld)										
	<p>NodalSupportSpringParams <i>spring characteristics</i></p> <p>Nodeld <i>node index</i> (0 < Index ≤ AxisVMNodes.Count)</p> <p><i>Adds a node support. Components are defined in the global directions.</i> If successful, returns the index of the node support, otherwise returns an error code (errDatabaseNotReady, seNodeIndexOutOfBounds, errIndexOutOfBounds, nsePropertyNotValidForThisType, nseInvalidType).</p>										
long	AddNodalLocal_V153 ([i/o] RNodalSupportSpringParams * NodalSupportSpringParams, [in] long Nodeld , [in] long Referenceldx , [in] long Referenceldz)										
	<p>NodalSupportSpringParams <i>spring characteristics</i></p> <p>Nodeld <i>node index</i> (0 < Index ≤ AxisVMNodes.Count)</p> <p>Referenceldx <i>reference index for x direction</i> (0 < Index ≤ AxisVMReferences.Count)</p> <p>Referenceldz <i>reference index for z direction</i> (0 < Index ≤ AxisVMReferences.Count)</p> <p><i>Adds a nodal support to the model, with custom x and z direction. Referenceldx defines the x direction, Referenceldz is used to calculate the perpendicular z direction to x. The y direction is derived from x and z. If successful, returns the index of the nodal support, otherwise returns an error code (errDatabaseNotReady, seNodeIndexOutOfBounds, seReferenceIndexOutOfBounds, selIncompatibleReferences, errIndexOutOfBounds, nsePropertyNotValidForThisType, nseInvalidType).</i></p>										
long	AddNodalMemberRelative ([i/o] RStiffnesses Stiffnesses , [i/o] RNonLinearity NonLinearity , [i/o] RResistances Resistances , [in] long Nodeld , [in] long BeamId)										
	<p>Warning! This function has become obsolete, was superseded by AddNodalMemberRelative_V153</p> <table> <tr> <td>Stiffnesses</td><td><i>nodal support stiffnesses</i></td></tr> <tr> <td>NonLinearity</td><td><i>nonlinear behaviour of support components</i></td></tr> <tr> <td>Resistances</td><td><i>resistance of support components</i></td></tr> <tr> <td>Nodeld</td><td><i>node index</i> (0 < Index ≤ AxisVMNodes.Count)</td></tr> <tr> <td>MemberID</td><td><i>member index (beam or rib)</i> (0 < Index ≤ AxisVMMembers.Count)</td></tr> </table> <p><i>Adds a node support relative to member. Components are defined in the local system of the member (beam or rib). If successful, returns the index of the node support, otherwise returns an error code (errDatabaseNotReady, nseInvalidMemberAndNodeCombination).</i></p>	Stiffnesses	<i>nodal support stiffnesses</i>	NonLinearity	<i>nonlinear behaviour of support components</i>	Resistances	<i>resistance of support components</i>	Nodeld	<i>node index</i> (0 < Index ≤ AxisVMNodes.Count)	MemberID	<i>member index (beam or rib)</i> (0 < Index ≤ AxisVMMembers.Count)
Stiffnesses	<i>nodal support stiffnesses</i>										
NonLinearity	<i>nonlinear behaviour of support components</i>										
Resistances	<i>resistance of support components</i>										
Nodeld	<i>node index</i> (0 < Index ≤ AxisVMNodes.Count)										
MemberID	<i>member index (beam or rib)</i> (0 < Index ≤ AxisVMMembers.Count)										
long	AddNodalMemberRelative_V153 ([i/o] RNodalSupportSpringParams * NodalSupportSpringParams, [in] long Nodeld , [in] long BeamId)										
	<p>NodalSupportSpringParams <i>spring characteristics</i></p> <p>Nodeld <i>node index</i> (0 < Index ≤ AxisVMNodes.Count)</p> <p>MemberID <i>member index (beam or rib)</i> (0 < Index ≤ AxisVMMembers.Count)</p> <p><i>Adds a node support relative to member. Components are defined in the local system of the member (beam or rib). If successful, returns the index of the node support, otherwise returns an error code (errDatabaseNotReady, nseInvalidMemberAndNodeCombination, errIndexOutOfBounds, nsePropertyNotValidForThisType, nseInvalidType).</i></p>										
long	AddNodalDomainRelative ([i/o] RStiffnesses Stiffnesses , [i/o] RNonLinearity NonLinearity , [i/o] RResistances Resistances , [in] long Nodeld , [in] long MemberID , [in] long DomainId1 , [in] long DomainId2)										
	<p>Warning! This function has become obsolete, was superseded by AddNodalDomainRelative_V153</p> <table> <tr> <td>Stiffnesses</td><td><i>nodal support stiffnesses</i></td></tr> <tr> <td>NonLinearity</td><td><i>nonlinear behaviour of support components</i></td></tr> <tr> <td>Resistances</td><td><i>resistance of support components</i></td></tr> <tr> <td>Nodeld</td><td><i>node index</i> (0 < Index ≤ AxisVMNodes.Count)</td></tr> <tr> <td>MemberID</td><td><i>member index (beam or rib)</i> (0 < Index ≤ AxisVMMembers.Count)</td></tr> </table>	Stiffnesses	<i>nodal support stiffnesses</i>	NonLinearity	<i>nonlinear behaviour of support components</i>	Resistances	<i>resistance of support components</i>	Nodeld	<i>node index</i> (0 < Index ≤ AxisVMNodes.Count)	MemberID	<i>member index (beam or rib)</i> (0 < Index ≤ AxisVMMembers.Count)
Stiffnesses	<i>nodal support stiffnesses</i>										
NonLinearity	<i>nonlinear behaviour of support components</i>										
Resistances	<i>resistance of support components</i>										
Nodeld	<i>node index</i> (0 < Index ≤ AxisVMNodes.Count)										
MemberID	<i>member index (beam or rib)</i> (0 < Index ≤ AxisVMMembers.Count)										

	DomainId1 first connecting domain ($0 < \text{DomainId1} \leq \text{AxisVMDomains.Count}$) DomainId2 second connecting domain, could be 0 ($0 < \text{DomainId2} \leq \text{AxisVMDomains.Count}$)	Adds a node support. Connecting s 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 member (edge of domain) and the domain's local z direction. If only one connecting domain is specified (other domain index is zero) the local x direction is along the edge, the local y direction is in the plane of the domain and perpendicular to the edge, the orientation of the local z direction depends on the local z direction of the domain. If two indexes 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 node support index, otherwise returns an error code (errDatabaseNotReady , nseInvalidMemberAndNodeCombination).
long	AddNodalDomainRelative_V153 ([i/o] RNodalSupportSpringParams * NodalSupportSpringParams, [in] long NodId , [in] long MemberID , [in] long DomainId1 , [in] long DomainId2)	
	NodalSupportSpringParams spring characteristics NodId node index ($0 < \text{Index} \leq \text{AxisVMNodes.Count}$) MemberID member index (beam or rib) ($0 < \text{Index} \leq \text{AxisVMMembers.Count}$) DomainId1 first connecting domain ($0 < \text{DomainId1} \leq \text{AxisVMDomains.Count}$) DomainId2 second connecting domain, could be 0 ($0 < \text{DomainId2} \leq \text{AxisVMDomains.Count}$)	Adds a node support. Connecting s 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 member (edge of domain) and the domain's local z direction. If only one connecting domain is specified (other domain index is zero) the local x direction is along the edge, the local y direction is in the plane of the domain and perpendicular to the edge, the orientation of the local z direction depends on the local z direction of the domain. If two indexes 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 node support index, otherwise returns an error code (errDatabaseNotReady , nseInvalidMemberAndNodeCombination , errIndexOutOfBounds , nsePropertyNotValidForThisType , nseInvalidType).
long	AddNodalReferenceRelative ([i/o] RStiffnesses Stiffnesses , [i/o] RNonLinearity NonLinearity , [i/o] RResistances Resistances , [in] long NodId , [in] long ReferencId) Warning! This function has become obsolete, was superseded by AddNodalReferenceRelative_V153	Adds a node support. 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 , nseNodeIndexOutOfBounds , nseReferenceIndexOutOfBounds).
long	AddNodalReferenceRelative_V153 ([i/o] RNodalSupportSpringParams * NodalSupportSpringParams, [in] long NodId , [in] long ReferencId)	Adds a node support. 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 , nseNodeIndexOutOfBounds , nseReferenceIndexOutOfBounds , errIndexOutOfBounds , nsePropertyNotValidForThisType , nseInvalidType).
long	Delete ([in] long Index) Index node support index	Deletes a node support. If successful, returns the number of deleted node supports, otherwise returns an error code (errDatabaseNotReady , errIndexOutOfBounds).

long	DeleteSelected
Deletes the selected node supports. If successful, returns the number of deleted supports, otherwise returns an error code (errDatabaseNotReady).	
long	GetFootingParams ([in] long Index, [i/o] RPadFootingParams FootingParams)
Warning! This function has become obsolete, was superseded by GetFootingParams_V153	
<p style="text-align: center;">Index node support index FootingParams pad footing parameters</p> <p>Get the footing design calculation parametres of the support. If successful, returns the index of the line support, otherwise returns an error code(errDatabaseNotReady).</p>	
long	GetFootingParams_V153 ([in] long Index, [i/o] RPadFootingParams_V153 FootingParams)
<p style="text-align: center;">Index node support index FootingParams pad footing parameters</p> <p>Get the specified and calculated parameters of the footing. If successful, returns the index of the line support, otherwise returns an error code(errDatabaseNotReady).</p>	
long	GetNodalSupportID ([in] long Index)
<p style="text-align: center;">Index node support index</p> <p>Get nodal support index of IAxisVMNodalSupports which will be deprecated. If successful, returns the number of deleted supports, otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBounds).</p>	
long	GetNonLinearity ([in] long Index, [i/o] RNonLinearity NonLinearity)
<p style="text-align: center;">Index node support index NonLinearity node support non-linearity</p> <p>Get the nonlinear behaviour of the node support. If unsuccessful, returns an error code(errIndexOutOfBounds, errDatabaseNotReady).</p>	
long	GetResistances ([in] long Index, [i/o] RResistances Resistances)
<p style="text-align: center;">Index node support index Resistances node support resistances</p> <p>Get the resistance components of the node support. If unsuccessful, returns an error code(errIndexOutOfBounds, errDatabaseNotReady).</p>	
long	GetStiffnesses ([in] long Index, [i/o] RStiffnesses Stiffness)
<p style="text-align: center;">Index node support index Stiffness node support stiffnesses</p> <p>Get the stiffness components of the node support. If unsuccessful, returns an error code(errIndexOutOfBounds, errDatabaseNotReady).</p>	
long	GetStiffnessCalcParams ([in] long Index, [i/o] RNodalSupportStiffParams StiffnessParams)
<p style="text-align: center;">Index node support index StiffnessParams node support stiffnesses parameters</p> <p>Get calculation parameters for calculating the stiffness of the node support. If unsuccessful, returns an error code(errIndexOutOfBounds, errDatabaseNotReady).</p>	
long	GetSelectedItemIds ([out] SAFEARRAY (long) * ItemIds)
<p style="text-align: center;">ItemIds list of selected node supports</p> <p>If successful, returns the number of selected elements otherwise returns an error code(errDatabaseNotReady).</p>	
long	GetTrMatrix ([in] long Index, [i/o] RMatrix3x3 TrMatrix)
<p style="text-align: center;">Index node support index TrMatrix Transformation matrix of the node support</p> <p>Gets transformation matrix of the node support. If successful, returns the index of the node support, otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBounds).</p>	
long	RenameSelected ([in] long NewBase, [in] BSTR FormatStr)

NewBase Start number for renaming, must be a positive number
FormatStr Prefix string of the new name + "_" eg "NodeSupport_"
Rename selected node supports.
Example: NewBase=100 and FormatStr= 'new_ ' then names are: 'new100', 'new101',...etc.
If successful returns 1, otherwise returns an error code ([errDatabaseNotReady](#)).

long **SelectAll** ([in] [ELongBoolean](#) **Select**)

Select *lbTrue for select or lbFalse for deselect*

Select/deselect all node supports. If successful, returns the number of selected node supports, otherwise returns an error code ([errDatabaseNotReady](#)).

long **SetNonLinearity** ([in] long **Index**, [i/o] [RNonLinearity](#) **NonLinearity**)

Index *node support index*

NonLinearity *node support non-linearity*

Set the nonlinear behaviour of the nodal support. If unsuccessful, returns an error code([errIndexOutOfBounds](#), [errDatabaseNotReady](#)).

long **SetResistances** ([in] long **Index**, [i/o] [RResistances](#) **Resistances**)

Index *node support index*

Resistances *node support resistances*

Set the resistance components of the node support. If unsuccessful, returns an error code([errIndexOutOfBounds](#), [errDatabaseNotReady](#)).

long **SetStiffnesses** ([in] long **Index**, [i/o] [RStiffnesses](#) **Stiffness**)

Index *node support index*

Stiffness *node support stiffnesses*

Set the stiffness components of the node support. If unsuccessful, returns an error code([errIndexOutOfBounds](#), [errDatabaseNotReady](#)).

long **SetStiffnessCalcParams** ([in] long **Index**, [i/o] [RNodalSupportStiffParams](#) **StiffnessParams**)

Index *node support index*

StiffnessParams *node support stiffnesses calculation parameters*

Set calculation parameters for calculating the stiffness of the node support. If unsuccessful, returns an error code([errIndexOutOfBounds](#), [errDatabaseNotReady](#)).

Properties

long **Count** *Get number of node supports in the model*

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

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

[ELongBoolean](#) **HasFooting**[long **Index**] *Returns lbTrue if footing have been defined for the node support*

[ELongBoolean](#) **HasStiffnessCalcParam**[long **Index**] *Returns lbTrue if support has defined parameters for calculating stiffness*

[EPadFootingType](#) **FootingType** [long **Index**] *Type of pad footing if it has been defined in AxisVM*

long **IndexOfUID** [long **UID**] *node support index of node support UID*

long **LineId** [long **Index**] *(if SupportType is nstNodalEdgeRelative) edge index*

BSTR **Name** [long **Index**] *node support name*

long **NodeId** [long **Index**] *node index*

long **ReferencId** [long **Index**] *(if SupportType is nstNodalReference) a reference index*

long **SelCount** *Get number of selected nodal supports in the model*

[ELongBoolean](#) **Selected** [long **Index**] • *Get or set the selection status of a node support*

NOTE: Call [Refresh](#) function afterwards if not called between functions

[BeginUpdate](#) and [EndUpdate](#)

RNodalSupportSpringParams **SpringParams** [long **Index**] *spring characteristics*
ENodalSupportType **SupportType** [long **Index**] *support type*
 long **SurfaceId1** [long **Index**] *(if SupportType is nstNodalEdgeRelative) 1st connecting surface*
 long **SurfaceId2** [long **Index**] *(if SupportType is nstNodalEdgeRelative) 2nd connecting surface*
 long **UID** [long **Index**] *unique index of the node support*

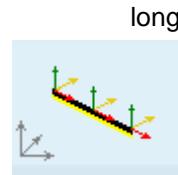
IAxisVMMembersSupports

New interface for member supports

Error codes

enum EMembersSupportsError = {	
mseSectionIdOutOfBounds = -100001	section index is out of range
msePadFootingNotDefined = -100002	not applicable, pad footing is not defined
mseInvalidType = -100003	not applicable, invalid type of support
mseStiffnessCalcParamsNotDefined = -100004	not applicable, stiffness calculation parameters are not defined
mseInvalidRefType = -100005 }	ref. type is invalid

Functions



long **AddMembersSupport** ([in] long MemberID,
[i/o] RStiffnessesXYZ StiffnessesXYZ, [i/o] RNonLinearityXYZ NonlinearityXYZ,
[i/o] RResistancesXYZ ResistancesXYZ)

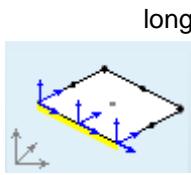
MemberID index of the member

StiffnessesXYZ stiffnesses of the elastic foundation in local direction

NonlinearityXYZ nonlinear behaviour of the elastic foundation in local direction

ResistancesXYZ resistances for stiffness components in local direction

Adds a linear support to a member in members's local coordination system. If successful, returns the member support index, otherwise returns an error code ([mseInvalidType](#), if the line is not a beam/rib or [errDatabaseNotReady](#), [errIndexOutOfBoundsException](#)).



long **AddDomainEdgeSupport** ([in] long MemberID, [in] ELineSupportType SupportType,
[i/o] RStiffnesses Stiffnesses, [i/o] RNonLinearity Nonlinearity,
[i/o] RResistances Resistances, [in] long Edgeld, [in] long Surfaceld1,
[in] long Surfaceld2, [in] long DomainId1, [in] long DomainId2)

MemberID index of the member

SupportType type of the member support

Stiffnesses stiffness components in the local system of the edge

Nonlinearity nonlinear behaviour of the support

Resistances resistances for stiffness components

Surfaceld1 first connecting surface

(0 < Surfaceld1 ≤ [AxisVMSurfaces.Count](#))

Surfaceld2 second connecting surface, can be 0

(0 < Surfaceld2 ≤ [AxisVMSurfaces.Count](#))

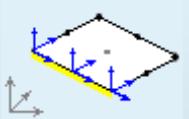
DomainId1 first connecting domain

(0 < DomainId1 ≤ [AxisVMDomains.Count](#))

DomainId2 second connecting domain, can be 0

(0 < DomainId2 ≤ [AxisVMDomains.Count](#))

Adds an domain edge support. Attached surfaces and domains are stored so that if all of them are deleted the support is deleted as well. If only one attached surface or domain is specified (other surface/domain indexes are zero) the local x direction is along the edge, the local y direction is in the plane of the domain and perpendicular to the edge, the orientation of the local z direction depends on the local z direction of the domain. If two indexes 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 an error code ([errDatabaseNotReady](#), [errIndexOutOfBoundsException](#)).

 long	AddDomainEdgeRefSupport ([in] long MemberID, [in] long ReferenceID, [in] ELineSupportType SupportType, [i/o] RStiffnesses Stiffnesses, [i/o] RNonLinearity Nonlinearity, [i/o] RResistances Resistances, [in] long EdgeID, [in] long SurfaceId1, [in] long SurfaceId2, [in] long DomainId1, [in] long DomainId2)
	MemberID index of the member ReferenceID index of the reference used as loc. z vector SupportType type of the member support Stiffnesses stiffness components in the local system of the edge Nonlinearity nonlinear behaviour of the support Resistances resistances for stiffness components SurfaceId1 first connecting surface $(0 < SurfaceId1 \leq \text{AxisVMSurfaces.Count})$ SurfaceId2 second connecting surface, can be 0 $(0 < SurfaceId2 \leq \text{AxisVMSurfaces.Count})$ DomainId1 first connecting domain $(0 < DomainId1 \leq \text{AxisVMDomains.Count})$ DomainId2 second connecting domain, can be 0 $(0 < DomainId2 \leq \text{AxisVMDomains.Count})$
	<i>Adds an domain edge support where loc. z vector of the support is set by reference. Attached surfaces and domains are stored so that if all of them are deleted the support is deleted as well. If only one attached surface or domain is specified (other surface/domain indexes are zero) the local x direction is along the edge, the local y direction is in the plane of the domain and perpendicular to the edge, the orientation of the local z direction depends on the local z direction of the domain. If two indexes 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 an error code (errDatabaseNotReady, errIndexOutOfBounds).</i>
long	Delete ([in] long Index) Index member support index <i>Deletes a member support. If successful, returns the number of deleted member supports, otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBounds).</i>
long	DeleteSelected <i>Deletes the selected member supports. If successful, returns the number of deleted supports, otherwise returns an error code (errDatabaseNotReady).</i>
long	GetFootingDimensions ([in] long Index, [i/o] RPadFootingDimensions Dimensions) Warning! This function has become obsolete, was superseded by GetFootingParams_V153 Index member support index Dimensions member footing dimesnions <i>Get the calculated dimensions of the footing. If successful, returns the index of the member support, otherwise returns an error code (errIndexOutOfBounds, errDatabaseNotReady).</i>
long	GetFootingParams ([in] long Index, [i/o] RPadFootingParams FootingParams) Warning! This function has become obsolete, was superseded by GetFootingParams_V153 Index member support index FootingParams pad footing parameters <i>Get the footing design calculation parametres of the support. If successful, returns the index of the line support, otherwise returns an error code(errIndexOutOfBounds, errDatabaseNotReady).</i>
long	GetFootingParams_V153 ([in] long Index, [i/o] RLinearFootingParams FootingParams) Index member support index FootingParams pad footing parameters

	<i>Get the specified and calculated footing parameters of the support. If successful, returns the index of the line support, otherwise returns an error code(errIndexOutOfBounds, errDatabaseNotReady).</i>
long	GetLineSupportIds ([in] long Index , [out] SAFEARRAY (long) * LineSupportIds) Index member support index LineSupportIds list of line support indexes IAxisVMLineSupports <i>Get line support indexes of IAxisVMLineSupports which will be deprecated. If successful, returns the number of line supports, otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBounds).</i>
long	GetNonLinearity ([in] long Index , [i/o] RNonLinearity NonLinearity) Index member support index NonLinearity member support non-linearity <i>Get the nonlinear behaviour of the member support. If unsuccessful, returns an error code(errIndexOutOfBounds, errDatabaseNotReady).</i>
long	GetResistances ([in] long Index , [i/o] RResistances Resistances) Index member support index Resistances nonlinear behaviour of support components <i>Get the resistance components of the member support. If unsuccessful, returns an error code(errIndexOutOfBounds, errDatabaseNotReady).</i>
long	GetStiffnesses ([in] long Index , [i/o] RStiffnesses Stiffness) Index member support index Stiffness member support stiffness <i>Get the stiffness components of the member support. If unsuccessful, returns an error code(errIndexOutOfBounds, errDatabaseNotReady).</i>
long	GetStiffnessCalcParams ([in] long Index , [i/o] RLineSupportStiffParams Value) Index member support index StiffnessParams member support stiffnesses parameters <i>Get calculation parameters for calculating the stiffness of the member support. If unsuccessful, returns an error code(errIndexOutOfBounds, errDatabaseNotReady).</i>
long	GetSelectedItemIds ([out] SAFEARRAY (long) * ItemIds) ItemIds list of selected member supports <i>If successful, returns the number of selected elements otherwise returns an error code(errDatabaseNotReady).</i>
long	GetTrMatrix ([in] long Index , [i/o] RMatrix3x3 TrMatrix) Index member support index TrMatrix Transformation matrix of the member support <i>Gets transformation matrix of the node support. If successful, returns the index of the node support, otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBounds).</i>
long	RenameSelected ([in] long NewBase , [in] BSTR FormatStr) NewBase Start number for renaming, must be a positive number FormatStr Prefix string of the new name + "_" eg "Support_" <i>Rename selected member supports.</i> <i>Example: NewBase=100 and FormatStr= 'new_' then names are: 'new100', 'new101',...etc.</i> <i>If successful returns 1, otherwise returns an error code (errDatabaseNotReady).</i>
long	SelectAll ([in] ELongBoolean Select) Select <i>IbTrue for select or IbFalse for deselect</i> <i>Select/deselect all member supports. If successful, returns the number of selected node supports, otherwise returns an error code (errDatabaseNotReady).</i>

long	SetNonLinearity ([in] long Index , [i/o] RNonLinearity NonLinearity)
	Index member support index NonLinearity member support non-linearity <i>Set the nonlinear behaviour of the member support. If unsuccessful, returns an error code(errIndexOutOfBounds, errDatabaseNotReady).</i>
long	SetResistances ([in] long Index , [i/o] RResistances Resistances)
	Index member support index Resistances nonlinear behaviour of support components <i>Set the resistance components of the member support. If unsuccessful, returns an error code(errIndexOutOfBounds, errDatabaseNotReady).</i>
long	SetStiffnesses ([in] long Index , [i/o] RStiffnesses Stiffness)
	Index member support index Stiffness member support stiffness <i>Set the stiffness components of the member support. If unsuccessful, returns an error code(errIndexOutOfBounds, errDatabaseNotReady).</i>
long	SetStiffnessCalcParams ([in] long Index , [i/o] RLineSupportStiffParams StiffnessParams)
	Index member support index StiffnessParams member support stiffnesses parameters <i>Set calculation parameters for calculating the stiffness of the member support. If unsuccessful, returns an error code(errIndexOutOfBounds, errDatabaseNotReady).</i>

Properties

long	Count Get number of member supports in the model
long	DomainId1 [long Index] (if SupportType is nstNodalEdgeRelative) 1 st connecting domain
long	DomainId2 [long Index] (if SupportType is nstNodalEdgeRelative) 2 nd connecting domain
ELongBoolean	HasFooting [long Index] lbTrue if footing have been defined for the support
ELongBoolean	HasStiffnessCalcParam [long Index] Returns lbTrue if support has defined parameters for calculating stiffness
EPadFootingType	FootingType [long Index] Type of pad footing if it has been defined
long	IndexOfUID [long UID] member support index of node support UID
long	MemberID [long Index] member index
BSTR	Name [long Index] member support name
long	NodId [long Index] node index
long	ReferencId [long Index] (if SupportType is lstEdgeReference) a reference index
long	SectionCount [long Index , EAnalysisType AnalysisType] number of sections where support force results can be obtained
long	SelCount Get number of selected member supports in the model
ELongBoolean	Selected [long Index] • Get or set the selection status of a member support <i>NOTE: Call Refresh function afterwards if not called between functions BeginUpdate and EndUpdate</i>
ELineSupportType	SupportType [long Index] support type
long	SurfacId1 [long Index] (if SupportType is nstNodalEdgeRelative) 1 st connecting surface
long	SurfacId2 [long Index] (if SupportType is nstNodalEdgeRelative) 2 nd connecting surface
long	UID [long Index] unique index of the member support

IAxisVMDomainsSupports

Domain supports of the model.

Note:

In this interface you can access supports of the domains.

Functions

long	AddDomainsSupport ([in] long DomainId, [i/o] RStiffnessesXYZ StiffnessesXYZ , [i/o] RNonLinearityXYZ NonlinearityXYZ , [i/o] RResistancesXYZ ResistancesXYZ)	<p>DomainId index of the domain StiffnessesXYZ stiffnesses of the domain support NonlinearityXYZ nonlinear behaviour of the domain support ResistancesXYZ resistance of the domain support</p> <p>Adds a domain support. If successful, returns number of domain supports, otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBoundsException).</p>
long	Delete ([in] long Index)	<p>Index domain support index</p> <p>Deletes a domain support. If successful, returns the number of deleted domain supports, otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBoundsException).</p>
long	DeleteSelected	<p>Deletes the selected domain supports. If successful, returns the number of deleted supports, otherwise returns an error code (errDatabaseNotReady).</p>
long	GetNonLinearity ([in] long Index, [i/o] RNonLinearity NonLinearity)	<p>Index domain support index NonLinearity domain support non-linearity</p> <p>Get the nonlinear behaviour of the domain support. If unsuccessful, returns an error code(errIndexOutOfBoundsException, errDatabaseNotReady).</p>
long	GetResistances ([in] long Index, [i/o] RResistances Resistances)	<p>Index domain support index Resistances nonlinear behaviour of support components</p> <p>Get the resistance components of the member support. If unsuccessful, returns an error code(errIndexOutOfBoundsException, errDatabaseNotReady).</p>
long	GetStiffnesses ([in] long Index, [i/o] RStiffnesses Stiffness)	<p>Index domain support index Stiffness domain support stiffness</p> <p>Get the stiffness components of the domain support. If unsuccessful, returns an error code(errIndexOutOfBoundsException, errDatabaseNotReady).</p>
long	GetStiffnessCalcParams ([in] long Index, [i/o] RLineSupportStiffParams Value)	<p>Index domain support index StiffnessParams domain support stiffnesses parameters</p> <p>Get calculation parameters for calculating the stiffness of the domain support. If unsuccessful, returns an error code(errIndexOutOfBoundsException, errDatabaseNotReady).</p>
long	GetSelectedItemIds ([out] SAFEARRAY (long) * ItemIds)	<p>ItemIds list of selected domain supports</p> <p>If successful, returns the number of selected supports otherwise returns an error code(errDatabaseNotReady).</p>
long	RenameSelected ([in] long NewBase, [in] BSTR FormatStr)	<p>NewBase Start number for renaming, must be a positive number FormatStr Prefix string of the new name + "_" eg "Support_"</p> <p>Rename selected domain supports.</p>

*Example: NewBase=100 and FormatStr= 'new_' then names are: 'new100', 'new101',...etc.
If successful returns 1, otherwise returns an error code ([errDatabaseNotReady](#)).*

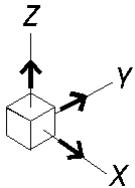
long	SelectAll ([in] ELongBoolean Select) Select <i>IbTrue for select or IbFalse for deselect</i> <i>Select/deselect all domain supports. If successful, returns the number of selected domain supports, otherwise returns an error code (errDatabaseNotReady).</i>
long	SetNonLinearity ([in] long Index , [i/o] RNonLinearity NonLinearity) Index <i>domain support index</i> NonLinearity <i>domain support non-linearity</i> <i>Set the nonlinear behaviour of the member support. If unsuccessful, returns an error code (errIndexOutOfBounds, errDatabaseNotReady).</i>
long	SetResistances ([in] long Index , [i/o] RResistances Resistances) Index <i>domain support index</i> Resistances <i>nonlinear behaviour of support components</i> <i>Set the resistance components of the domain support. If unsuccessful, returns an error code (errIndexOutOfBounds, errDatabaseNotReady).</i>
long	SetStiffnesses ([in] long Index , [i/o] RStiffnesses Stiffness) Index <i>domain support index</i> Stiffness <i>domain support stiffness</i> <i>Set the stiffness components of the domain support. If unsuccessful, returns an error code (errIndexOutOfBounds, errDatabaseNotReady).</i>

Properties

long	Count <i>Get number of domain supports in the model</i>
long	DomainID [long Index] <i>domain index</i>
long	IndexOfUID [long UID] <i>domain support index of domain support UID</i>
BSTR	Name [long Index] <i>domain support name</i>
long	SelCount <i>Get number of selected domain supports in the model</i>
ELongBoolean	Selected [long Index] • <i>Get or set the selection status of a domain support</i> <i>NOTE: Call Refresh function afterwards if not called between functions BeginUpdate and EndUpdate</i>
long	SurfaceId1 [long Index] <i>(if SupportType is nstNodalEdgeRelative) 1st connecting surface</i>
long	SurfaceId2 [long Index] <i>(if SupportType is nstNodalEdgeRelative) 2nd connecting surface</i>
long	UID [long Index] <i>unique index of the domain support</i>

IAxisVMNodes

Nodes in the model.



Enumerated types

```
enum EDegreeOfFreedom = {  
    dofFree = 0x00  
    dofXfix = 0x01,  
    dofYfix = 0x02,  
    dofZfix = 0x04,  
    dofXXfix = 0x08,  
    dofYYfix = 0x10,  
    dofZZfix = 0x20,  
    dofTrussAndMembraneXZ = 0x3A,  
    dofFrameYZ = 0x31,  
    dofFrameXZ = 0x2A,  
    dofFrameXY = 0x1C,  
    dofPlateYZ = 0x0E,  
    dofPlateXZ = 0x15,  
    dofPlateXY = 0x23}  
  
no constraint (free node)  
no displacement in X direction  
no displacement in Y direction  
no displacement in Z direction  
no rotation about X direction  
no rotation about Y direction  
no rotation about Z direction  
  
 $dofYfix+dofXXfix+dofYYfix+dofZZfix = 2+8+16+32 = 58$   
 $dofXfix+dofYYfix+dofZZfix = 1+16+32 = 49$   
 $dofYfix+dofXXfix+dofZZfix = 2+8+32 = 42$   
 $dofZfix+dofXXfix+dofYYfix = 4+8+16 = 28$   
 $dofYfix+dofZfix+dofXXfix = 2+4+8 = 14$   
 $dofXfix+dofZfix+dofYYfix = 1+4+16 = 21$   
 $dofXfix+dofYfix+dofZZfix = 1+2+32 = 35$ 
```

Degrees of freedom (DOF). Combined constraints by adding the constants.

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, number can correspond to any EDegreeOfFreedom type or custom combination  
)
```

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 an existing node is closer than **IAxisVMSettings.EditingTolerance** no new node is created just the degrees of freedom assigned to the existing node is set to **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, number can correspond to any *EDegreeOfFreedom* type or custom combination

Adds a node to the model with the specified degrees of freedom. If an existing node is closer than *IAxisVMSettings.EditingTolerance* no new node is created just the degrees of freedom assigned to the existing node is set to dof. If successful, returns node index otherwise returns an error code ([errDatabaseNotReady](#)).

long **BulkAdd** ([in] SAFEARRAY(RPoint3d) **Coords**, [out] SAFEARRAY(long)* **Results**)

Coords Array of node coordinates.

Results The indexes of the created nodes. Each index will satisfy : (1 ≤ Index ≤ *AxisVMNodes.Count*)

Creates new nodes. The degree of freedom of the newly created nodes will be dofFree. Use this function if a large number of nodes are to be created, it will be significantly faster than creating each node individually with Add. If there is an existing node closer than *IAxisVMSettings.EditingTolerance*, no new node will be created for that coordinate. The Results will contain the indexes of the new and/or existing nodes corresponding to the coordinates. The return value on success is the number of nodes created. If an error code is returned, then the content of Results is empty. For modifying the coordinates of already existing nodes use the BulkSetNodeCoord instead. The possible error codes are([errDatabaseNotReady](#), [errInternalException](#), [errOutOfMemory](#)).

long **BulkGetCoord** ([in] SAFEARRAY(long) **Indexes**, [out] SAFEARRAY(RPoint3d)* **Coords**)

Indexes Array of node indexes. Each index must satisfy : (1 ≤ Index ≤ *AxisVMNodes.Count*)

Coords The returned coordinates of the nodes.

Queries the coordinates of the nodes in the Indexes list. Use this function if a large number of nodes are to be queried, it will be significantly faster than calling GetNodeCoord for each node individually. The return value on success is the number of coordinates queried. If an error code is returned, then the content of Coords is empty. The possible error codes are([errDatabaseNotReady](#), [errInternalException](#), [errIndexOutOfBoundsException](#), [errOutOfMemory](#)).

long **BulkGetDOF** ([in] SAFEARRAY(long) **Indexes**, [out] SAFEARRAY(long)* **DOFs**)

Indexes Array of node indexes. Each index must satisfy : (1 ≤ Index ≤ *AxisVMNodes.Count*)

DOFs The returned degrees of freedom of the nodes. A value is a combination of basic *EDegreeOfFreedom* values.

Queries the degrees of freedom of the nodes in the Indexes list. The return value on success is the number of degrees of freedom queried. If an error code is returned, then the content of DOFs is empty. The possible error codes are([errDatabaseNotReady](#), [errInternalException](#), [errIndexOutOfBoundsException](#), [errOutOfMemory](#)).

long **BulkSetDOF** ([in] SAFEARRAY(long) **Indexes**, [in] SAFEARRAY(long)* **DOFs**)

Indexes Array of node indexes. Each index must satisfy : (1 ≤ Index ≤ *AxisVMNodes.Count*)

DOFs The degrees of freedom of the nodes. A value is a combination of basic *EDegreeOfFreedom* values.

Sets the degrees of freedom of the nodes in the Indexes list. The return value on success is the number of degrees of freedom set. If the length of Indexes isn't equal to the length of Dofs, an [errIndexOutOfBoundsException](#) will result. The possible error codes are([errDatabaseNotReady](#), [errInternalException](#), [errIndexOutOfBoundsException](#), [errOutOfMemory](#)).

long **BulkSetNodeCoord** ([in] SAFEARRAY(long) **Indexes**, [in] SAFEARRAY(RPoint3d)* **Coords**)

Indexes Array of node indexes. Each index must satisfy : (1 ≤ Index ≤ *AxisVMNodes.Count*)

Coords Array of node coordinates.

Changes the coordinates of the nodes in the Indexes list. The return value on success is the number of changed coordinates. If the length of Indexes isn't equal to the length of Coords, an [errIndexOutOfBoundsException](#) will result. For creating new nodes use the BulkAdd function instead. The possible error codes are([errDatabaseNotReady](#), [errInternalException](#), [errIndexOutOfBoundsException](#), [errOutOfMemory](#)).

long **Check** ([in] double **Eps**, [in] ELongBoolean **Delete**, [in] ELongBoolean **Repaint**)

	Eps	Maximum projected distance (dx, dy or dz) between nodes [m]
	Delete	If true deletes nodes, which are within close proximity except node with the lowest number. If false, then only selects nodes, which are within close proximity, while node with smallest number is also selected.
	Repaint	Redraw and reconnect nodes in the model.
		Finds and selects or deletes nodes, which are within close proximity. If nodes within close proximity found, returns 1, 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. $1 \leq Index \leq Count$. 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	DeleteNameOfAllNodes	Deletes previously added names. If successful, returns 1, otherwise returns an error code (errDatabaseNotReady).
long	GetConnectedLines ([in] long Index, [out] SAFEARRAY(long)* LinIdxList)	<p>Index node index LinIdxList line indexes according to AxisVMLines</p> <p>Obtains indexes of lines connecting to the node. $1 \leq Index \leq Count$. If successful, returns the number of lines connecting to the node otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBounds).</p>
long	GetConnectedSurfaces ([in] long Index, [out] SAFEARRAY(long)* SurfacIdxList)	<p>Index node index SurfacIdxList surface indexes according to AxisVMSurfaces</p> <p>Obtains indexes of surface elements connecting to the node. $1 \leq Index \leq Count$. If successful, returns the number of surface elements connecting to the node otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBounds).</p>

long	GetNode ([in] long Index, [i/o] RNode Value)
	Index node index
	Get a node by index. 1 ≤ Index ≤ Count.
	If unsuccessful, returns an error code (errDatabaseNotReady , errIndexOutOfBounds).
long	GetNodeLines ([in] long Nodeld, [out] SAFEARRAY (long) * LineIds)
	Nodeld node index. It must satisfy: (1 ≤ Nodeld ≤ AxisVMNodes.Count)
	LineIds list of lines having Nodeld as a start or end node. Each line index will satisfy: (1 ≤ Index ≤ AxisVMLines.Count)
	Retrieves the lines having NodeID as a start or end node. If successful, returns the number of attached lines. The possible error codes are (errDatabaseNotReady , errIndexOutOfBounds , errInternalException , errOutOfMemory).
long	GetSelectedItemIds ([out] SAFEARRAY (long) * ItemIds)
	ItemIds list of selected nodes
	If successful, returns the number of selected elements otherwise returns an error code (errDatabaseNotReady).
long	GetNodeCoord ([in] long Index, [i/o] RPoint3d Value)
	Index node index
	Value node coordinates
	If successful, returns the node index, otherwise returns an error code (errDatabaseNotReady , errIndexOutOfBounds).
long	GetSelectedNodes ([out] SAFEARRAY(long)* ItemIds, [out] SAFEARRAY(RNode)* Items)
	ItemIds list of indexes of selected nodes
	Items list of coordinates of selected nodes
	If successful, returns the number of selected nodes, otherwise returns an error code (errDatabaseNotReady , errCOMServerInternalError).
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 ≤ StartIndex ≤ Count
	Finds a node by its coordinates. If successful, returns the node index otherwise returns an error code (errDatabaseNotReady , errIndexOutOfBounds , errNotFound).
long	RemoveSelectedIntermedNodes ()
	If successful, returns the number of selected nodes, otherwise returns an error code (errDatabaseNotReady).
long	RenameSelectedNodes ([in] long NewBase, [in] BSTR FormatStr)
	NewBase Start number for renaming, must be a positive number
	FormatStr Prefix string of the new name + "_" eg "Node_"
	Example: NewBase=100 and FormatStr= 'new_' then names are: 'new100', 'new101',...etc.
	If successful, returns the number of selected nodes, otherwise returns an error code (errDatabaseNotReady).
long	SelectAll ([in] ELongBoolean 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).
	NOTE: Call Refresh function afterwards if not called between functions BeginUpdate and EndUpdate

long **SetNode** ([in] long **Index**, [i/o] RNode **Value**)
 Index node index
 Set a node by index. $1 \leq \text{Index} \leq \text{Count}$.
 If unsuccessful, returns an error code ([errDatabaseNotReady](#), [errIndexOutOfBounds](#)).

long **SetNodeCoord** ([in] long **Index**, [i/o] **RPoint3d Value**)
 Index node index
 Value node coordinates
If successful, returns the node index, otherwise returns an error code ([errDatabaseNotReady](#), [errIndexOutOfBounds](#)).

Properties

AxisVMAttachments *	Attachments Get the attachments interface
AxisVMAtributes *	Attributes Get the attributes interface
long	Count Get number of nodes in the model
long	IndexOfUID [long UID] Get index of the node
	UID unique index of the node
BSTR	Name [long Index] • Get the name of the node
	Index index of the node
ELongBoolean	Selected [long Index] • Get or set the selection status of a node
	Index index of the node
	<i>NOTE: Call Refresh function afterwards if not called between functions BeginUpdate and EndUpdate</i>
long	SelCount Get number of selected nodes in the model
long	UID [long Index] Get unique index of the node which remains the same while exists in the model
	Index index of the node

IAxisVMPushoverHingeFunctions



Interface used for setting hinge functions for pushover. If property returning this interface is null (nil) then the extension module SE2 is not available.

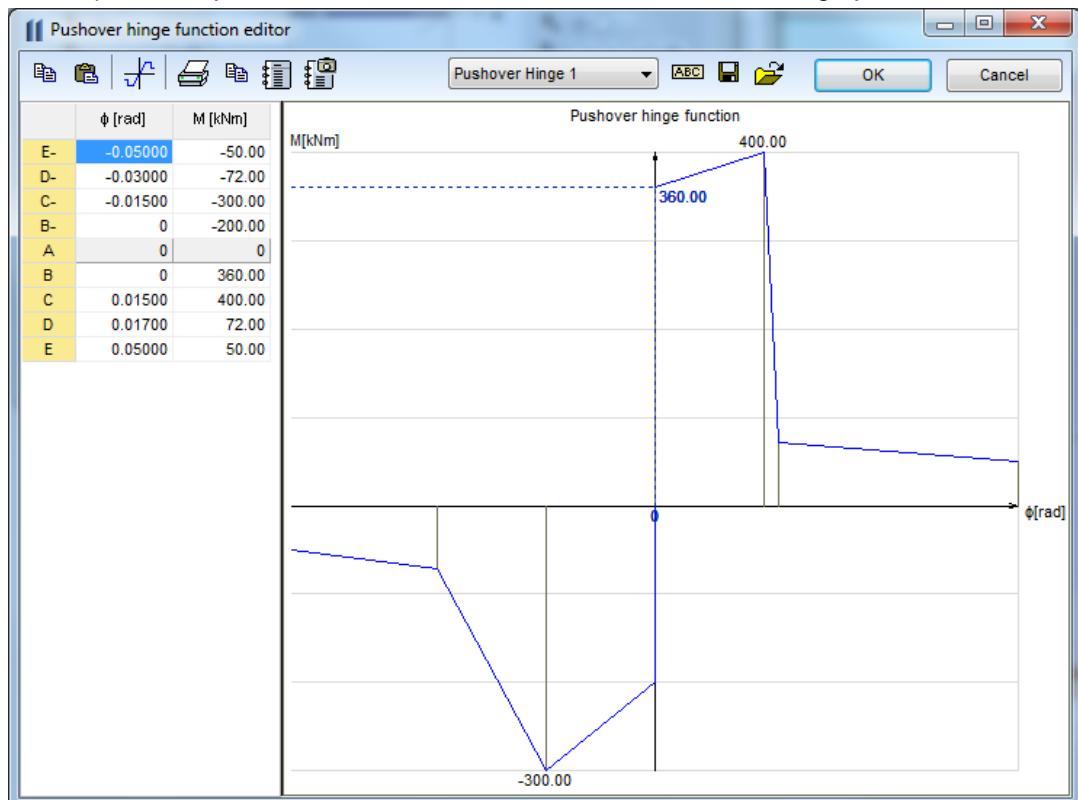
Functions

long **Add** ([in] BSTR Name, [in] SAFEARRAY(RPoint2D)* FunctionPoints)

Name name of the new pushover hinge function

FunctionPoints Function points of the pushover hinge function

Adds a new pushover hinge function. Coord1 is rotation [rad] and Coord2 is moment capacity [kNm]. Must have 9 points in total where first 4 point coordinates (Coord1 and Coord2) must be negative. 5th point must be [0,0] and last 4 point coordinates (Coord1 and Coord2) must be positive. Points in order from E- to E, see table and graph:



If successful, returns index of the new pushover hinge function, otherwise an error code ([fueNameAlreadyExists](#), [fueInvalidFunction](#), [errDatabaseNotReady](#), [errCOMServerInternalError](#)).

long **Add_vb** (Visual Basic compatible function of Add)

long **AddFromFile** ([in] BSTR Name, [in] BSTR FileName)

Name Name of the pushover hinge function

FileName Name of the file containing pushover hinge function

If successful, returns index of the new pushover hinge function, otherwise an error code ([errDatabaseNotReady](#), [fueFailedToAddFromFile](#)).

long **Clear**

Deletes all pushover hinge functions.

If successful, returns 1, otherwise an error code ([errDatabaseNotReady](#)).

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

index pushover hinge function index

If successful, returns number of pushover hinge functions after delete, otherwise an error code ([errDatabaseNotReady](#) or [errIndexOutOfBounds](#)).

long	GetPoints ([in] long index , [out] SAFEARRAY(RPoint2d)* FunctionPoints)
	index pushover hinge function index
	FunctionPoints point array of the pushover hinge function
	<i>If successful, returns number of points of the function, otherwise an error code (errDatabaseNotReady, errIndexOutOfBounds)</i>
long	IndexOf ([in] BSTR Name)
	Name Name of the pushover hinge function
	<i>If successful, returns pushover hinge function index, otherwise an error code (errDatabaseNotReady)</i>
long	Modify ([in] long index , [in] SAFEARRAY(RPoint2d)* FunctionPoints)
	index pushover hinge function index
	FunctionPoints point array of the pushover hinge function, see function Add for more info about point coordinates.
	<i>If successful, returns function index, otherwise an error code (errDatabaseNotReady, errIndexOutOfBounds, fueFailedToModifyFunction or errCOMServerInternalError)</i>
long	Modify_vb (Visual Basic compatible function of Modify)
long	SaveToFile ([in] long index , [in] BSTR FileName)
	index pushover hinge function index
	FileName Name of the file used for saving without path
	<i>Saves function to AxisVM directory \pohinge with file extension: poh</i>
	<i>If successful, returns pushover hinge function index, otherwise an error code (errDatabaseNotReady, errIndexOutOfBounds or fueFileExists)</i>

Properties

long	Count Get number of pushover hinge functions.
BSTR	Name [long Index] Get or set name of the pushover hinge function with index Index .
long	PointCount [long Index] Get number of points of the pushover hinge function with index Index .

IAxisVMRCBeamDesign

Interface used for setting design parameters and reading results of RC beam design.
If property returning this interface is null (nil) then the extension module RC2 is not available.
Only one instance of this interface is allowed. Free it before creating new AxisVM model.

IMPORTANT NOTE: Interface functions must be called in this order:

1. AddLines (for lines) or AddMembers (for members)
2. SetDesignParameters
3. Calculate

Error codes

enum	ERCBeamDesignError = {	
	rcbdePrestressedBeamsNotSupported = -100001	Prestressed beams not supported in RC design
	rcbdeErrorSettingLines = -100002	Lines can not be added together
	rcbdeInvalidLoadCaseId = -100003	Load case index invalid
	rcbdeInvalidLoadCombinationId = -100004	Load combination index invalid
	rcbdeInvalidCombinationOfLoadCaseAndLoadLevel = -100005	Invalid combination of load case and load level
	rcbdeInvalidCombinationOfLoadCombinationAndLoadLevel = -100006	Invalid combination of load combination and load level
	rcbdeInvalidMaterialId = -100007	Material index invalid
	rcbdeInvalidArrayLength = -100008	Length of arrays don't match or invalid
	rcbdeInvalidAnalysisType = -100009	Invalid type of analysis
	rcbdeInvalidValue_bw = -100010	bw value is invalid
	rcbdeInvalidValue_h = -100011	h value is invalid
	rcbdeInvalidValue_hf = -100012	hf value is invalid
	rcbdeInvalidValue_beff = -100013	beff value is invalid
	rcbdeInvalidRebarMaterial = -100014	Rebar material is invalid
	rcbdeInvalidStirrupMaterial = -100015	stirrup material is invalid
	rcbdeInvalidStirrupDiameter = -100016	stirrup diameter is invalid
	rcbdeInvalidStirrupLegs = -100017	Number of stirrup legs is invalid
	rcbdeInvalidBottomPos = -100018	Bottom position of rebar is invalid
	rcbdeInvalidTopPos = -100019	Top position of rebar is invalid
	rcbdeInvalidAst_min = -100020	Minimum area of top reinforcement is invalid
	rcbdeInvalidAsb_min = -100021	Minimum area of bottom reinforcement is invalid
	rcbdeErrorSettingMembers = -100022	Members can not be added together
	rcbdeInvalidThetaValue = -100023	Theta value is out of valid range
	rcbdeDesignCodeParametersNotValidForUsedDesignCode = -100024	used parameter record type is not valid for current design code
	rcbdeEnvironmentClassNotValidForUsedDesignCode = -100025	specified environment class is not valid for current design code
	rcbdeInvalidEnvelopeID = -100026	Invalid EnvelopeID
	rcbdeInvalidShrinkageValue = -100027	shrinkage strain is invalid (it must be positive)
	rcbdeInvalidDesignParameters = -100028	at least of the given design parameters is invalid
	rcbdeInvalidPlasticHingeParams = -100029 }	plastic hinge's parameter is invalid

Design message codes

RC Beam design messages

2101	Compression reinforcement is needed
2215, 2225, 2245	Shear reinforcement is not needed
2501, 2502	The cross-section is unsatisfactory for bending
2615, 2625, 2627, 2628, 2645	The cross-section is unsatisfactory for shear
2626	Cracking analysis is necessary

Enumerated types

enum	ERCBeamShape = {	
	rcbsRectangle = 0,	beam with rectangular shape
	rcbsDownStand = 1,	down stand beam
	rcbsUpStand = 2 }	up stand beam
	Shape of beam	

enum	ERCBeamDesignPlane = {	
	rcbdpQzMy = 0,	design plane in local x,z plane of the beam
	rcbdpQyMz = 1 }	design plane in local x,y plane of the beam
	Design plane for beam design	

```

enum ERCBeam_EC_SIA_SeismicZone = {
    rcbsecSeismicH = 0,      see AxisVM manual RC beam design
    rcbsecSeismicM = 1,      see AxisVM manual RC beam design
    rcbsecAntiSeismic = 2   see AxisVM manual RC beam design
}

enum ERCBeam_ECRO_STAS_SeismicZone = {
    rcbssSeismicH = 0,      see AxisVM manual RC beam design
    rcbssSeismicM = 1,      see AxisVM manual RC beam design
    rcbsszAntiSeismic = 2,  see AxisVM manual RC beam design
    rcbsszSeismicH = 3,      see AxisVM manual RC beam design
    rcbsszSeismicM = 4}     see AxisVM manual RC beam design
Seismic zone in accordance with Romaian STAS design code

```

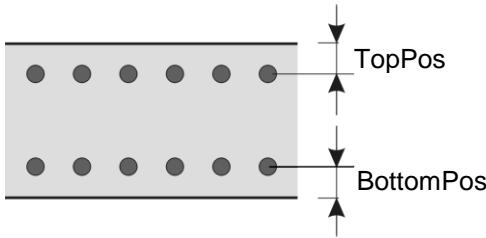
Records / structures

```

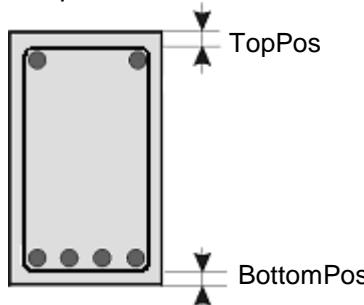
RPartialRCBeamDesignParameters = (
    RRCBeamCrossSections          RC cross section at start and end
    RRCBeamSupports                RC beam support at start and end
)
Partial RC Beam design parameters

RRCBeamDesignParameters = (
    long ConcreteMaterial           index of the concrete material
    double Dmax                  max. size of the aggregate
    long RebarMaterial            index of the rebar material
    long StirrupMaterial          index of the stirrup material
    double StirrupDiameter        stirrup diameter
    long StirrupLegs              number of stirrup legs
    ERCBeamShape                Shape                      shape of the beam
    double c_bottom              cover of bottom longitudinal bars
    double c_top                 cover of top longitudinal bars
    double ds_bottom              diameter of bottom longitudinal bars
    double ds_top                diameter of top longitudinal bars
    ELongBoolean               TakeConcTensileStrengthNL  take tensile strength of the concrete into account in nonlinear analysis
    EBoolean                  UsefcmtfNL                consider flexural tensile strength in nonlinear analysis instead of tensile strength (only for EC and ITA)
    double ShrinkageEpsNL         shrinkage strain considered in nonlinear analysis
)
General RC beam reinforcement parameters .

```



In national design codes MSZ and STAS
c_bottom and **c_top** indicates position of the rebar



In all other national design codes **c_bottom** and **c_top** indicates cover to the shear links

RRCBeamPlasticHingeParams = (

EBoolean	Active	<i>plastic hinge is active</i>
EBoolean	AppliedReinforcement	<i>use applied reinforcement</i>
double	As_Bottom	<i>area of reinforcement at the bottom</i>
double	As_Top	<i>area of reinforcement at the top</i>
double	Depth_Bottom	<i>depth of reinforcement at the bottom</i>
double	Depth_Top	<i>depth of reinforcement at the top</i>

)

RRCBeamPlasticHinges = (

EBoolean	EnablePlasticHinges	<i>consider plastic hinges</i>
RRCBeamPlasticHingeParams	Hinge1	<i>parameters of hinge 1</i>
RRCBeamPlasticHingeParams	Hinge2	<i>parameters of hinge 2</i>
double	Pos_Hinge1	<i>position of center of the hinge 1</i>
double	Pos_Hinge2	<i>position of center of the hinge 2</i>
double	MinRebarDiameter	<i>minimal diameter among the considered rebars</i>
double	GammaRd	<i>safety factor</i>

)

RRCBeamDesignParameters_EC = (

EBoolean	VariableAngleTrussMethod	<i>variable or fixed (at 45 deg.) strut angle</i>
double	Theta	<i>angle of the concrete compression strut Θ (in rads.), must be set if VariableAngleTrussMethod = lbFalse</i>
double	fse	<i>load factor for seismic forces (default is 1 =>no change)</i>
EBoolean	ApplyMinimumCover	<i>Calculate concrete covers to all reinforcements based on environment and structural class</i>
EBoolean	CrackWidthCheck	<i>Increase reinforcement to limit crack width to MaxCrackWidth_Bottom and MaxCrackWidth_Top values</i>
double	MaxCrackWidth_Bottom	<i>max. allowed crack width at top</i>
double	MaxCrackWidth_Top	<i>max. allowed crack width at bottom</i>
EBoolean	TakeConcTensileStrength	<i>Take tensile strength of the concrete into account</i>
EBoolean	ShortTerm	<i>See RC beam design in AxisVM manual</i>
double	Deflection_Beam_L_div	<i>cantilever deflection limit L/x, see RC beam design in AxisVM manual</i>
double	Deflection_Cantilever_L_div	<i>beam deflection limit L/x, see RC beam design in AxisVM manual</i>
EEnvironmentClass	TopSurface	<i>environment class at top</i>
EEnvironmentClass	BottomSurface	<i>environment class at bottom</i>
EStructClass_EC	StructClass	<i>structural class</i>
ERCBeam EC_SIA SeismicZone	SeismicZone	<i>Seismic zone</i>
ne	PlasticHinges	<i>parameters of plastic hinges at the ends of the beam</i>

)

Beam reinforcement parameters according to the Euro code.

RRCBeamDesignParameters_EC_RO = (

ERCBeam_ECRO STAS_Sea	Ksi_RO	$\xi_{c0} = x_0 / d$, for more info see national design code (STAS)
micZone	SeismicZone	<i>Seismic zone in accordance with Romaian STAS design code</i>
double	fse	<i>load factor for seismic forces (default is 1 =>no change)</i>
EBoolean	VariableAngleTrussMethod	<i>variable or fixed (at 45 deg.) strut angle</i>
double	Theta	<i>angle of the concrete compression strut Θ (in rads.), must be set if VariableAngleTrussMethod = lbFalse</i>
double	Deflection_Beam_L_div	<i>cantilever deflection limit L/x, see RC beam design in AxisVM manual</i>
double	Deflection_Cantilever_L_div	<i>beam deflection limit L/x, see RC beam design in AxisVM manual</i>
EBoolean	CrackWidthCheck	<i>Increase reinforcement to limit crack width to MaxCrackWidth_Bottom and MaxCrackWidth_Top values</i>
double	MaxCrackWidth_Bottom	<i>max. allowed crack width at top</i>
double	MaxCrackWidth_Top	<i>max. allowed crack width at bottom</i>
EBoolean	TakeConcTensileStrength	<i>Take tensile strength of the concrete into account</i>
EBoolean	ShortTerm	<i>See RC beam design in AxisVM manual</i>
RRCBeamPlasticHinges	PlasticHinges	<i>parameters of plastic hinges at the ends of the beam</i>

)

Beam reinforcement parameters according to the Romanian Euro code.

		RRCBeamDesignParameters_ITA = (
EBoolean	double	VariableAngleTrussMethod
		Theta variable or fixed (at 45 deg.) strut angle
	double	fse angle of the concrete compression strut Θ (in rads.), must be set if VariableAngleTrussMethod = lbFalse
		load factor for seismic forces (default is 1 =>no change)
EBoolean		CrackWidthCheck Increase reinforcement to limit crack width to MaxCrackWidth_Bottom and MaxCrackWidth_Top values
	double	MaxCrackWidth_Bottom max. allowed crack width at top
	double	MaxCrackWidth_Top max. allowed crack width at bottom
EBoolean		TakeConcTensileStrength Take tensile strength of the concrete into account
EBoolean		ShortTerm See RC beam design in AxisVM manual
	double	Deflection_Beam_L_div cantilever deflection limit L/x, see RC beam design in AxisVM manual
	double	Deflection_Cantilever_L_div beam deflection limit L/x, see RC beam design in AxisVM manual
ERCBeam_EC_SIA_SeismicZone		SeismicZone Seismic zone
RRCBeamPlasticHinges		PlasticHinges parameters of plastic hinges at the ends of the beam
)
		Beam reinforcement parameters according to the Italian design code.
		RRCBeamDesignParameters_SIA = (
EBoolean		ApplyMinimumCover Calculate concrete covers to all reinforcements based on environment and structural class
EnvironmentClass		TopSurface environment class at top
EnvironmentClass		BottomSurface environment class at bottom
	double	Deflection_Beam_L_div cantilever deflection limit L/x, see RC beam design in AxisVM manual
	double	Deflection_Cantilever_L_div beam deflection limit L/x, see RC beam design in AxisVM manual
EBoolean	double	VariableAngleTrussMethod variable or fixed (at 45 deg.) strut angle
		Theta angle of the concrete compression strut Θ (in rads.), must be set if VariableAngleTrussMethod = lbFalse
EBoolean		CrackWidthCheck Increase reinforcement to limit crack width to MaxCrackWidth_Bottom and MaxCrackWidth_Top values
	double	MaxCrackWidth_Bottom max. allowed crack width at top
	double	MaxCrackWidth_Top max. allowed crack width at bottom
EBoolean		TakeConcTensileStrength Take tensile strength of the concrete into account
ERCBeam_EC_SIA_SeismicZone		SeismicZone Seismic zone
RRCBeamPlasticHinges		PlasticHinges parameters of plastic hinges at the ends of the beam
)
		Beam reinforcement parameters according to the Swiss design code SIA.
		RRCBeamDesignParameters_STAS = (
	double	mbc concrete's compression capacity factor, see Romanian design code (STAS)
	double	mbt concrete's tension capacity factor, see Romanian design code (STAS)
	double	ksi0 $\xi_{c0} = x_0 / d$, for more info see national design code (STAS)
ERCBeam_ECRO_STAS_SeismicZone		SeismicZone Seismic zone in accordance with Romaian STAS design code
	double	fse load factor for seismic forces (default is 1 =>no change)
RRCBeamPlasticHinges		PlasticHinges parameters of plastic hinges at the ends of the beam
)
		Beam reinforcement parameters according to the STAS national design code.

RRCBeamDesignParameters_DIN = (

<u>EEnvironmentClass</u>	EnvironmentClass	<i>environment class</i>
double	Deflection_Beam_L_div	<i>cantilever deflection limit L/x, see RC beam design in AxisVM manual</i>
double	Deflection_Cantilever_L_div	<i>beam deflection limit L/x, see RC beam design in AxisVM manual</i>
<u>EBoolean</u>	VariableAngleTrussMethod	<i>variable or fixed (at 45 deg.) strut angle</i>
double	Theta	<i>angle of the concrete compression strut Θ (in rads.), must be set if VariableAngleTrussMethod = lbFalse</i>
<u>EBoolean</u>	CrackWidthCheck	<i>Increase reinforcement to limit crack width to MaxCrackWidth_Bottom and MaxCrackWidth_Top values</i>
double	MaxCrackWidth_Bottom	<i>max. allowed crack width at top</i>
double	MaxCrackWidth_Top	<i>max. allowed crack width at bottom</i>
<u>EBoolean</u>	TakeConcTensileStrength	<i>Take tensile strength of the concrete into account</i>
)	

Beam reinforcement parameters according to the German design code DIN.

RRCBeamDesignParameters_MSZ = (

<u>EBoolean</u>	CrackWidthCheck	<i>Increase reinforcement to limit crack width to MaxCrackWidth_Bottom and MaxCrackWidth_Top values</i>
double	MaxCrackWidth_Bottom	<i>max. allowed crack width at top</i>
double	MaxCrackWidth_Top	<i>max. allowed crack width at bottom</i>
<u>EBoolean</u>	TakeConcTensileStrength	<i>Take tensile strength of the concrete into account</i>
<u>EBoolean</u>	AutoPsi	<i>See RC beam design in AxisVM manual</i>
double	Deflection_Beam_L_div	<i>cantilever deflection limit L/x, see RC beam design in AxisVM manual</i>
double	Deflection_Cantilever_L_div	<i>beam deflection limit L/x, see RC beam design in AxisVM manual</i>
)	

Beam reinforcement parameters according to the Hungarian design code MSZ.

RRCBeamDesignReqReinfs = (

<u>RRCBeamDesignReqReinf</u>	Top	<i>reinforcement at top required by design</i>
<u>RRCBeamDesignReqReinf</u>	Bottom	<i>reinforcement at bottom required by design</i>
<u>RRCBeamDesignReqReinf</u>	Top_CC	<i>reinforcement at top required to limit crack width (crack control)</i>
<u>RRCBeamDesignReqReinf</u>	Bottom_CC	<i>reinforcement at bottom required to limit crack width (crack control)</i>
)	

Number of required bars at top and bottom of the cross-section

RRCBeamDesignReqReinf = (

long	N	<i>total number of bars</i>
long	Nf	<i>number of bars outside of web width (for T and I sections)</i>
long	NRow	<i>number of rows with rebar</i>
double	U	<i>distance of rebar's group center of gravity to section's edge</i>
)	

Number of rows and required bars and their position.

RRCBeamDesignCrackResults = (

<u>RRCBeamDesignCrackRe</u>	Top	<i>crack design results at top</i>
<u>sult</u>		
<u>RRCBeamDesignCrackRe</u>	Bottom	<i>crack design results at bottom</i>
<u>sult</u>)	

Crack results at top and bottom of the cross-section

RRCBeamDesignCrackResult = (

Double	Wk	<i>crack width</i>
Double	I1	<i>inertia of un-cracked cross section</i>
Double	x1	<i>distance of neutral axis of un-cracked cross section</i>
Double	I2	<i>inertia of cracked cross section (according to elastic theory)</i>
Double	x2	<i>distance of neutral axis of cracked cross section (according to elastic theory)</i>
Double	Mcr	<i>cracking moment</i>
Double	Sr_max	<i>maximum distance of cracks</i>

Details of the crack results

RPartialRCBeamDesignParameters = (

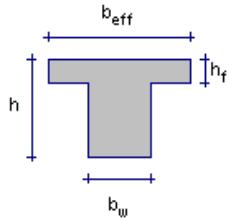
<u>RRCBeamCrossSections</u>	<i>RC cross section at start and end</i>
<u>RRCBeamSupports</u>	<i>RC beam support at start and end</i>

Partial RC Beam design parameters

RRCBeamCrossSections = (

<u>RRCBeamSection</u>	<i>section parameter at the start</i>
<u>RRCBeamSection</u>	<i>section parameters at the end</i>

Start and end RC beam cross-section parameters



RRCBeamSection = (

double	bw	<i>breadth of the beam</i>
double	h	<i>height of the section</i>
double	hf	<i>flange thickness</i>
double	beff	<i>width of the flange</i>

RC beam section parameters

RRCBeamSupports = (

<u>RRCBeamSupport</u>	<i>support parameters at the start</i>
<u>RRCBeamSupport</u>	<i>support parameters at the end</i>

RC beam's supports

NOTE:

To enable/disable shear force reduction at supports set RRCBeamSupport record accordingly and call AddMembers function.

RRCBeamSupport = (

<u>ELongBoolean</u>	OverWrite	<i>if true, overwrite automatically generated values calculated by Axis VM depending on beam and column depths</i>
double	ActualHalfWidth	<i>actual half-width of the support (from axis to the edge of the support)</i>
double	TheoreticalHalfWidth	<i>theoretical half-width of the support (from axis to the edge of the support)</i>
<u>ELongBoolean</u>	ShearReduction	<i>true if shear reduction enabled around supports. See picture</i>

Support parameters used for shear reduction

		RRCBeamDesignResult = (
	double	x <i>x coordinate of design member</i>
RRCBeamDesignBendingResult		Top <i>RC beam bending design results at top</i>
RRCBeamDesignBendingResult		Bottom <i>RC beam bending design results at bottom</i>
	double	s <i>spacing of shear links, $s=\min(s_v, s_{vmax}, s_t)$, construction criteria and torsion considered</i>
	double	sv <i>spacing of shear links for shear design only</i>
	double	smax <i>maximum spacing of shear links according to construction criteria</i>
	double	st <i>spacing of shear links for torsion only</i>
	double	Tsd <i>T_{ed}/T_{rd} utilisation (see national design code)</i>
	double	Ved_Vrd <i>V_{ed}/V_{rd} utilisation (see national design code)</i>
	double	Vsdred_Min <i>minimum reduced design shear force</i>
	double	Vsdred_Max <i>maximum reduced design shear force</i>
	double	VRdc <i>Shear resistance of concrete without shear reinforcement (for more information see $V_{Rd,c}$ in design code)</i>
	double	VRds <i>Shear resistance of shear reinforcement only (for more information see $V_{Rd,c}$ in design code)</i>
	long	VErrorMessage <i>RC beam shear design message code</i>
	double	Astor) <i>calculated area of longitudinal reinforcement for torsion only</i>
		<i>RC beam design results including shear and torsion reinforcement</i>
		RRCBeamDesignBendingResult = (
	double	M <i>design moment</i>
	double	Ast <i>calculated area of tension reinforcement (required by design forces)</i>
	double	Ast_min <i>calculated minimum area of tension reinforcement (required by design code ignoring forces)</i>
	double	Asc <i>calculated area of compression reinforcement (required by design forces)</i>
	double	Asc_min <i>calculated minimum area of compression reinforcement (required by design code ignoring forces)</i>
	double	xc <i>depth of concrete section in compression</i>
	long	MErrorMessage) <i>RC beam bending design message code See here</i>
		<i>RC beam flexural design results from bending only</i>
		RRCBeamDesignDeflectionResult = (
	long	CombinationOrLoadCaseID <i>Index of load combination or load case</i>
	double	ez <i>Vertical displacements / deflection</i>
		<i>Vertical displacements / deflection with corresponding load case or combination index</i>
		RRCBeamDesignDeflectionResults = (
RRCBeamDesignDeflectionResult		Min <i>min. deflection result</i>
RRCBeamDesignDeflectionResult		Max <i>max. deflection result</i>
		<i>Deflection results</i>

Functions

long	AddLines ([in] SAFEARRAY (long)* LineIDs , [i/o] SAFEARRAY (RPartialRCBeamDesignParameters)* PartialRCBeamDesignParameters)	
	LineIDs	Line indexes for RC beam design
	PartialRCBeamDesignParameters	Partial RC beam design parameters(Cross-sections and supports for each line)
	Returns number of lines if successful, otherwise returns an error code	
	(rcbdePrestressedBeamsNotSupported , rcbdeErrorSettingLines , rcbdeInvalidArrayLength, rcbdeInvalidValue_bw, rcbdeInvalidValue_h, rcbdeInvalidValue_hf, rcbdeInvalidValue_beif, rcbdeInvalidThetaValue , errDatabaseNotReady)	
long	AddLines _vb (Visual Basic compatible function of AddLines)	
long	AddMembers ([in] SAFEARRAY (long)* MemberIDs , [i/o] SAFEARRAY (RPartialRCBeamDesignParameters)* PartialRCBeamDesignParameters)	
	MemberIDs	Member indexes for RC beam design
	PartialRCBeamDesignParameters	Partial RC beam design parameters(Cross-sections and supports for each member)
	Returns number of lines if successful, otherwise returns an error code	
	(rcbdePrestressedBeamsNotSupported , rcbdeErrorSettingMembers , rcbdeInvalidArrayLength, rcbdeInvalidValue_bw, rcbdeInvalidValue_h, rcbdeInvalidValue_hf, rcbdeInvalidValue_beif, rcbdeInvalidThetaValue , errDatabaseNotReady)	
long	AddMembers _vb (Visual Basic compatible function of AddMembers)	

```

long Calculate ([in] EResultType ResultType, [in] long LoadCaseOrCombinationOrEnvelopeUID, [in]
long LoadLevel, [in] EAnalysisType AnalysisType, [out] long SectionCount, [out]
SAFEARRAY(RRCBeamDesignResult)* Results, [out]
SAFEARRAY(RRCBeamDesignCrackResults)* CrackResults, [out]
SAFEARRAY(RRCBeamDesignReqReinfs)* RequiredReinforcements, [out] SAFEARRAY (long)
NumberOfRebars_Top, [out] SAFEARRAY (long)* NumberOfRebars_Bottom, [out] SAFEARRAY
(long)* NumberOfRebars_Top_CC, [out] SAFEARRAY (long)* NumberOfRebars_Bottom_CC, [out]
SAFEARRAY (double)* URow_Top, [out] SAFEARRAY (double)* URow_Bottom, [out] SAFEARRAY
(double)* URow_Top_CC, [out] SAFEARRAY (double)* URow_Bottom_CC, [out] SAFEARRAY
(RRCBeamDesignDeflectionResults)* DeflectionsRel,
[out] SAFEARRAY (RRCBeamDesignDeflectionResults)* DeflectionsAbs)

ResultType type of result
LoadCaseOrCombinationOrEnvelop load case ,combination or EnvelopeUID index, ignored if
eUID ResultType is rtCritical
LoadLevel load level (increment) index, if ResultType is rtLoadCase or
rtLoadCombination otherwise ignored
AnalysisType Type of Analysis
SectionCount Number of sections
Results detailed results of RC beam design including section's
position (inc. local x coordinate), array length =
SectionCount
CrackResults crack width check calculation results, array length =
SectionCount
RequiredReinforcements rebar details of required reinforcement, array length =
SectionCount
NumberOfRebars_Top number of rebars at top required by design, array length ≥
SectionCount
NumberOfRebars_Bottom number of rebars at bottom required by design, array length ≥
SectionCount
NumberOfRebars_Top_CC number of rebars at top required for crack control, array
length ≥ SectionCount
NumberOfRebars_Bottom_CC number of rebars at bottom required for crack control, array
length ≥ SectionCount
URow_Top row distances from top edge required by design, array length ≥
SectionCount
URow_Bottom row distances from bottom edge required by design, array
length ≥ SectionCount
URow_Top_CC row distances from top edge required for crack control, array
length ≥ SectionCount
URow_Bottom_CC row distances from bottom edge required for crack control,
array length ≥ SectionCount
DeflectionsRel Min. and max. relative deflections, array length ≥
SectionCount
URow_Bottom_CC Min. and max. absolute deflections, array length ≥
SectionCount

```

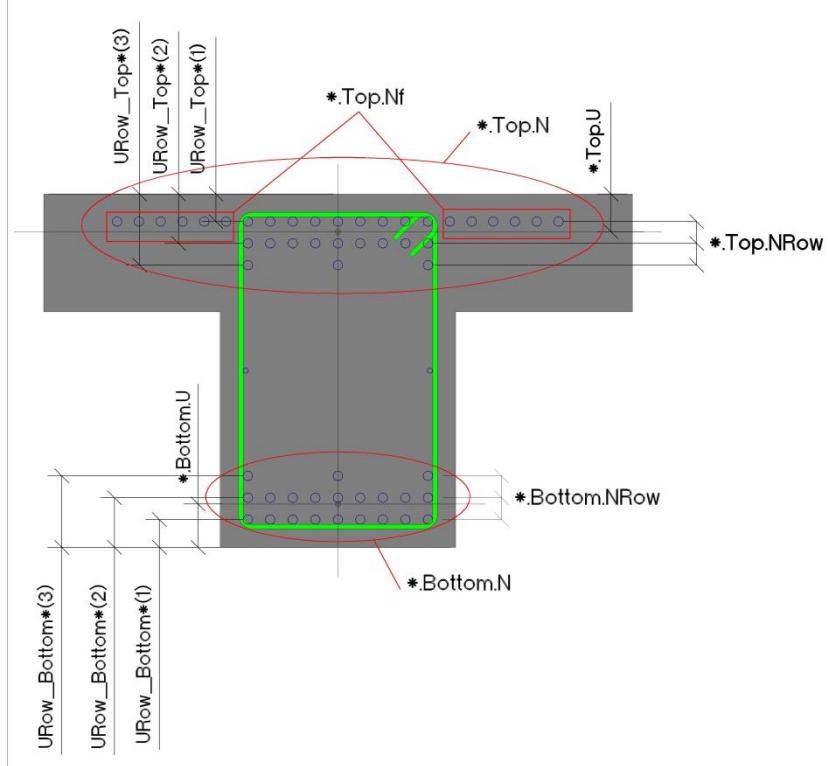
Returns number of results if successful, otherwise returns an error code ([errDatabaseNotReady](#) ,
[rcbdeInvalidCombinationOfLoadCaseAndLoadLevel](#),[rcbdeInvalidCombinationOfLoadCombinationAnd](#)
[LoadLevel](#), [rcbdeInvalidLoadCaseId](#), [rcbdeInvalidLoadCombinationId](#), [rcbdeInvalidAnalysisType](#)).

NOTE: prior to calling this function (Calculate), of these functions must be called in this order:

1. **AddLines** (for lines) or **AddMembers** (for members)

Example of output arrays:

1st section with results: RequiredReinforcements(1)



RequiredReinforcements(1).Top. Nf = 12 -> 12 bars in top flange

RequiredReinforcements(1).Top. N = 33 -> Total 33 bars at top($2 \times 6 + 9 + 9 + 3 = 33$ bars)

RequiredReinforcements(1).Top. Nrow = 3 -> 3 rows of bars at top

NumberOfRebars_Top(1) = 21 -> 21 bars in 1st row

NumberOfRebars_Top(2) = 9 -> 9 bars in 2nd row

NumberOfRebars_Top(3) = 3 -> 3 bars in 3rd row

URow_Top(1) = 0,031 -> 1st row is 0,031m from top edge

URow_Top(2) = 0,068 -> 2nd row is 0,068 m from top edge

URow_Top(3) = 0,105 -> 3rd row is 0,105 m from top edge

... similarly with other arrays: NumberOfRebars_*, URow_*

next section with results:

RequiredReinforcements(2).Top. Nf = 4 -> 4 bars in top flange

RequiredReinforcements(2).Top. N = 9 -> Total 9 bars ($5+4 = 9$ bars)

RequiredReinforcements(2).Top. Nrow = 2 -> Bars arranged in 2 rows

NumberOfRebars_Top(4) = 5 -> 5 bars in 1st row

NumberOfRebars_Top(5) = 4 -> 4 bars in 2nd row

URow_Top(4) = 0,031 -> 1st row is 0,031 m from top edge

URow_Top(5) = 0,068 -> 2nd row is 0,068 m from top edge

... similarly with other arrays: NumberOfRebars_*, URow_*

next section with results:

RequiredReinforcements(3).Top. Nf = 0 -> 0 bars in top flange

RequiredReinforcements(3).Top. N = 3 -> Total 3 bars at top

RequiredReinforcements(3).Top. Nrow = 1 -> 1 row of bars at top

NumberOfRebars_Top(6) = 5 -> 5 bars in one row

URow_Top(6) = 0,031 -> row is 0,031 m from top edge

long	Clear
	<i>Clears all results and settings from the interface.</i>
	<i>Returns 1 if successful, otherwise returns an error code (errDatabaseNotReady)</i>
long	GetDesignParameters ([i/o] RRCBeamDesignParameters RCBeamDesignParameters, [out] SAFEARRAY(byte) DesignCodeParameters)
	RCBeamDesignParameters <i>RC beam design parameters</i>
	DesignCodeParameters <i>a pointer to the RC beam design parameters record in SAFEARRAY format, record type depending on used design code. Typecast to a corresponding record (see below) to the design code, see also How to read load data (GetLoad).</i>
	<i>Returns 1 if successful, otherwise returns an error code (errDatabaseNotReady)</i>
	<i>Record type depends on current NationalDesignCode</i>
	RRCBeamDesignParameters_DIN :
	<ul style="list-style-type: none">• <i>ndcGerman_DIN1045_1</i>
	RRCBeamDesignParameters_EC :
	<ul style="list-style-type: none">• <i>ndcEuroCode, ndcEuroCode_GER, ndcEuroCode_Austrian, ndcEuroCode_UK, ndcEuroCode_NL, ndcEuroCode_FIN, ndcEuroCode_HU, ndcEuroCode_CZ, ndcEuroCode_B, ndcEuroCode_PL, ndcEuroCode_DK, ndcEuroCode_S</i>
	RRCBeamDesignParameters_EC_RO :
	<ul style="list-style-type: none">• <i>ndcEuroCode_RO</i>
	RRCBeamDesignParameters_ITA :
	<ul style="list-style-type: none">• <i>ndcItalian</i>
	RRCBeamDesignParameters_MSZ :
	<ul style="list-style-type: none">• <i>ndcHungarian_MSZ</i>
	RRCBeamDesignParameters_SIA :
	<ul style="list-style-type: none">• <i>ndcSwiss_SIA26x</i>
	RRCBeamDesignParameters_STAS :
	<ul style="list-style-type: none">• <i>ndcRomanian_STAS</i>
long	GetLines ([out] SAFEARRAY(long) Linelds)
	Linelds <i>array of line indexes for RC beam design</i>
	<i>Returns number of lines.</i>
long	SetDesignParameters ([i/o] RRCBeamDesignParameters RCBeamDesignParameters, [in] SAFEARRAY(byte) DesignCodeParameters)
	RCBeamDesignParameters <i>RC beam design parameters</i>
	DesignCodeParameters <i>pointer to the reinforcement parameters record in SAFEARRAY format, record type depending on used design code. Typecast a corresponding record to a safearray (see GetDesignParameters), see also How to modify load data (SetLoad)</i>
	<i>Call AddMembers or Addlines function before calling this function. Returns 1 if successful, otherwise returns an error code (errDatabaseNotReady, rcbdeInvalidMaterialId, rcbdeInvalidStirrupMaterial, rcbdeInvalidStirrupDiameter, rcbdeInvalidStirrupLegs, rcbdeInvalidBottomPos, rcbdeInvalidTopPos, rcbdeInvalidThetaValue)</i>
long	SetDesignParameters_vb ([i/o] RRCBeamDesignParameters RCBeamDesignParameters, [i/o] SAFEARRAY(byte) DesignCodeParameters)
	<i>Visual Basic compatible function of SetDesignParameters</i>

Properties

[ERCBeamDesignPlane](#) **BeamDesignPlane** • Get or set design plane of the RC beam

IAxisVMRCColumnChecking

Interface used for reading RC column design results.

If property returning this interface is null (nil) then the extension module RC2 is not available.

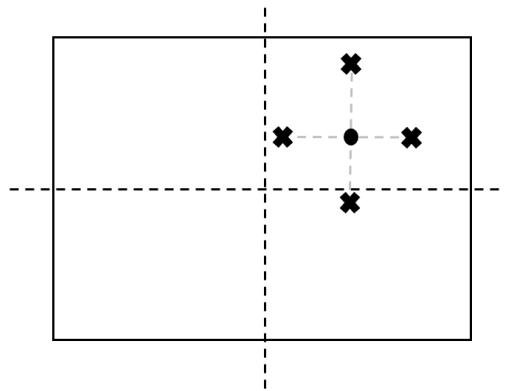
Error codes

enum	ERCColumnCheckingError = {	
	rcccelnvalidLineId = -100001	<i>Line index not valid</i>
	rcccelnvalidMemberId = -100002	<i>Member indexes not valid</i>
	rcccelnvalidLoadCaseId = -100003	<i>Load case index invalid</i>
	rcccelnvalidLoadCombinationId = -100004	<i>Load combination index invalid</i>
	rcccelnvalidCombinationOfLoadCaseAndLoadLevel = -100005	<i>Invalid combination of load case and load level</i>
	rcccelnvalidCombinationOfLoadCombinationAndLoadLevel = -100006	<i>Invalid combination of load combination and load level</i>
	rccceColumnReinforcementParametersNotSet = -100007	<i>Column reinforcement parameters not set</i>
	rccceCapacityCurveCannotBeGenerated = -100008	<i>Capacity curve can not be generated</i>
	rcccelnvalidRebarSteelGradeId = -100009	<i>Rebar steel grade is invalid</i>
	rcccelnvalidConcreteMaterialId = -100010	<i>Concrete material index is invalid</i>
	rcccelnvalidColumnRebarsId = -100011	<i>Column rebar index is invalid</i>
	rccceCapacityCurveNotYetGenerated = -100012	<i>Capacity curve has not yet been generated</i>
	rccceDifferentColumnReinforcementParameters = -100013	<i>Column reinforcement parameters vary</i>
	rcccelnvalidArrayLength = -100014	<i>Invalid length of array</i>
	rcccelnvalidAnalysisType = -100015	<i>Invalid type of analysis</i>
	rccceExcentricity = -100016	<i>Invalid eccentricity</i>
	rccceColumnReinforcementNoForces = -100017	<i>Forces for column checking are not available</i>
	rcccelnvalidEnvelopeID = -100018	<i>Invalid EnvelopeID</i>
	rcccelnvalidCheckingParameters = -100019	<i>at least one of the checking parameters is invalid</i>
	rccceShrinkageEpsMustBePositive = -100020	<i>shrinkage strain must be positive</i>
	rccceVTCcheckIsNotSupported = -100021	<i>shear/torsion check is not supported</i>
	rccceStirrupParametersAreInvalid = -100022	<i>stirrup parameters are invalid</i>
	rccceShearCrackAngleIsInvalid = -100023	<i>defined shear crack angle is too low/high</i>
	rccceVTCapacityDesignIsNotSupported = -100024	<i>capacity design is not supported</i>
	rcccelnvalidSteelMaterialId = -100025 }	<i>steel grade is invalid</i>

Records / structures

	RMyMz = (
double	My	<i>My bending moment about local y axis [kNm]</i>
double	Mz)	<i>Mz bending moment about local z axis [kNm]</i>
	<i>Bending moments in the RC column</i>	
	RMyMzFi = (
double	My	<i>My bending moment about local y axis [kNm]</i>
double	Mz	<i>Mz bending moment about local z axis [kNm]</i>
double	fi)	<i>rotation angle of neutral axis [rad]</i>
	<i>Bending moments with fi in the RC column</i>	
	RNMInteractionDiagMin	
	Max = (
double	NMin	<i>min. axial force [kN]</i>
double	NMax	<i>max. axial force [kN]</i>
double	MyMin	<i>min. My bending moment about local y axis [kNm]</i>
double	MyMax	<i>max. My bending moment about local y axis [kNm]</i>
double	MzMin	<i>min. Mz bending moment about local z axis [kNm]</i>
double	MzMax	<i>max. Mz bending moment about local z axis [kNm]</i>
	<i>Min. and max. forces in the RC column</i>	

	RColumnCheckResult = (
double	xRelPos	relative position of the cross section
<u>EBoolean</u>	Passed	Forces are within N-M interaction diagram (see RColumnForces)
BSTR	Combination	name of load case of load combination
double	Eff_Const_N	Column's efficiency ($N = \text{const.}$)
double	Eff_Const_e	Column's efficiency ($M/N = e = \text{const.}$)
double	My_c	My without considering second order eccentricity
double	My_e2y_P	My considering $+e2y$
double	My_e2y_N	My considering $-e2y$
double	My_e2z_P	My considering $+e2z$
double	My_e2z_N	My considering $-e2z$
double	Mz_c	Mz without considering second order eccentricity
double	Mz_e2y_P	Mz considering $+e2y$
double	Mz_e2y_N	Mz considering $-e2y$
double	Mz_e2z_P	Mz considering $+e2z$
double	Mz_e2z_N	Mz considering $-e2z$



RC column check at a specific cross section

	RColumnVTCheckResult = (
double	xRelPos	relative position of the cross section
<u>EBoolean</u>	Passed	the column is adequate against shear and torsion effects at $xRelPos$
double	VyR	shear resistance in local y direction without considering torsion
double	VzR	shear resistance in local z direction without considering torsion
double	kT	shear resistance reduction factor for stirrups due to torsion
double	Ast	additional longitudinal reinforcement required due to torsion
double	Eff_Vy	shear utilization in local y direction without considering torsion
double	Eff_Vz	shear utilization in local z direction without considering torsion
double	Eff_Vy_Vz	overall shear utilization without torsion
double	Eff_Vy_Vz_Tx	overall utilization considering shear and torsion interaction
double	Eff_Vy_Vz_Tx_max	utilization of the concrete compression strut
)	

	RColumnForces = (
double	Nx	axial force [kN]
double	My	My bending moment about local y axis [kNm]
double	Mz	Mz bending moment about local z axis [kNm]
double	Vy	shear force in local y axis [kN]
double	Vz	shear force in local z axis [kN]
double	Tx	torsional moment [kNm]

Main forces in the RC column at at a specific cross section

Functions

long	CheckByParameters ([i/o] RCColumnReinforcementParameters Parameters , [in] SAFEARRAY(double), xPos , [in] SAFEARRAY(RCColumnForces) ColumnForces , [out] SAFEARRAY(RCColumnCheckResult) ColumnCheckResults)
	Parameters <i>RC column reinforcement parameters</i> xPos <i>array of positions along the column where check is performed</i> ColumnForces <i>internal forces on the RC column related to xPos array</i> ColumnCheckResults <i>check forces on RC column on top, bottom and centre</i>
	<i>If successful returns number of RC column check results, otherwise an error code (errDatabaseNotReady, rccceExcentricity, errCOMServerInternalError, rccceColumnReinforcementParametersNotSet, rccceCapacityCurveCannotBeGenerated, rccceInvalidRebarSteelGradeId, rccceInvalidConcreteMaterialId, rccceInvalidColumnRebarsId, rccceInvalidArrayLength).</i>
long	CheckByParameters_vb (Visual Basic compatible function of CheckByParameters)
long	CheckMembers ([in] SAFEARRAY(long) MemberIds , [in] EResultType ResultType , [in] long LoadCaseOrCombinationOrEnvelopeUID , [in] long LoadLevel , [in] EAnalysisType AnalysisType , [in] ELongBoolean Creep , [out] SAFEARRAY(RCColumnCheckResult) ColumnCheckResults , [out] SAFEARRAY(BSTR) Combinations)
	MemberIds <i>index of the members</i> ResultType <i>Type of result</i> LoadCaseOrCombinationOrEnvelopeUID <i>load case, load combination or unique envelope index</i> LoadLevel <i>load level (increment) index</i> AnalysisType <i>Type of Analysis</i> Creep <i>Nonlinear analysis can be performed with and without considering creep of RC elements. With this parameter on can select whether the column check is performed considering nonlinear analysis results with or without creep. More here...</i> ColumnCheckResults <i>check forces on RC column</i> <i>Note: xRelPos returns always with 0</i> Combinations <i>list of strings in parallel to ColumnCheckResults values. The textual representation of the source load case / load combination / critical combination of the corresponding ColumnCheckResults record</i>
	<i>If successful returns number of RC column check results, otherwise an error code (errDatabaseNotReady, rccceInvalidCombinationOfLoadCaseAndLoadLevel, rccceInvalidCombinationOfLoadCombinationAndLoadLevel, rccceInvalidLoadCaseId, rccceInvalidLoadCombinationId, rccceDifferentColumnReinforcementParameters, rccceColumnReinforcementParametersNotSet, errCOMServerInternalError, rccceCapacityCurveCannotBeGenerated, rccceColumnReinforcementParametersNotSet, rccceInvalidArrayLength, rccceInvalidAnalysisType).</i>
long	CheckMembers_vb (Visual Basic compatible function of CheckMembers)
long	Clear <i>Clears all results and settings from the interface.</i> <i>Returns 1 if successful, otherwise returns an error code (errDatabaseNotReady)</i>

long	CheckVTByParameters ([i/o] RColumnReinforcementParameters Parameters , [in] SAFEARRAY(double), xPos , [in] SAFEARRAY(RColumnForces) ColumnForces , [out] SAFEARRAY(RColumnVTChekResult) ColumnVTChekResults)
	<p style="margin-left: 20px;">Parameters <i>RC column reinforcement parameters</i></p> <p style="margin-left: 20px;">xPos <i>array of positions along the column where check is performed</i></p> <p style="margin-left: 20px;">ColumnForces <i>internal forces on the RC column related to xPos array</i></p> <p style="margin-left: 20px;">ColumnVTChekResults <i>results of shear and torsion checks</i></p>
	<p>If successful returns number of RC column shear/torsion check results, otherwise an error code (errDatabaseNotReady, rccceExcentricity, errCOMServerInternalError, rccceColumnReinforcementParametersNotSet, rccceCapacityCurveCannotBeGenerated, rccceInvalidRebarSteelGradeId, rccceInvalidConcreteMaterialId, rccceInvalidColumnRebarsId, rccceInvalidArrayLength, rccceVTChekIsNotSupported, rccceStirrupParametersAreInvalid, rccceShearCrackAngleIsInvalid).</p>
long	CheckVTByParameters_vb (Visual Basic compatible function of CheckVTByParameters)
long	CheckVTByParameters_EQCapacityDesign ([i/o] RColumnReinforcementParameters Parameters , [in] SAFEARRAY(double), xPos , [in] SAFEARRAY(RColumnForces) ColumnForces , [in] SAFEARRAY(RColumnForces) EQColumnForces , [out] SAFEARRAY(RColumnVTChekResult) ColumnVTChekResults)
	<p style="margin-left: 20px;">Parameters <i>RC column reinforcement parameters</i></p> <p style="margin-left: 20px;">xPos <i>array of positions along the column where check is performed</i></p> <p style="margin-left: 20px;">ColumnForces <i>internal forces on the RC column related to xPos array</i></p> <p style="margin-left: 20px;">EQColumnForces <i>internal forces from seismic effect only on the RC column related to xPos array</i></p> <p style="margin-left: 20px;">ColumnVTChekResults <i>results of shear and torsion checks</i></p>
	<p>If successful returns number of RC column shear/torsion check results, otherwise an error code (errDatabaseNotReady, rccceExcentricity, errCOMServerInternalError, rccceColumnReinforcementParametersNotSet, rccceCapacityCurveCannotBeGenerated, rccceInvalidRebarSteelGradeId, rccceInvalidConcreteMaterialId, rccceInvalidColumnRebarsId, rccceInvalidArrayLength, rccceVTChekIsNotSupported, rccceStirrupParametersAreInvalid, rccceShearCrackAngleIsInvalid, rccceVTCapacityDesignIsNotSupported).</p>
long	CheckVTByParameters_EQCapacityDesign_vb (Visual Basic compatible function of CheckVTByParameters_EQCapacityDesign)

long **CheckVTMembers** ([in] SAFEARRAY(long) **MemberIds**, [in] [EResultType](#) **ResultType**,
 [in] long **LoadCaseOrCombinationOrEnvelopeUID**, [in] long **LoadLevel**,
 [in] [EAnalysisType](#) **AnalysisType**, [in] [EBoolean](#) **Creep**,
 [out] SAFEARRAY([RColumnVTChekResult](#)) **ColumnVTChekResults**)

MemberIds	<i>index of the members</i>
ResultType	<i>Type of result</i>
LoadCaseOrCombinationOrEnvelopeUID	<i>load case, load combination or unique envelope index</i>
LoadLevel	<i>load level (increment) index</i>
AnalysisType	<i>Type of Analysis</i>
Creep	<i>Nonlinear analysis can be performed with and without considering creep of RC elements. With this parameter on can select whether the column check is performed considering nonlinear analysis results with or without creep. More here...</i>
ColumnForces	<i>internal forces on the RC column</i>
ColumnVTChekResults	<i>results of shear and torsion checks</i> <i>Note: xRelPos returns always with 0</i>

If successful returns number of RC column shear/torsion check results, otherwise an error code ([errDatabaseNotReady](#), [rccelInvalidCombinationOfLoadCaseAndLoadLevel](#), [rccelInvalidCombinationOfLoadCombinationAndLoadLevel](#), [rccelInvalidLoadCaseId](#), [rccelInvalidLoadCombinationId](#), [rccedifferentColumnReinforcementParameters](#), [rccolumnReinforcementParametersNotSet](#), [errCOMServerInternalError](#), [rccCapacityCurveCannotBeGenerated](#), [rccColumnReinforcementParametersNotSet](#), [rccelInvalidArrayLength](#), [rccelInvalidAnalysisType](#), [rccvTChekIsNotSupported](#), [rccStirrupParametersAreInvalid](#), [rccShearCrackAngleIsInvalid](#)).

long **CheckVTMembers_vb** (Visual Basic compatible function of **CheckVTMembers**)

long	GenerateNMInteractionDiagByLine ([in] long LinId , [out] SAFEARRAY(double) N , [out] SAFEARRAY(long) ItemCounts , [out] SAFEARRAY(RMyMzFi) * MyMzFi)
	LinId <i>index of the line</i> N <i>array with axial forces</i> ItemCounts <i>array with number of moment combinations</i> MyMzFi <i>array with moment combinations</i>
	<i>Returns number of generated N-M interaction diagrams if successful, otherwise returns an error code (errDatabaseNotReady , rccceCapacityCurveCannotBeGenerated, rccceColumnReinforcementParametersNotSet, rccceInvalidLinId).</i>
long	GenerateNMInteractionDiagByMember ([in] long MemberId , [out] SAFEARRAY(double) N , [out] SAFEARRAY(long) ItemCounts , [out] SAFEARRAY(RMyMzFi) * MyMzFi)
	MemberId <i>index of the member</i> N <i>array with axial forces</i> ItemCounts <i>array with number of MY,Mz combinations</i> MyMzFi <i>array with moment combinations</i>
	<i>Returns number of generated N-M interaction diagrams if successful, otherwise returns an error code (errDatabaseNotReady , rccceCapacityCurveCannotBeGenerated, rccceColumnReinforcementParametersNotSet, rccceInvalidMemberId, errInternalException).</i>
long	GenerateNMInteractionDiagByParameters ([i/o] RColumnReinforcementParameters Parameters , [out] SAFEARRAY(double) N , [out] SAFEARRAY(long) ItemCounts , [out] SAFEARRAY(RMyMzFi) * MyMzFi)
	Parameters <i>RC column reinforcement parameters</i> N <i>array with axial forces</i> ItemCounts <i>array with number of moment combinations</i> MyMzFi <i>array with moment combinations</i>
	<i>Returns number of moment combinations if successful, otherwise returns an error code (errDatabaseNotReady , rccceCapacityCurveCannotBeGenerated, rccceInvalidRebarSteelGradeId, rccceInvalidConcreteMaterialId, rccceInvalidColumnRebarsId).</i>
long	GetMyMzPolyByNx ([in] double Nx , [out] SAFEARRAY(RMyMz) Poly)
	Nx <i>axial force</i> Poly <i>polygon with moment combination My - Mz</i>
	<i>Returns number of moment combinations if successful, otherwise returns an error code (rccceCapacityCurveNotYetGenerated).</i>
long	GetNMInteractionDiagMinMax ([i/o] RNMInteractionDiagMinMax NMInteractionDiagMinMax)
	NMInteractionDiagMinMax <i>polygon with moment combination My - Mz</i> <i>Returns 1 if successful, otherwise returns an error code (rccceCapacityCurveNotYetGenerated).</i>

IAxisVMReferences

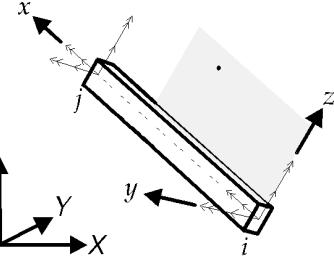
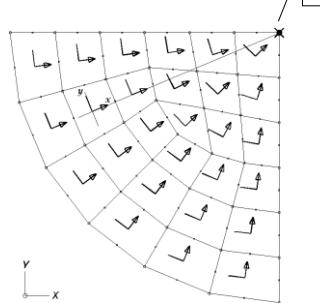
References in the model

Enumerated types

enum **EReferenceType** = {

rtPoint = 0x01, *reference point*

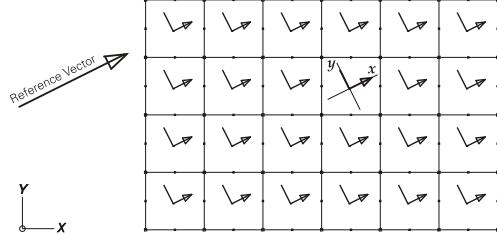
Reference point



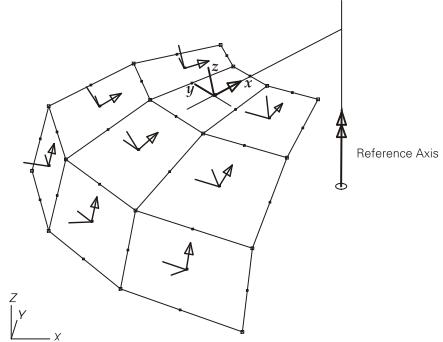
Reference point
(local z axis of
beam inplane
where ref. point
is)

rtVector =
0x02,
reference vector

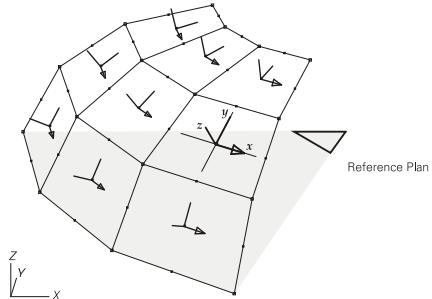
reference vector



rtAxis = 0x03,
reference axis



rtPlane = 0x04,
reference plane



rtBeta = 0x5,

reference angle

(If the element is parallel with the global Z direction, the angle is relative to the global X axis. In any other case the angle is relative to the global Z axis)

rtNone = 0x6 }

none of the above

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** ([*i/o*] **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 **GetItem** ([*in*] long Index, [*i/o*] **RReference** Value)

Index the reference to get

Get a reference by index. If successful, returns Index otherwise returns an error code ([errDatabaseNotReady](#), [errNotFound](#)).

long	GetSelectedItemIds ([out] SAFEARRAY (long) * ItemIds)
	ItemIds list of selected references If successful, returns the number of selected elements otherwise returns an error code (errDatabaseNotReady).
long	IndexOf ([i/o] 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] EBoolean 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). <u>NOTE:</u> Call Refresh function afterwards if not called between functions BeginUpdate and EndUpdate
long	SetItem ([in] long Index , [i/o] RReference Value)
	Index the reference to get Set a reference with a given index. If successful, returns Index otherwise returns an error code (errDatabaseNotReady , errNotFound).

Properties

long	Count Get number of references in the model
EBoolean	Selected [long Index] • Get or set the selection status of a reference <u>NOTE:</u> Call Refresh function afterwards if not called between functions BeginUpdate and EndUpdate
long	SelCount Get number of selected references in the model

IAxisVMReports

Reports in the model

Functions

long **AddDrawingFromLibrary** ([in] long **ReportIndex**, [in] long **DrawingIndex**)

ReportIndex index of the report

DrawingIndex index of the drawing, $0 < \text{DrawingIndex} \leq \text{IAxisVMDrawingsLibrary.Count}$

Adds drawing from library to the end of report. If successful, returns the report index otherwise an error code (see [EGeneralErrors](#)).

long **AddImageFromFile** ([in] long **ReportIndex**, [in] BSTR **FileName**, [in] BSTR **Caption**,
[in] BSTR **OptionsJSON**)

ReportIndex index of the report

FileName name of the image file

Caption caption of the image

OptionsJSON optional parameters in JSON string , can be empty

Handled fields in JSON string:

Boolean **Fit in page**:

If true, image is scaled to fit on one page of the report, default: false

Ignored during RTF file export

Boolean **Rotate**:

If true, image is rotated page of the report, default: false

Example: {"Fit in page":true,"Rotate":false}

Adds image file to the end of the report. Image file must be in subfolder **Images_*** where * is the AxisVM model's filename without extension (.axs). This folder must be in same folder as the loaded AxisVM model file (file extension: .axs). Supported extensions are: BMP, JPG, WMF, EMF. If successful, returns the report index otherwise an error code (see [EGeneralErrors](#)).

long **AddRootFolder** ([in] long **ReportIndex**, [in] BSTR **Name**)

ReportIndex index of the report

Name name of the folder

Adds a root folder to the report. If successful, returns the count of elements in the report, otherwise an error code (see [EGeneralErrors](#)).

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

Index index of the report

Deletes the report with index. $1 \leq \text{Index} \leq \text{Count}$.

If successful, returns Index otherwise returns an error code (see [EGeneralErrors](#)).

long **DeleteImage** ([in] long **ReportIndex**, [in] long **ImageIndex**)

ReportIndex index of the report

ImageIndex index of the image

Deletes an image from the report with index. $1 \leq \text{Index} \leq \text{ImageCount}$.

If successful, returns ImageIndex otherwise returns an error code (see [EGeneralErrors](#)).

long **GenerateFromTemplateFile** ([in] long **Index**, [in] BSTR **FileName**)

Index index of the report

FileName name of the template file, .rep extension

Regenerates report with index based on template file. If successful, returns the report index otherwise an error code (see [EGeneralErrors](#)).

long	ImagesInFolder ([in] long ReportIndex , [in] BSTR FolderPath , [out] SAFEARRAY(long)* ImageIds)
	ReportIndex <i>index of the report</i>
	FolderPath <i>folder's path and name required format '/(Folder1)/(Folder2)/etc...' If FolderPath is '/' then images will be found that are stored in the report's main directory not in subdirectories.</i>
	ImageIds <i>array of image indices</i>
	<i>Returns with image indices situated in a given folder of a report with ReportIndex. 1 ≤ Index ≤ Count. If successful, returns the number of images otherwise returns an error code (see EGeneralErrors).</i>
long	IndexOf ([in] BSTR Name)
	Name <i>name of the report</i>
	<i>get index of the report by name. If successful, returns Index otherwise returns an error code (see EGeneralErrors).</i>
long	NewFromTemplateFile ([in] BSTR Name , [in] BSTR FileName)
	Name <i>name of the report</i>
	FileName <i>name of the template file, .rep extension</i>
	<i>Creates new report based on template file. If successful, returns the report index otherwise an error code (see EGeneralErrors).</i>

Properties

long	Count <i>Get number of reports in the model</i>
long	ImageCount [long Index]
	Index - report index
	<i>Get number of images in the report</i>
BSTR	ImagePath [long ReportIndex , long ImageIndex]
	ReportIndex - report index
	ImageIndex - image index
	<i>Get the path with caption of an image in the report</i>
BSTR	ImageCaption [long ReportIndex , long ImageIndex]
	ReportIndex - report index
	ImageIndex - image index
	<i>Get the caption of an image in the report</i>
BSTR	Name [long Index] • <i>Get or set the name of the report</i>
BSTR	TemplateDescription [long Index] • <i>Get or set the description of the template used with the report</i>

IAxisVMResults

Results of different analysis types are stored in *result blocks*. In linear analysis one result block contains all results of a load case or combination, in other analysis types multiple result blocks can be associated to a load case or combination. These result blocks store results for intermediate load steps or different mode shapes.

All results from AxisVM, results of the section can be accessed through [IAxisVMSections](#) interface.

The following diagram shows an example for a model where 5 load cases are defined (LC1, LC2, LC3, LC4, LC5). For some load cases a nonlinear, a vibration or a buckling analysis is also completed. Each table cell represents a result block. Result cases are abbreviated as [RCx].

<i>linear analysis</i>	<i>nonlinear analysis</i>	<i>vibration analysis</i>	<i>buckling analysis</i>	<i>dynamic analysis</i>
[RC1] LC1	[RC1] LC3 Level 1	[RC1] LC2 Shape 1	[RC1] LC3 Shape 1	[RC1] LC2 Time 1
[RC2] LC2	LC3 Level 2	LC2 Shape 2	LC3 Shape 2	LC2 Time 2
[RC3] LC3	LC3 Level 3	LC2 Shape 3	LC3 Shape 3	LC2 Time 3
[RC4] LC4	LC3 Level 4	LC2 Shape 4	LC3 Shape 4	LC2 Time 4
[RC5] LC5	LC3 Level 5	LC2 Shape 5	LC3 Shape 5	LC2 Time 5
	[RC2] LC5 Level 1	LC2 Shape 6		LC2 Time 6
	LC5 Level 2	LC2 Shape 7		LC2 Time 7
	LC5 Level 3	LC2 Shape 8		LC2 Time 8
	LC5 Level 4	LC2 Shape 9		LC2 Time 9
	LC5 Level 5	LC2 Shape10		LC2 Time 10
	LC5 Level 6	LC2 Shape11		
		[RC2] LC1 Shape 1		
		LC1 Shape 2		
		LC1 Shape 3		

IAxisVMResults properties will have the following values:

```

ResultCaseCount[atLinearStatic] = 5
ResultCaseCount[atNonlinearStatic] = 2
ResultCaseCount[atLinearVibration] = 2
ResultCaseCount[atBuckling] = 1
ResultCaseCount[atDynamic] = 1

LoadCasId[atNonlinearStatic, 1] = 3
LoadCasId[atNonlinearStatic, 2] = 5
LoadCasId[atVibration, 1] = 2
LoadCasId[atVibration, 2] = 1
LoadCasId[atBuckling, 1] = 3
LoadCasId[atDynamic, 1] = 2

LoadLevelCount[atNonlinearStatic,1] = 5
LoadLevelCount[atNonlinearStatic,2] = 6
LoadLevelCount[atLinearVibration,1] = 11
LoadLevelCount[atLinearVibration,2] = 3
ModeShapeCount[atBuckling, 1] = 5
TimeStepCount[atDynamic, 1] = 10

```

```

ResultCaseOfLoadCase[atNoninearStatic, 3] = 1
ResultCaseOfLoadCase[atNoninearStatic, 5] = 2
ResultCaseOfLoadCase[atVibration, 1] = 2
ResultCaseOfLoadCase[atVibration, 2] = 1
ResultCaseOfLoadCase[atBuckling, 1] = 3
ResultCaseOfLoadCase[atDynamic, 2] = 1

```

If the model database is not available while calling, functions or reading properties an [errDatabaseNotReady](#) error code is returned.

Displacement results can be read through a **Displacements** interface, internal forces can be read through a **Forces** interface, stresses can be read through a **Stresses** interface. For more info see [Overview of AxisVM result objects ...](#)

Error codes

enum EResultsError = {	
reInvalidAnalysisType = -100001	<i>invalid analysis type</i>
reResultCaseIndexOutOfBounds = -100002	<i>result block index is out of bounds</i>
reResultCasesNotLoadCase = -100003	<i>result block belongs to a load combination</i>
reResultCasesNotLoadCombination = -100004	<i>result block belongs to a load case</i>
reLoadCasesOutOfBounds = -100005	<i>load case index is out of bounds</i>
reLoadCombinationsOutOfBounds = -100006	<i>load combination index is out of bounds</i>
reFrequencyIndexOutOfBounds = -100007	<i>frequency index is out of bounds</i>
reModeShapeIndexOutOfBounds = -100008	<i>mode shape index is out of bounds</i>
reLoadLevelIndexOutOfBounds = -100009	<i>load level index is out of bounds</i>
reInvalidArrayLength = -100010	<i>length of the array is not the expected value</i>
reInvalidLoadCaseType = -100011	<i>invalid load case type</i>
reInvalidNationalDesignCode = -100012	<i>Invalid national design code</i>
reInvalidResponseSpectraParam = -100013	<i>Invalid response spectrum parameters</i>
reITAReductionCriterionNotSatisfied = -100014	<i>ITA reduction criterion not satisfied</i>
reSteelDesignResultsDisabled = -100015	<i>extension module SD1 is not available</i>
reCalculatedReinforcementDisabled = -100016	<i>extension module RC1 is not available</i>
reReinforcementCheckDisabled = -100017	<i>When used design code doesn't support reinforcement checks or extension module RC1 is not available</i>
reMissingAnalysisResults = -100018	<i>Results are not available run the analysis first</i>
reRC3moduleNotAvailable = -100019	<i>extension module RC3 is not available</i>
reRC1moduleNotAvailable = -100020	<i>extension module RC1 is not available</i>
reTD1moduleNotAvailable = -100021	<i>extension module TD1 is not available</i>
reDYNmoduleNotAvailable = -100022	<i>extension module DYN is not available</i>
reSE2moduleNotAvailable = -100023	<i>extension module SE2 is not available</i>
reNLpackageNotAvailable = -100024	<i>extension package NL (non-linear) is not available</i>
rePushoverSpectrumIsNotValid = -100025	<i>pushover spectrum is invalid or not available</i>
rePushoverSpectrumIsNotParametric = -100026	<i>only parametric spectrum is supported for pushover</i>
}	

Enumerated types

enum EAnalysisType = {	
atLinearStatic = 0x00	<i>linear analysis</i>
atNonLinearStatic = 0x01	<i>nonlinear analysis</i>
atLinearVibration = 0x02	<i>linear vibration analysis</i>
atNonLinearVibration = 0x03	<i>nonlinear vibration analysis</i>
atBuckling = 0x04	<i>buckling analysis</i>
atDynamic = 0x05 }	<i>time history analysis</i>
	<i>Type of the analysis</i>
enum EMinMaxType = {	
mtMin = 0x01,	<i>minimum values</i>
mtMax = 0x02,	<i>maximum values</i>
mtMinMax = 0x03 }	<i>used only in display parameters (IAxisVMWindows and IAxisVMWindow interfaces)</i>
	<i>Identifies minimum or maximum values for envelope and critical results when reading results.</i>
	<i>Alternatively to set display settings in the window(s).</i>
enum ESurfaceVertexType = {	
svtContourPoint = 0x00,	<i>surface polygon vertex (node)</i>
svtContourLineMidPoint = 0x01,	<i>edge midpoint</i>
svtCenterPoint = 0x02 }	<i>surface center point</i>
	<i>Identifies a surface result location</i>

enum	EXYchartFillType = { xycftNone = 0x0, xycftSolid = 0x1, xycftGradient = 0x2 }	<i>dataset line only</i> <i>dataset line with fill</i> <i>dataset line with fill colour changing with gradient</i>
<i>Used for showing the dataset in chart</i>		
enum	EXYchartLabelingStyle = { xyclsArrange = 0x0, xyclsOverlapFilter = 0x1 }	<i>labels are rearranged in order show most of them</i> <i>labels are filtered in order to prioritize important labels</i>
<i>Used when showing labels in chart</i>		

Records / structures

	RFrequencyParameters = (
double	Frequency	<i>frequency</i>
double	EigenValConv	<i>eigenvalue error</i>
double	EigenVecConv	<i>eigenvector error</i>
double	EigenValConvLimit	<i>convergence tolerance for the eigenvalue</i>
double	EigenVecConvLimit	<i>convergence tolerance for the eigenvector</i>
long	Iteration	<i>number of actual iterations completed</i>
)		
	RNcrParameters = (
double	Ncr	<i>critical load parameter</i>
double	EigenValConv	<i>eigenvalue error</i>
double	EigenVecConv	<i>eigenvector error</i>
double	EigenValConvLimit	<i>convergence tolerance for the eigenvalue</i>
double	EigenVecConvLimit	<i>convergence tolerance for the eigenvector</i>
long	Iteration	<i>number of actual iterations completed</i>
)		
	RNonLinearAnalysisResultInfo = (
long	Increments	<i>number of increments (load levels)</i>
long	Iterations	<i>number of actual iterations completed</i>
double	Lambda	<i>load factor of a load level ($0 < \text{Lambda} \leq 1$)</i>
double	ErrorP	<i>relative error of the force convergence</i>
double	ErrorU	<i>relative error of the displacement convergence</i>
double	ErrorE	<i>relative error of the work convergence</i>
double	ErrorEq	<i>solution error</i>
double	ControlVal	<i>control parameter value</i>
ELongBoolean	Convergence	<i>if True, the iteration has converged</i>
)		
	RResultBlockInfo = (
long	ResultCase	<i>result case index</i>
long	LoadLevelOrModeShapeOrTimeStep	<i>load level or mode shape or time step index of the result block within the result case: for load level: $(0 < \text{LoadLevelOrModeShapeOrTimeStep} \leq \text{LoadLevelCount}$ for mode shape: $0 < \text{LoadLevelOrModeShapeOrTimeStep} \leq \text{ModeShapeCount}$ for time step: $0 < \text{LoadLevelOrModeShapeOrTimeStep} \leq \text{TimeStepCount}$</i>
)		
	RSeismicEq = (
double	seEpsX	<i>seismic equivalent coefficient in global x direction (% of participated mass)</i>
double	seEpsY	<i>seismic equivalent coefficient in global y direction (% of participated mass)</i>
double	seEpsZ	<i>seismic equivalent coefficient in global z direction (% of participated mass)</i>
)		
	RGlobalForces = (
double	Fx, Fy, Fz	<i>x, y, z load components in global directions [kN]</i>
double	Mx, My, Mz	<i>moment components about the global x, y, z axis [kNm]</i>
)		
	RTotalLoads = (
RGlobalForces	ExternalForces	<i>sum of applied loads in all global directions</i>
RGlobalForces	Unbalancedloads	<i>sum of unbalanced loads in all global directions</i>
)		

JSON strings

OptionsJSON= (
string	<i>Description saved in metafile</i>
string	<i>Authors name saved in metafile</i>
long	<i>width of the metafile in pixels</i>
long	<i>height of the metafile in pixels</i>
string	<i>Main title shown above the chart</i>
string	<i>Subtitle shown below the chart</i>
boolean	<i>Determines whether main title and subtitle are shown</i>
integer(EGeneralAlignmentHorizontal)	<i>horizontal align of the labels</i>
boolean	<i>Determines whether show tick marks on axis X and Y</i>
boolean	<i>Determines whether show arrows on axis X and Y</i>
integer	<i>Determines size of arrows on axis X and Y</i>
string	<i>name of the used font, must be available in windows</i>
integer	<i>size of the used font</i>
double	<i>proportional margin at left (0 < margin < 0,5)</i>
double	<i>proportional margin at right (0 < margin < 0,5)</i>
double	<i>proportional margin at top (0 < margin < 0,5)</i>
double	<i>proportional margin at bottom (0 < margin < 0,5)</i>
string	<i>Title of axis x</i>
string	<i>Title of axis y</i>
double	<i>proportional (x values) elongation of x axis</i>
double	<i>proportional (x values) elongation of y axis</i>
boolean	<i>IF True, labelling of larger y values is preferred</i>
boolean	<i>Label strings can contain TeX format strings</i>
boolean	<i>Additional labels</i>
integer(EXYchartLabelingStyle)	<i>Style of labelling,length must be same as number of datasets</i>
array of integer(EXYchartFillType)	<i>Style of dataset,length must be same as number of datasets</i>
array of boolean	<i>whether show label,length must be same as number of datasets</i>
array of boolean	<i>whether label are absolute values,length must be same as number of datasets</i>
array of boolean	<i>Dataset with thick line,length must be same as number of datasets</i>
array of integer	<i>Colours of datasets,length must be same as number of datasets</i>
array of integer	<i>Colours of dataset fills,length must be same as number of datasets</i>
array of integer	<i>Labels of datasets,length must be same as number of datasets</i>
array of string	<i>Titles of datasets,length must be same as number of datasets.</i>
boolean	<i>Empty strings are not shown in the legend</i>
integer(EGeneralAlignmentHorizontal)	<i>Determines whether show the legend</i>
integer(EGeneralAlignmentVertical)	<i>Determines horizontal alignment of the legend</i>
boolean	<i>Determines vertical alignment of the legend</i>
boolean	<i>Determines whether show the grid</i>
double	<i>Determines whether show frame around the chart</i>
double	<i>Determines the spacing of tick on axis x</i>
)	<i>Determines the spacing of tick on axis y</i>
PropertiesJSON= (
integer	<i>horizontal pixel coordinate of the origin (0,0)</i>
integer	<i>vertical pixel coordinate of the origin (0,0)</i>
double	<i>horizontal scale (pixels per unit)</i>
double	<i>vertical scale (pixels per unit)</i>
array of double	<i>x coordinates (units) of labels</i>
array of integer	<i>dataset index (starts with 1) of shown labels</i>
array of integer	<i>indexes of the points (starts with 1) in datasets of shown labels</i>
)	

Functions

long **GetAllFrequencies** ([in] EAnalysisType AnalysisType, [in] long ResultCase, [out] SAFEARRAY(double)* Frequencies)

AnalysisType analysis type, must be atLinearVibration or atNonLinearVibration
ResultCase result case index
($0 < \text{ResultCase} \leq \text{ResultCaseCount}[\text{AnalysisType}]$)
Frequencies array of frequencies
(length of the array is ModeShapeCount[AnalysisType,ResultCase])

Retrieves frequencies calculated from linear or nonlinear analysis.

Error codes are [reInvalidAnalysisType](#) and [reResultCaseIndexOutOfBounds](#).

long **GetAllModalMassfactors** ([in] EAnalysisType AnalysisType, [in] long ResultCase, [out] SAFEARRAY(double)* ModalMassfactors)

AnalysisType analysis type, must be atLinearVibration or atNonLinearVibration
ResultCase result case index
($0 < \text{ResultCase} \leq \text{ResultCaseCount}[\text{AnalysisType}]$)
ModalMassfactors array of modal mass factors
(length of the array is ModeShapeCount[AnalysisType,ResultCase])

Retrieves modal mass factors calculated from linear or nonlinear analysis.

Error codes are [reInvalidAnalysisType](#) and [reResultCaseIndexOutOfBounds](#).

long **GetAllActivatedMasses** ([in] EAnalysisType AnalysisType, [in] long ResultCase, [out] SAFEARRAY(RPoint3D)* ActivatedMasses)

AnalysisType analysis type, must be atLinearVibration or atNonLinearVibration
ResultCase result case index
($0 < \text{ResultCase} \leq \text{ResultCaseCount}[\text{AnalysisType}]$)
ActivatedMasses array of activated masses in global coordinate system (x, y, z, respectively)
(for further details from activated masses see the AxisVM User's manual)

Retrieves position of activated masses calculated from linear or nonlinear analysis. If successful it returns with the length of the array ActivatedMasses.

Error codes are [reInvalidAnalysisType](#) and [reResultCaseIndexOutOfBounds](#).

long **GetUsedMassOfNodes** ([i/o] SAFEARRAY(long) NodeIds, [in] EAnalysisType AnalysisType, [in] long ResultCase, [out] SAFEARRAY(RPoint3D)* UsedMass)

NodeIds nodes' indexes
AnalysisType analysis type, must be atLinearVibration or atNonLinearVibration
ResultCase result case index
($0 < \text{ResultCase} \leq \text{ResultCaseCount}[\text{AnalysisType}]$)
UsedMass array of used masses in vibration analysis in global coordinate system (x, y, z, respectively)

Retrieves position of used masses calculated from linear or nonlinear analysis. If successful it returns with the length of the array UsedMass.

Error codes are [reInvalidAnalysisType](#) and [reResultCaseIndexOutOfBounds](#).

long **GetAllNcr** ([in] long ResultCase, [out] SAFEARRAY(double)* Ncr)

ResultCase result case index
($0 < \text{ResultCase} \leq \text{ResultCaseCount}[\text{atBuckling}]$)
Ncr array of critical load factors
(length of the array is ModeShapeCount[atBuckling,ResultCase])

Retrieves critical load factors calculated from buckling analysis.

Error code is [reResultCaseIndexOutOfBounds](#) or [reNLpackageNotAvailable](#).

long	GetBucklingParameters ([in] long ResultCase, [i/o] RBuckling BucklingParameters)
	ResultCase result case index $(0 < ResultCase \leq ResultCaseCount[atBuckling])$
	BucklingParameters parameters of the buckling analysis
	Retrieves parameters of a buckling analysis.
	Possible error code: reResultCaseIndexOutOfBounds or reNLpackageNotAvailable
long	GetCapacityCurve ([in] ECapacityCurveType CapacityCurveType, [in] long LoadCaseld, [in] RSpectrumData SpectrumData, [out] SAFEARRAY(double)* X, [out] SAFEARRAY(double)* Y)
	CapacityCurveType Type of capacity curve
	LoadCaseld load case index ($0 < LoadCaseld \leq \text{AxisVMLoadCases.Count}$)
	SpectrumData Spectrum data
	X Array with x coordinates of the capacity curve
	Y Array with y coordinates of the capacity curve
	If successful returns number of points in the capacity curve, otherwise returns an error code (errDatabaseNotReady , reInvalidResponseSpectraParam , reITAReductionCriterionNotSatisfied , reInvalidLoadCaseType , reLoadCasesOutOfBounds , reInvalidNationalDesignCode , reSE2moduleNotAvailable)
	PLEASE NOTE:
	Number of points depends on number of increments set in RPushOverAnalysis .
long	GetCapacityCurvePushOver ([in] ECapacityCurveType CapacityCurveType, [in] long LoadCaseld, [in] RSpectrumData SpectrumData, [out] SAFEARRAY(double)* X, [out] SAFEARRAY(double)* Y)
	CapacityCurveType Type of capacity curve
	LoadCaseld load case index ($0 < LoadCaseld \leq \text{AxisVMLoadCases.Count}$)
	SpectrumData Spectrum data
	X Array with x coordinates of the capacity curve
	Y Array with y coordinates of the capacity curve
	Same as GetCapacityCurve , but it is using the parametric spectrum parameters set in SpectrumPushOver property. If successful returns number of points in the capacity curve, otherwise returns an error code (errDatabaseNotReady , reInvalidResponseSpectraParam , reITAReductionCriterionNotSatisfied , reInvalidLoadCaseType , reLoadCasesOutOfBounds , reInvalidNationalDesignCode , reSE2moduleNotAvailable)
	PLEASE NOTE:
	Number of points depends on number of increments set in RPushOverAnalysis .
long	GetFrequencyParameters ([in] EAnalysisType AnalysisType, [in] long ResultCase, [in] long FrequencyId, [i/o] RFrequencyParameters FrequencyParameters)
	AnalysisType analysis type must be <i>atLinearVibration</i> or <i>atNonLinearVibration</i>
	ResultCase result case index $(0 < ResultCase \leq ResultCaseCount[AnalysisType])$
	FrequencyId mode shape (frequency) index $(0 < FrequencyId \leq ModeShapeCount[AnalysisType, ResultCase])$
	FrequencyParameters parameters of the vibration analysis
	Retrieves parameters of a vibration analysis of a given shape.
	If the iteration was successful, <i>FrequencyParameters.EigenValConv</i> < <i>FrequencyParameters.EigenValConvLimit</i> , <i>FrequencyParameters.EigenVecConv</i> < <i>FrequencyParameters.EigenVecConvLimit</i> . Error codes are reInvalidAnalysisType , reFrequencyIndexOutOfBounds and reResultCaseIndexOutOfBounds .

long	GetNcrParameters ([in] long ResultBlock , [in] long ModeShapeId , [i/o] RNcrParameters NcrParameters)
	<p style="margin-left: 20px;">ResultCase <i>result case index</i> ($0 < ResultCase \leq ResultCaseCount[atBuckling]$)</p> <p style="margin-left: 20px;">ModeShapeId <i>mode shape (frequency) index</i> ($0 < ModeShapeId \leq ModeShapeCount[atBuckling, ResultBlock]$)</p> <p style="margin-left: 20px;">NcrParameters <i>parameters of the buckling analysis</i></p>
	<p><i>Retrieves parameters of a buckling analysis of a given buckling shape.</i></p> <p><i>If the iteration was successful,</i></p> <p><i>NcrParameters.EigenValConv < NcrParameters.EigenValConvLimit,</i></p> <p><i>NcrParameters.EigenVecConv < NcrParameters.EigenVecConvLimit.</i></p> <p><i>Error codes are reNL_packageNotAvailable, reModeShapeIndexOutOfBounds or reResultCaseIndexOutOfBounds.</i></p>
long	GetNonLinearAnalysisParameters ([in] long ResultCase , [i/o] RNonLinearAnalysis NonLinearAnalysisParameters)
	<p style="margin-left: 20px;">ResultCase <i>result case index</i> ($0 < ResultCase \leq ResultCaseCount[atNonLinearStatic]$)</p> <p style="margin-left: 20px;">NonLinearAnalysisParameters <i>parameters of the nonlinear analysis</i></p>
	<p><i>Retrieves parameters of a nonlinear analysis if successful, otherwise an error code (reResultCaseIndexOutOfBounds or reNL_packageNotAvailable).</i></p>
long	GetNonLinearAnalysisResultInfo ([in] long ResultCase , [in] long LoadLevel , [i/o] RNonLinearAnalysisResultInfo NonLinearAnalysisResultInfo)
	<p style="margin-left: 20px;">ResultCase <i>result case index</i> ($0 < ResultCase \leq ResultCaseCount[atNonLinearStatic]$)</p> <p style="margin-left: 20px;">LoadLevel <i>load level (increment) index</i> ($0 < LoadLevel \leq LoadLevelCount[atNonLinearStatic, ResultCase]$)</p> <p style="margin-left: 20px;">NonLinearAnalysisResultInfo <i>result information of a load level</i></p>
	<p><i>Retrieves result information for a load level of a nonlinear analysis if successful, otherwise an error code (reNL_packageNotAvailable, reLoadLevelIndexOutOfBounds or reResultCaseIndexOutOfBounds)</i></p>
long	GetModeActive ([in] EAnalysisType AnalysisType , [in] long ResultCase , [in] long FrequencyId , [out] ELongBoolean Value)
	<p style="margin-left: 20px;">AnalysisType <i>analysis type</i></p> <p style="margin-left: 20px;">ResultCase <i>result case index</i> ($0 < ResultCase \leq ResultCaseCount[atLinearVibration \text{ or } atNonLinearStatic]$)</p> <p style="margin-left: 20px;">FrequencyId <i>mode shape (frequency) index</i> ($0 < FrequencyId \leq ModeShapeCount[AnalysisType, ResultCase]$)</p> <p style="margin-left: 20px;">Value <i>True if mass active for this mode shape (frequency)</i></p>
	<p><i>Returns FrequencyId if successful, otherwise an error code (reInvalidAnalysisType, errDatabaseNotReady, reFrequencyIndexOutOfBounds, reResultCaseIndexOutOfBounds)</i></p>
long	GetNonLinearVibrationParameters ([in] long ResultCase , [i/o] RVibration NonLinearVibrationParameters)
	<p style="margin-left: 20px;">ResultCase <i>result case index</i> ($0 < ResultCase \leq ResultCaseCount[atNonLinearVibration]$)</p> <p style="margin-left: 20px;">NonLinearVibrationParameters <i>parameters of the nonlinear vibration analysis</i></p>
	<p><i>Retrieves parameters of a nonlinear vibration analysis.</i></p> <p><i>Possible error code: reResultCaseIndexOutOfBounds or reNL_packageNotAvailable</i></p>

long **GetResultsValid** ([in] [EAnalysisType](#) AnalysisType, [in] long LoadCombinationId, [out] [ELongBoolean](#) ResultsValid)

AnalysisType analysis type
 LoadCombinationId load combination index
 ResultsValid results are valid or invalid

Note: If one changes the model, the FE analysis has to be repeated. This function helps to get information whether the results of last calculation are valid or invalid. If successful it returns with load combination's index. Error codes are: [errIndexOutOfBounds](#), [errDatabaseNotReady](#).

long **GetSectionCoordinates** ([i/o] SAFEARRAY(long)* ContinousMemberIds, [in] [EAnalysisType](#) AnalysisType, [in] [ELongBoolean](#) AbsX, [out] SAFEARRAY(long)* LineIds, [out] SAFEARRAY(long)* SectionCounts, [out] SAFEARRAY(double)* SectionCoorindates)

ContinousMemberIds array of member indices
 AnalysisType analysis type
 AbsX
 LineIds array of line indices
 SectionCounts array of section count of members
 SectionCoorindates array of coordinates of sections along the continuous member

Returns with section coordinates, section counts and line indices of members given in ContinousMemberIds array that compose a continuous member. If successful it returns with number of members. Error codes are: [errIndexOutOfBounds](#), [errDatabaseNotReady](#), [errNoResults](#).

long **GetSeismicEqCoeff** ([in] [EAnalysisType](#) AnalysisType, [in] long ResultCase, [in] long FrequencyId, [i/o] [RSeismicEq](#) Value)

AnalysisType analysis type
 must be atLinearVibration or atNonLinearVibration
 ResultCase result case index
 (0 < ResultCase ≤ ResultCaseCount[atLinearVibration or atNonLinearStatic])
 FrequencyId mode shape (frequency) index
 (0 < FrequencyId ≤ ModeShapeCount[AnalysisType,ResultCase])
 Value Seismic equivalent coefficient

Returns FrequencyId if successful, otherwise an error code ([reInvalidAnalysisType](#), [errDatabaseNotReady](#), [reFrequencyIndexOutOfBounds](#), [reResultCaseIndexOutOfBounds](#))

long **GetTotalLoadsByLoadCaselD** ([in] [EAnalysisType](#) AnalysisType, [in] long LoadCaselD, [out] [ELongBoolean](#) UnbalancedLoadsExist, [i/o] [RTotalLoads](#) TotalLoads)

AnalysisType analysis type
 must be atLinearVibration or atNonLinearVibration
 LoadCaselD load case index
 UnbalancedLoadsExist lbTrue if AxisVM model contains unbalanced loads
 TotalLoads All loads in global directions

Returns total and unbalanced loads, if successful, otherwise an error code ([reInvalidAnalysisType](#), [errDatabaseNotReady](#), [reMissingAnalysisResults](#))

long **GetVibrationParameters** ([in] long ResultCase, [i/o] [RVibration](#) VibrationParameters)

ResultCase result case index
 (0 < ResultCase ≤ ResultCaseCount[atLinearVibration])
 VibrationParameters parameters of the linear vibration analysis

Retrieves parameters of a linear vibration analysis. Possible error code: [reResultCaseIndexOutOfBounds](#).

long	GetXYchartOptionsJSON ([in] ELongBoolean Indent , [in] ELongBoolean Escape , [out] BSTR OptionsJSON)
	<p>Indent determines indents usage in the JSON string Escape determines usage of escape in the JSON string OptionsJSON json string with charts options (labelling, visual, etc.), length of the arrays is 1 with lower bound 0.</p> <p>Get default properties of the XY chart. Returns number all points from all length of the JSON string.</p>
long	SaveXYchartToMetaFile ([in] BSTR FileName , [in] SAFEARRAY(long)* PointsCount , [in] SAFEARRAY(double)* X , [in] SAFEARRAY(double)* Y , [in] SAFEARRAY(BSTR)* Ylabels , [in] SAFEARRAY(double)* Xmarkers , [in] BSTR OptionsJSON , [out] BSTR PropertiesJSON)
	<p>FileName name of the enhanced metafile (*.emf) PointsCount array with number of points in each dataset (length is same as number of datasets) X x values in all datasets (length is sum of all points in all of datasets) Y y values in all datasets (length is sum of all points in all of datasets) Ylabels labels in all datasets (length is sum of all points in all of datasets) Xmarkers x values of vertical markers shown as dashed lines OptionsJSON json string with charts options (labelling, visual, etc.) PropertiesJSON json string with charts properties (pixel coordinates of origin, shown labels)</p> <p>Generates enhanced metafile containing XY chart from points in datasets. Returns number all points from all datasets. Possible error code: errDatabaseNotReady.</p>
long	SetModeActive ([in] EAnalysisType AnalysisType , [in] long ResultCase , [in] long FrequencyId , [in] ELongBoolean Value)
	<p>AnalysisType analysis type <i>must be atLinearVibration or atNonLinearVibration</i></p> <p>ResultCase result case index <i>(0 < ResultCase ≤ ResultCaseCount[atLinearVibration or atNonLinearStatic])</i></p> <p>FrequencyId mode shape (frequency) index <i>(0 < FrequencyId ≤ ModeShapeCount[AnalysisType,ResultCase])</i></p> <p>Value True if mass active for this mode shape (frequency)</p> <p>Returns FrequencyId if successful, otherwise an error code (reInvalidAnalysisType, errDatabaseNotReady, reFrequencyIndexOutOfBounds, reResultCaseIndexOutOfBounds)</p>
long	SetUserCreep ([in] ELongBoolean Creep)
	<p>Creep If lbTrue concrete creep will be considered in results of nonlinear analysis, if national design code allows it. More here...</p> <p>Enable or disable consideration of concrete creep in nonlinear analysis results. If successful returns 1, otherwise an error code (errDatabaseNotReady, errCreepNotSupported).</p>
long	UpdateResults
	<p>Updates all results , to consider changes in input values without re-analysis. Returns 1 if succesfull, otherwise an error code (errDatabaseNotReady)</p>

Properties

IAxisVMAcceleration *	Acceleration Get acceleration results of the model (extension module DYN is required)
IAxisVMCalculatedReinforcement *	CalculatedReinforcement Get calculated reinforcement type of results of the model
IAxisVMCrackWidth *	CrackWidth Get crack widths of surface elements (extension module RC1 is required)
IAxisVMDisplacements *	Displacements displacement results of the model

IAxisVMForces *		Forces internal force results of the model
double	Frequency [[in] EAnalysisType AnalysisType , [in] long ResultCase , [in] long FrequencyID] Frequency of a given mode shape. <i>AnalysisType must be atLinearVibration or atNonLinearVibration, (0 < ResultCase ≤ ResultCaseCount[AnalysisType]) (0 < FrequencyID ≤ ModeShapeCount[AnalysisType, ResultCase]) Error codes are reInvalidAnalysisType, reResultCaseIndexOutOfBounds or reFrequencyIndexOutOfBounds.</i>	
long	LoadCaseId [[in] EAnalysisType AnalysisType , [in] long ResultCase , [out] ErrorCode] Load case index of a result case of a specific analysis type. Result is valid only if ErrorCode > 0. ErrorCode can be reInvalidAnalysisType and reResultCaseIndexOutOfBounds . If ResultCase belongs to a load combination reResultCasesIsNotLoadCase is returned.	
long	LoadCombinationId [[in] EAnalysisType AnalysisType , [in] long ResultCase] Load combination index of a result case of a specific analysis type (0 < ResultCase ≤ ResultCaseCount[AnalysisType]) or an error code. Input error codes are reInvalidAnalysisType and reResultCaseIndexOutOfBounds . If ResultCase belongs to a load case reResultCasesIsNotLoadCombination is returned.	
long	LoadLevelCount [[in] EAnalysisType AnalysisType , [in] long ResultCase] Total number of load levels in a nonlinear analysis (atNonLinearStatic) (0 < ResultCase ≤ ResultCaseCount[AnalysisType]) or an error code. Error codes are reInvalidAnalysisType and reResultCaseIndexOutOfBounds .	
long	ModeShapeCount [[in] EAnalysisType AnalysisType , [in] long ResultCase] Total number of calculated mode shapes in a vibration or buckling analysis (atLinearVibration, atNonLinearVibration, atBuckling) (0 < ResultCase ≤ ResultCaseCount[AnalysisType]) or an error code. Error codes are reInvalidAnalysisType and reResultCaseIndexOutOfBounds .	
double	Ncr [[in] EAnalysisType AnalysisType , [in] long ResultCase , [in] long ModeShapeId] Critical load factor of a given buckling shape. <i>AnalysisType must be atBuckling, (0 < ResultCase ≤ ResultCaseCount[atBuckling]) (0 < ModeShapeID ≤ ModeShapeCount[AnalysisType, ResultCase]) Error codes are reNLpackageNotAvailable, reInvalidAnalysisType, reResultCaseIndexOutOfBounds or reModeShapeIndexOutOfBounds .</i>	
IAxisVMReinforcementCheck *		ReinforcementCheck Get reinforcement check type of results of the model
long	ResultCaseCount [[in] EAnalysisType AnalysisType] Number of result cases for a specific analysis typ. Returns 0 if model does not have results.	

long	ResultCaseOfLoadCase [[in] EAnalysisType AnalysisType, [in] long LoadCaseId] <i>Result case index of a given load case for a specific analysis type $(0 < \text{LoadCaseId} \leq \text{IAxisVMLoadCases.Count})$ or an error code. Error codes are reInvalidAnalysisType and reLoadCasesOutOfBounds.</i>
long	ResultCaseOfLoadCombination [[in] EAnalysisType AnalysisType, [in] long LoadCombination] <i>Result case index of a given load combination for a specific analysis type $(0 < \text{LoadCombination} \leq \text{IAxisVMLoadCombinations.Count})$ or an error code. Error codes are reInvalidAnalysisType and reLoadCombinationIsOutOfBounds.</i>
IAxisVMShearCapacity *	ShearCapacity Get shear capacity type of results of the model (extension module RC3 is required)
IAxisVMStresses *	Stresses stress results of the model
IAxisVMSteelDesignResults *	SteelDesignResults Get steel design results of the model
long	TimeStepCount [[in] EAnalysisType AnalysisType, [in] long ResultCase] <i>Total number of time steps in a dynamic analysis (atDynamic) $(0 < \text{ResultCase} \leq \text{ResultCaseCount[AnalysisType]})$ or an error code. Error codes are reInvalidAnalysisType and reResultCaseIndexOutOfBounds.</i>
IAxisVMTimberDesignResults *	TimberDesignResults Get timber design results of the model (extension module TD1 is required)
ELongBoolean	UserCreep <i>Returns lbTrue if nonlinear analysis results consider concrete creep. More here...</i>
IAxisVMVelocity *	Velocity Get velocity results of the model (extension module DYN is required)

Overview of AxisVM result objects

Result storage objects have some common properties.

Important Note:

Before calling reading functions all properties of the interface must be set because some functions use these properties .

By setting these properties:

AnalysisType, ...Component (LineForceComponent, SurfaceForceComponent, etc.), LoadCaseId, LoadCombinationId, *LoadLevelOrModeShapeOrTimeStep or LoadLevelOrTimeStep*, EnvelopeUID, CombinationType, MinMaxType,...) the desired subset of results can be conveniently selected.

LoadLevelOrModeShapeOrTimeStep or LoadLevelOrTimeStep property should be set according to the following table:

Analysis Type	Value
atLinearStatic	1
atNonLinearStatic	$0 < \text{LoadLevel} \leq \text{LoadLevelCount}[\text{atNonLinearStatic}, \text{ResultBlockOfLoadCase}[\text{atNonLinearStatic}, \text{LoadCaseId}]]$
atLinearVibration	$0 < \text{ModeShape} \leq \text{ModeShapeCount}[\text{atLinearVibration}, \text{ResultBlockOfLoadCase}[\text{atLinearVibration}, \text{LoadCaseId}]]$
atNonLinearVibration	$0 < \text{ModeShape} \leq \text{ModeShapeCount}[\text{atNonLinearVibration}, \text{ResultBlockOfLoadCase}[\text{atNonLinearVibration}, \text{LoadCaseId}]]$
atBuckling	$0 < \text{ModeShape} \leq \text{ModeShapeCount}[\text{atBuckling}, \text{ResultBlockOfLoadCase}[\text{atBuckling}, \text{LoadCaseId}]]$
atDynamic	$0 < \text{TimeStep} \leq \text{TimeStepCount}[\text{atDynamic}, \text{ResultBlockOfLoadCase}[\text{atDynamic}, \text{LoadCaseId}]]$

Single LOCATION reader functions

There are four ways of getting **one** result for a given element location (node, location along the member, support, point of a surface, etc.).

xxx is a placeholder for a special result name (e.g. NodalDisplacement or LineSupportForce)

Single result reader functions	Description
xxxByLoadCaseId	Get xxx results for a certain element location in the load case identified by the LoadCaseId property in the subject interface (IAxisVMDisplacements, IAxiVMForces, etc). <i>LoadLevelOrModeShapeOrTimeStep or LoadLevelOrTimeStep property must be set before calling these functions.</i>
xxxByLoadCombinationId	Get xxx results for a certain element location in the load combination identified by the LoadCombinationId property in the subject interface (IAxisVMDisplacements, IAxiVMForces, etc). <i>LoadLevelOrModeShapeOrTimeStep or LoadLevelOrTimeStep property must be set before calling these functions.</i>
Envelopexxx	Get envelope value of xxx results for a certain element location according to the xxx type EnvelopeUID , ... Component and MinMaxType (if available) properties. Name of the load case or combination belonging to resulting value is also retrieved.
Envelopexxx2	Same as Envelopexxx but also load case or load combination index belonging to resulting value is retrieved.
Criticalxxx	Get critical value of xxx results for a certain element location according to the xxx type CombinationType , ... Component and MinMaxType (if available) properties. The description string of the actual combination belonging to resulting value is also retrieved.
Criticalxxx2	Same as Criticalxxx but also partial factors, load case indexes and the critical combination type belonging to resulting value are retrieved.

SINGLE element reader functions

Certain elements like beams or surfaces have multiple locations (sections along, surface points). For these elements there are four ways of getting **all results for a given element**. String of the critical combination in which Component has its minimum or maximum at a certain location can be read using the respective single location reader function.

xxx is a placeholder for a special result name (e.g. NodalDisplacement or LineSupportForce)

Single element reader functions	Description
---------------------------------	-------------

xxxSByLoadCaseId	Get xxx results in all locations of a given element in the load case identified by the LoadCaseId property in subject interface (IAxisVMDisplacements, IAxiVMForces, etc). <i>LoadLevelOrModeShapeOrTimeStep or LoadLevelOrTimeStep property must be set before calling these functions.</i>
xxxSByLoadCombinationId	Get xxx results in all locations of a given line element in the load combination identified by the LoadCombinationId property in the subject interface (IAxisVMDisplacements, IAxiVMForces, etc). <i>LoadLevelOrModeShapeOrTimeStep or LoadLevelOrTimeStep property must be set before calling these functions.</i>
EnvelopexxxS	Get all envelope values of xxx results for a given element according to the EnvelopeUID , ... Component and MinMaxType (if available) properties.
CriticalxxxS	Get all critical values of xxx results for a given element according to the CombinationType , ... Component and MinMaxType (if available) properties.

Multiple element reader functions

There are four ways of getting **results for all elements** with one function call, which can be much more efficient than calling the single element or single location reader COM-function many times. String of the critical combination in which Component has its minimum or maximum at a certain location and be read using the respective single location reader function.

<i>Multiple element reader functions</i>	<i>Description</i>
AllxxxSByLoadCaseId	Get xxx results for all elements in the load case identified by the LoadCaseId property in the subject interface (IAxisVMDisplacements, IAxiVMForces, etc).. <i>LoadLevelOrModeShapeOrTimeStep or LoadLevelOrTimeStep property must be set before calling these functions.</i>
AllxxxSByLoadCombinationId	Get xxx results for all elements in the load case identified by the LoadCombinationId property in subject interface (IAxisVMDisplacements, IAxiVMForces, etc). <i>LoadLevelOrModeShapeOrTimeStep or LoadLevelOrTimeStep property must be set before calling these functions.</i>
AllEnvelopexxxS	Get envelope value of xxx results for all elements according to the EnvelopeUID , ... Component and MinMaxType (if available) properties.
AllCriticalxxxS	Get critical value of xxx results for all elements according to the CombinationType , ... Component and MinMaxType (if available) properties. The description strings of critical combination are not retrieved.

Multiple BLOCK reader functions

There is also a way to retrieve results of a certain element location (node, beam cross-section, support, surface point etc.) for all result blocks according to **AnalysisType**.

xxxSForResultBlocks	For linear analysis, it gets results of all load cases, for nonlinear analysis it gets results of all load levels of all analysed load cases, for vibration and buckling it gets results of all vibration or modal shapes of all analysed load cases.
----------------------------	---

IAxisVMAcceleration

Interface containing acceleration results within the model.

If property returning this interface is null (nil) then the extension module DYN is not available.

Error codes

```
enum EAxleAccelerationError = {
    aeLoadCaseIdIndexOutOfBounds = -100001
    aeInvalidCombinationOfLoadCaseAndTimeStep = -100002
    aeInvalidAnalysisType = -100003
    aeLoadCombinationHasNoDynamicResult = -100004 }
```

*LoadCaseId is invalid
invalid combination of LoadCaseId and TimeStep
AnalysisType is incompatible with the function
dynamic results for the LoadCaseId are missing*

Enumerated types

```
enum EAxleAcceleration = {
    acX = 0
    acY = 1
    acZ = 2
    acXX = 3
    acYY = 4
    acZZ = 5
    acR = 6
    acRR = 7}
```

*acceleration in local x direction
acceleration in local y direction
acceleration in local z direction
angular acceleration about local x direction
angular acceleration about local y direction
angular acceleration about local z direction
resultant acceleration
resultant angular acceleration*

Acceleration types

Records / structures

```
RAxleAccelerationValues = (
    double avX    acceleration in local x direction [m/s2]
    double avY    acceleration in local y direction [m/s2]
    double avZ    acceleration in local z direction [m/s2]
    double avXX   angular acceleration about local x direction
                  [rad/s2]
    double avYY   angular acceleration about local y direction
                  [rad/s2]
    double avZZ   angular acceleration about local z direction
                  [rad/s2]
    double avR    resultant acceleration [m/s2]
    double avRR   resultant angular acceleration [rad/s2]
)
```

Functions

```
long NodalAccelerationByLoadCaseId ([in] long NodeId,
                                    [i/o] RAxleAccelerationValues AccelerationValues, [out] BSTR Combination)
```

NodeId node index or line midpoint index
(0 < NodeId ≤ AxisVMMModel.Nodes.Count) or a value returned by AxisVMMModel.Lines.MidpointId[LineIndex]

AccelerationValues acceleration results

Combination name of the load case

Retrieves acceleration values of a node or line midpoint according to the LoadCaseId (and TimeStep) property. Returns NodeId if successful I, otherwise an error code ([errDatabaseNotReady](#), [errIndexOutOfBoundsException](#), [aeLoadCaseIdIndexOutOfBoundsException](#), [aeInvalidCombinationOfLoadCaseAndTimeStep](#), [aeInvalidAnalysisType](#), [aeLoadCombinationHasNoDynamicResult](#)).

long **EnvelopeNodalAcceleration** ([in] long **Nodeld**,
 [i/o] [RAccelerationValues](#) **AccelerationValues**, [out] BSTR **Combination**)

Nodeld *node index or line midpoint index
 ($0 < \text{Nodeld} \leq \text{AxisVMMModel.Nodes.Count}$) or a value returned by
 $\text{AxisVMMModel.Lines.MidpointId[LineIndex]}$*

AccelerationValues *acceleration results*

Combination *name of the load case*

Retrieves acceleration values of a node or line midpoint according to the LoadCaseId (and TimeStep) property. Returns Nodeld if successful l, otherwise an error code ([errDatabaseNotReady](#), [errIndexOutOfBoundsException](#), [aeLoadCaseIdIndexOutOfBoundsException](#), [aeInvalidCombinationOfLoadCaseAndTimeStep](#), [aeInvalidAnalysisType](#), [aeLoadCombinationHasNoDynamicResult](#)).

Properties

[**EAnalysisType**](#) • Get or set type of analysis

[**EVelocitY**](#) • Get or set the velocity component. Used for envelope and critical results.

long [**EnvelopeUID**](#) • Get or set the unique index of the envelope used in functions for reading envelope results

long [**LoadCaseId**](#) • Get or set the load case index
 $(0 < \text{LoadCaseId} \leq \text{AxisVMMModel.LoadCases.Count})$

[**EMinMaxType**](#) • Get or set the if minimum or maximum values of the component will be read

long [**TimeStep**](#) • Get or set the time step increment.

IAxisVMCalculatedReinforcement

Interface for reading calculated reinforcement on surfaces. The AxisVM always calculates the absolute max. required reinforcements, see 6.5. R.C. Design in AxisVM manual for more info.

Prior to reading the calculated reinforcement values, reinforcement parameters must be set in interfaces [IAxisVMDomain](#) or [IAxisVMSurface](#).

Note 1:

Results depend on smoothing parameters which can be set with [SetCommonDisplayParameters](#) function. Set smoothing parameters in CommonDisplayParameters.MiselSettings record.

Note 2:

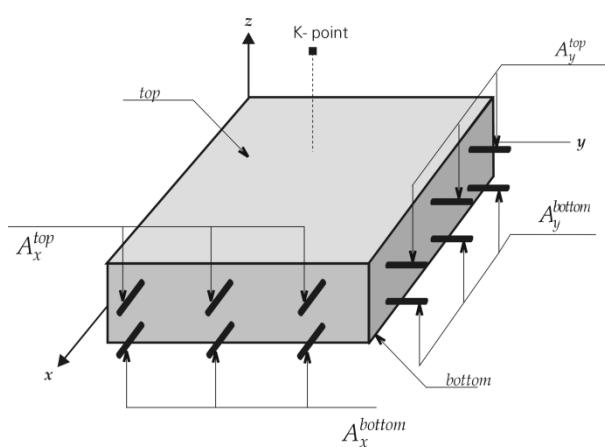
For defining actual reinforcement on surface elements and domains use interface [IAxisVMActualReinforcement](#).

Note 3:

If skew reinforcement is set for the domain or surface using RReinforcementParameters, x and y reinforcement directions are interpreted as ξ and η reinforcement directions, respectively.

Enumerated types

enum	EReinforcementStatus = {	
	rsOK = 0	Reinforcement OK
	rsComprReinforcementNeededX = 1	Compression reinforcement needed in x (or ξ) direction
	rsComprReinforcementNeededY = 2	Compression reinforcement needed in y (or η) direction
	rsCannotBeReinforcedX = 3	Cannot be reinforced according to code in x (or ξ) direction
	rsCannotBeReinforcedY = 4}	Cannot be reinforced according to code in y (or η) direction
		State of reinforcement.
enum	EReinforcement = {	
	rAsbx = 0	Bottom reinforcement in x (or ξ) direction
	rAsby = 1	Bottom reinforcement in y (or η) direction
	rAstx = 2	Top reinforcement in x (or ξ) direction
	rAsty = 3}	Top reinforcement in y (or η) direction
		Type of reinforcement.



Error codes

enum	ECalculatedReinforcementError = {	
	creCOMError = -100001	COM server error
	creLoadCaseIdIndexOutOfBounds = -100002	Invalid LoadCaseID while reading results
	creLoadCombinationIdIndexOutOfBounds = -100003	Invalid CombinationID while reading results
	creInvalidAnalysisType = -100004	Invalid type of results for used analysis

```

creCombinationTypeNotValidForCurrentNationalDesignCode
= -100005

creInvalidCombinationOfLoadCaseAndLoadLevel = -100006
creInvalidCombinationOfLoadCombinationAndLoadLevel = -
100007
}

```

The CombinationType cannot be used for the selected national code or results not available for CombinationType. Some design codes allow only certain ECombinationTypes:
-ndcHungarian MSZ, ndcDutch NEN,
ndcRomanian STAS:
ctSLS1, ctULS1, ctULSALL
-other national design codes:
ctSLS1, ctSLS2, ctSLS3, ctULS1, ctULS2, ctULS3,
ctULSALL

Invalid combination LoadCaseID and load level
Invalid combination CombinationID and load level

Records / structures

RReinforcementValues= (
double	Asbx	<i>Bottom reinforcement in x (or ξ) direction [m²/m]</i>
double	Asby	<i>Bottom reinforcement in y (or η) direction [m²/m]</i>
double	Astx	<i>Top reinforcement in x (or ξ) direction [m²/m]</i>
double	Asty	<i>Top reinforcement in y (or η) direction [m²/m]</i>
EReinforcementStatus	AsbxStatus	<i>Status of bottom reinforcement in x (or ξ) direction</i>
EReinforcementStatus	AsbyStatus	<i>Status of bottom reinforcement in y (or η) direction</i>
EReinforcementStatus	AstxStatus	<i>Status of top reinforcement in x (or ξ) direction</i>
EReinforcementStatus	AstyStatus	<i>Status of top reinforcement in y (or η) direction</i>
)	

Note: If no reinforcement is needed, smooth graphical representation of reinforcement values may require the use of artificial negative reinforcement values that are calculated from the required reinforcement on the opposite side of the plate. These negative values can be used only for graphical purposes. In other cases, please consider that this negative value means that no reinforcement is needed.

RReinforcements= (
long	ContourPointCount	<i>Number of contour points</i>
long	ContourPoint1Id	<i>index of contour point 1</i>
long	ContourPoint2Id	<i>index of contour point 2</i>
long	ContourPoint3Id	<i>index of contour point 3</i>
long	ContourPoint4Id	<i>index of contour point 4</i>
long	ContourLine1Id	<i>index of contour line midpoint 1</i>
long	ContourLine2Id	<i>index of contour line midpoint 2</i>
long	ContourLine3Id	<i>index of contour line midpoint 3</i>
long	ContourLine4Id	<i>index of contour line midpoint 4</i>
RReinforcementValues	rvCenterPoint	<i>Reinforcement at centre point</i>
RReinforcementValues	rvContourPoint1	<i>Reinforcement at contour point 1</i>
RReinforcementValues	rvContourPoint2	<i>Reinforcement at contour point 2</i>
RReinforcementValues	rvContourPoint3	<i>Reinforcement at contour point 3</i>
RReinforcementValues	rvContourPoint4	<i>Reinforcement at contour point 4</i>
RReinforcementValues	rvContourLineMidPoint1	<i>Reinforcement at contour line midpoint 1</i>
RReinforcementValues	rvContourLineMidPoint2	<i>Reinforcement at contour line midpoint 2</i>
RReinforcementValues	rvContourLineMidPoint3	<i>Reinforcement at contour line midpoint 3</i>
RReinforcementValues	rvContourLineMidPoint4	<i>Reinforcement at contour line midpoint 4</i>
)	

Functions

long **GetCalculatedReinforcementsByLoadCaselD** ([in] long **SurfacelD**, [in] long **LoadCaselD**,
 [in] long **LoadLevel**, [in] [EAnalysisType](#) **AnalysisType**, [i/o] [RReinforcements](#)* **Reinforcements**,
 [out] BSTR* **Combination**)

SurfacelD	<i>index of the surface element</i>
LoadCaselD	<i>load case index ($0 < LoadCaselD \leq AxisVMLoadCases.Count$)</i>
LoadLevel	<i>load level (increment) index</i>
AnalysisType	<i>Type of Analysis</i>
Reinforcements	<i>Reinforcement of the surface element</i>
Combination	<i>String with name of load case.</i>

If successful returns SurfacelD, otherwise an error code ([errDatabaseNotReady](#),
[errNotSupportedByNationalDesignCode](#), [errIndexOutOfBoundsException](#),
[creInvalidCombinationOfLoadCaseAndLoadLevel](#), [creInvalidAnalysisType](#)).

long **GetCalculatedReinforcementsByLoadCombinationId** ([in] long **SurfacelD**,
 [in] long **LoadCombinationId**, [in] long **LoadLevel**, [in] [EAnalysisType](#) **AnalysisType**,
 [i/o] [RReinforcements](#)* **Reinforcements**, [out] BSTR* **Combination**)

SurfacelD	<i>index of the surface element</i>
LoadCombinationId	<i>load combination index ($0 < LoadCombinationId \leq AxisVMLoadCombinations.Count$)</i>
LoadLevel	<i>load level (increment) index</i>
AnalysisType	<i>Type of Analysis</i>
Reinforcements	<i>Reinforcement of the surface element</i>
Combination	<i>String with name of combination</i>

If successful returns SurfacelD, otherwise an error code ([errDatabaseNotReady](#),
[errNotSupportedByNationalDesignCode](#), [errIndexOutOfBoundsException](#),
[creLoadCombinationIdIndexOutOfBoundsException](#),
[creInvalidCombinationOfLoadCombinationAndLoadLevel](#), [creInvalidAnalysisType](#)).

long **GetEnvelopeCalculatedReinforcements** ([in] long **SurfacelD**, [in] [EAnalysisType](#) **AnalysisType**,
 [i/o] [RReinforcements](#)* **Reinforcements**, [out] BSTR* **Combination**)

SurfacelD	<i>index of the surface element</i>
AnalysisType	<i>Type of Analysis</i>
Reinforcements	<i>Reinforcement of the surface element</i>
Combination	<i>Combination contain a multiline strings (separated with CR+LF) where the lines belong to: ContourPoint1, ContourLineMidPoint1, ContourPoint2, ContourLineMidPoint2, ContourPoint3, ContourLineMidPoint3, [ContourPoint4, ContourLineMidPoint4], CenterPoint</i>

Retrieves envelope results. Envelope is identified by Component , EnvelopeUID and MinMaxType properties.
 If successful returns SurfacelD, otherwise an error code ([errDatabaseNotReady](#),
[errNotSupportedByNationalDesignCode](#), [errIndexOutOfBoundsException](#), [creInvalidAnalysisType](#)).

long **GetEnvelopeCalculatedReinforcements2** ([in] long **SurfacelD**, [in] [EAnalysisType](#) **AnalysisType**,
 [i/o] [RReinforcements](#)* **Reinforcements**, [out] **LoadCaseOrCombinationId**, [out] long **LoadLevel**)

SurfacelD	<i>index of the surface element</i>
AnalysisType	<i>Type of Analysis</i>
Reinforcements	<i>Reinforcement of the surface element</i>
LoadCaseOrCombinationId	<i>load case or load combination index of surface midpoint, if index is > IAxisVMLoadcases.Count then Load combination index = LoadCaseOrCombinationId - IAxisVMLoadcases.Count</i>
LoadLevel	<i>load level</i>

Retrieves envelope results. Envelope is identified by Component , EnvelopeUID and MinMaxType properties.
 If successful returns SurfacelD, otherwise an error code ([errDatabaseNotReady](#),
[errNotSupportedByNationalDesignCode](#), [errIndexOutOfBoundsException](#), [creInvalidAnalysisType](#)).

long	GetCriticalCalculatedReinforcements ([in] long Surfaceld , [in] ECombinationType CombinationType , [in] EAnalysisType AnalysisType , [in] EReinforcement Component , [i/o] RReinforcements * Reinforcements , [out] BSTR* Combination)
	Surfaceld <i>index of the surface element</i>
	CombinationType <i>Combination Type</i>
	AnalysisType <i>Type of Analysis</i>
	Component <i>Reinforcement component</i>
	Reinforcements <i>Reinforcement of the surface element</i>
	Combination <i>Combination contain a multiline strings (separated with CR+LF) where the lines belong to: ContourPoint1, ContourLineMidPoint1, ContourPoint2, ContourLineMidPoint2, ContourPoint3, ContourLineMidPoint3, [ContourPoint4, ContourLineMidPoint4], CenterPoint</i>
	<i>If successful returns Surfaceld, otherwise an error code (errDatabaseNotReady, errNotSupportedByNationalDesignCode, errIndexOutOfBounds, creCombinationTypeNotValidForCurrentNationalDesignCode, creInvalidAnalysisType).</i>
long	GetCriticalCalculatedReinforcements2 ([in] long Surfaceld , [in] ECombinationType CombinationType , [in] EAnalysisType AnalysisType , [in] EReinforcement Component , [i/o] RReinforcements * Reinforcements , [out] ECombinationType CriticalCombinationType , [out] SAFEARRAY(double) Factors , [out] SAFEARRAY(long) LoadCaselds)
	Surfaceld <i>index of the surface element</i>
	CombinationType <i>Combination Type</i>
	AnalysisType <i>Type of Analysis</i>
	Component <i>Reinforcement component</i>
	Reinforcements <i>Reinforcement of the surface element</i>
	CriticalCombinationType <i>combination type corresponding to critical results</i>
	Factors <i>load factors of the critical load combination</i>
	LoadCaselds <i>load case indexes of the critical load combination</i>
	<i>If successful returns Surfaceld, otherwise an error code (errDatabaseNotReady, errNotSupportedByNationalDesignCode, errIndexOutOfBounds, creCombinationTypeNotValidForCurrentNationalDesignCode, creInvalidAnalysisType).</i>
long	GetAllCalculatedReinforcementsByLoadCaseld ([in] long LoadCaseld , [in] long LoadLevel , [in] EAnalysisType AnalysisType , [out] SAFEARRAY(RReinforcements)* Reinforcements , [out] SAFEARRAY (BSTR)* Combinations)
	LoadCaseld <i>load case index (0 < LoadCaseld ≤ AxisVMLoadCases.Count)</i>
	LoadLevel <i>load level (increment) index</i>
	AnalysisType <i>Type of Analysis</i>
	Reinforcements <i>Array of reinforcements for each surface element</i>
	Combinations <i>Array of strings with names of load case for all surface elements.</i>
	<i>If successful returns number of all surface elements, otherwise returns an error code (errDatabaseNotReady, errNotSupportedByNationalDesignCode, creInvalidCombinationOfLoadCaseAndLoadLevel, creInvalidAnalysisType).</i>

long **GetAllCalculatedReinforcementsByLoadCombinationId** ([in] long **LoadCombinationId**,
[in] long **LoadLevel**, [in] [EAnalysisType](#) **AnalysisType**,
[out] SAFEARRAY([RReinforcements](#))* **Reinforcements**,
[out] SAFEARRAY (BSTR)* **Combinations**)

LoadCombinationId *load combination index (0 < LoadCombinationId ≤ AxisVMLoadCombinations.Count)*
LoadLevel *load level (increment) index*
AnalysisType *Type of Analysis*
Reinforcements *Array of reinforcements for each surface element*
Combinations *Array of strings with combination names for all surface elements.*

*If successful returns number of all surface elements, otherwise returns an error code
([errDatabaseNotReady](#), [errNotSupportedByNationalDesignCode](#),
[creLoadCombinationIdIndexOutOfBounds](#),
[creInvalidCombinationOfLoadCombinationAndLoadLevel](#), [creInvalidAnalysisType](#)).*

long **GetAllEnvelopeCalculatedReinforcements** ([in] [EAnalysisType](#) **AnalysisType**,
[out] SAFEARRAY([RReinforcements](#))* **Reinforcements**,
[out] SAFEARRAY (BSTR)* **Combinations**)

AnalysisType *Type of Analysis*
Reinforcements *Array of reinforcements for each surface element*
Combinations *String array of multiline strings (separated with CR+LF) where the lines belong to: CenterPoint(X or ξ) bottom, Y (or η) bottom, X (or ξ) top, Y (or η) top) for all surface elements.*

*Retrieves envelope results. Envelope is identified by Component, EnvelopeUID and MinMaxType properties.
If successful returns number of all surface elements, otherwise returns an error code
([errDatabaseNotReady](#), [errNotSupportedByNationalDesignCode](#), [creInvalidAnalysisType](#)).*

long **GetAllCriticalCalculatedReinforcements** ([in] [ECombinationType](#) **CombinationType**,
[in] [EAnalysisType](#) **AnalysisType**, [in] [EReinforcement](#) **Component**,
[out] SAFEARRAY([RReinforcements](#))* **Reinforcements**,
[out] SAFEARRAY (BSTR)* **Combinations**)

CombinationType *Combination Type*
AnalysisType *Type of Analysis*
Component *Reinforcement component*
Reinforcements *Array of reinforcements for each surface element*
Combinations *String array of multiline strings (separated with CR+LF) where the lines belong to: ContourPoint1, ContourLineMidPoint1, ContourPoint2, ContourLineMidPoint2, ContourPoint3, ContourLineMidPoint3, [ContourPoint4, ContourLineMidPoint4], CenterPoint) for all surface elements.*

*If successful returns number of all surface elements, otherwise returns an error code
([errDatabaseNotReady](#), [errNotSupportedByNationalDesignCode](#), [creInvalidAnalysisType](#),
[creCombinationTypeNotValidForCurrentNationalDesignCode](#)).*

long **CalculatedReinforcementsByLoadCaseId** ([in] long **Surfaceld**,
[i/o] [RReinforcements](#)* **Reinforcements**, [out] BSTR* **Combination**)

Surfaceld *index of the surface element*
Reinforcements *Reinforcement of the surface element*
Combination *String with name of load case.*

*Get calculated reinforcement based on these properties: LoadCaseID, LoadLevel and AnalysisType.
If successful returns Surfaceld, otherwise an error code ([errDatabaseNotReady](#),
[errNotSupportedByNationalDesignCode](#), [errIndexOutOfBounds](#),
[creInvalidCombinationOfLoadCaseAndLoadLevel](#), [creInvalidAnalysisType](#)).*

long	CalculatedReinforcementsByLoadCombinationId ([in] long SurfaceId, [i/o] RReinforcements* Reinforcements, [out] BSTR* Combination)
	<p style="margin-left: 20px;">SurfaceId index of the surface element Reinforcements Reinforcement of the surface element Combination String with name of combination</p> <p>Get calculated reinforcement based on these properties: LoadCombinationID, LoadLevel and AnalysisType. If successful returns SurfaceId, otherwise an error code (errDatabaseNotReady, errNotSupportedByNationalDesignCode, errIndexOutOfBoundsException, creLoadCombinationIdOutOfBoundsException, creInvalidCombinationOfLoadCombinationAndLoadLevel, creInvalidAnalysisType).</p>
long	EnvelopeCalculatedReinforcements ([in] long SurfaceId, [i/o] RReinforcements* Reinforcements, [out] BSTR* Combination)
	<p style="margin-left: 20px;">SurfaceId index of the surface element Reinforcements Reinforcement of the surface element Combination Combination contain a multiline strings (separated with CR+LF) where the lines belong to: ContourPoint1, ContourLineMidPoint1, ContourPoint2, ContourLineMidPoint2, ContourPoint3, ContourLineMidPoint3, [ContourPoint4, ContourLineMidPoint4], CenterPoint</p> <p>Retrieves envelope results. Envelope is identified by Component, EnvelopeUID and MinMaxType properties. If successful returns number of all surface elements, otherwise an error code (errDatabaseNotReady, errNotSupportedByNationalDesignCode, errIndexOutOfBoundsException, creInvalidAnalysisType).</p>
long	EnvelopeCalculatedReinforcements2 ([in] long SurfaceId, [i/o] RReinforcements* Reinforcements, [out] long LoadCaseOrCombinationId, [out] long LoadLevel)
	<p style="margin-left: 20px;">SurfaceId index of the surface element Reinforcements Reinforcement of the surface element LoadCaseOrCombinationId load case or load combination index of surface midpoint, if index is > IAxisVMLoadcases.count then Load combination index = LoadCaseOrCombinationId - IAxisVMLoadcases.count LoadLevel load level</p> <p>Retrieves envelope results. Envelope is identified by Component, EnvelopeUID and MinMaxType properties. If successful returns number of all surface elements, otherwise an error code (errDatabaseNotReady, errNotSupportedByNationalDesignCode, errIndexOutOfBoundsException, creInvalidAnalysisType).</p>
long	CriticalCalculatedReinforcements ([in] long SurfaceId, [i/o] RReinforcements* Reinforcements, [out] BSTR* Combination)
	<p style="margin-left: 20px;">SurfaceId index of the surface element Reinforcements Array of reinforcements for each surface element Combinations Combination contain a multiline strings (separated with CR+LF) where the lines belong to: ContourPoint1, ContourLineMidPoint1, ContourPoint2, ContourLineMidPoint2, ContourPoint3, ContourLineMidPoint3, [ContourPoint4, ContourLineMidPoint4], CenterPoint</p> <p>Get calculated reinforcement based on these properties: LoadLevel and AnalysisType If successful returns number of all surface elements, otherwise returns an error code (errDatabaseNotReady, errNotSupportedByNationalDesignCode, creInvalidAnalysisType, creCombinationTypeNotValidForCurrentNationalDesignCode).</p>

long	CriticalCalculatedReinforcements2 ([in] long Surfaceld , [i/o] RReinforcements * Reinforcements , [out] ECombinationType CriticalCombinationType , [out] SAFEARRAY(double) Factors , [out] SAFEARRAY(long) LoadCaselds)
	Surfaceld <i>index of the surface element</i>
	Reinforcements <i>Reinforcement of the surface element</i>
	CriticalCombinationType <i>combination type corresponding to critical results</i>
	Factors <i>load factors of the critical load combination for midpoint results</i>
	LoadCaselds <i>load case indexes of the critical load combination for midpoint results</i>
	<i>If successful returns Surfaceld, otherwise an error code (errDatabaseNotReady, errNotSupportedByNationalDesignCode, errIndexOutOfBoundsException, creCombinationTypeNotValidForCurrentNationalDesignCode, creInvalidAnalysisType).</i>
long	AllCalculatedReinforcementsByLoadCaseld ([out] SAFEARRAY(RReinforcements)* Reinforcements , [out] SAFEARRAY (BSTR)* Combinations)
	Reinforcements <i>Reinforcement of the surface element</i>
	Combinations <i>Array of string with load case names for all surface elements.</i>
	<i>If successful I returns number of all surface elements, otherwise returns an error code (errDatabaseNotReady, errNotSupportedByNationalDesignCode, errIndexOutOfBoundsException, creInvalidCombinationOfLoadCaseAndLoadLevel, creInvalidAnalysisType).</i>
long	AllCalculatedReinforcementsByLoadCombinationId ([out] SAFEARRAY(RReinforcements)* Reinforcements , [out] SAFEARRAY (BSTR)* Combinations)
	Reinforcements <i>Array of reinforcements for each surface element</i>
	Combinations <i>Array of string with combination names for all surface elements.</i>
	<i>If successful returns number of all surface elements, otherwise returns an error code (errDatabaseNotReady, errNotSupportedByNationalDesignCode, creLoadCombinationIdIndexOutOfBoundsException, creInvalidCombinationOfLoadCombinationAndLoadLevel, creInvalidAnalysisType).</i>
long	AllEnvelopeCalculatedReinforcements ([out] SAFEARRAY(RReinforcements)* Reinforcements , [out] SAFEARRAY (BSTR)* Combinations)
	Reinforcements <i>Array of reinforcements for each surface element</i>
	Combinations <i>String array of multiline strings (separated with CR+LF) where the lines belong to: ContourPoint1, ContourLineMidPoint1, ContourPoint2, ContourLineMidPoint2, ContourPoint3, ContourLineMidPoint3, [ContourPoint4, ContourLineMidPoint4], CenterPoint for all surface elements</i>
	<i>Retrieves envelope results. Envelope is identified by Component , EnvelopeUID and MinMaxType properties. If successful returns number of all surface elements, otherwise returns an error code (errDatabaseNotReady, errNotSupportedByNationalDesignCode, creInvalidAnalysisType).</i>
long	AllCriticalCalculatedReinforcements ([out] SAFEARRAY(RReinforcements)* Reinforcements , [out] SAFEARRAY (BSTR)* Combinations)
	Reinforcements <i>Array of reinforcements for each surface element</i>
	Combinations <i>String array of multiline strings (separated with CR+LF) where the lines belong to: ContourPoint1, ContourLineMidPoint1, ContourPoint2, ContourLineMidPoint2, ContourPoint3, ContourLineMidPoint3, [ContourPoint4, ContourLineMidPoint4], CenterPoint for all surface elements</i>
	<i>If successful returns number of all surface elements, otherwise returns an error code (errDatabaseNotReady, errNotSupportedByNationalDesignCode, creInvalidAnalysisType, creCombinationTypeNotValidForCurrentNationalDesignCode).</i>

Properties

<u>EAnalysisType</u>	AnalysisType • Get or set the type of analysis
<u>EReinforcement</u>	Component • Get or set the reinforcement component
long	Count <i>Get number of surface elements in the model, if 0 then results not calculated or invalid.</i>
<u>ECombinationType</u>	CombinationType • Get or set the type of combination for critical results
long	EnvelopeUID • Get or set the unique index of the envelope used in functions for reading envelope results
long	LoadCaseId • Get or set the t load case index <i>(0 < LoadCaseId ≤ <u>AxisVMLoadCases</u>.Count)</i>
long	LoadCombinationId • Get or set the load combination index <i>(0 < LoadCaseId ≤ <u>AxisVMLoadCombinations</u>.Count)</i>
long	LoadLevel • Get or set the load level (increment) index

IAxisVMCrackWidth

Interface for reading crack width results. If property returning this interface is null (nil) then the extension module RC1 is not available. Actual reinforcement is considered in the calculations.

Enumerated types

enum	ECrackWidth = {	
	cwBottom = 0	<i>at bottom</i>
	cwTop = 1 }	<i>at top</i>
<i>Position of the crack</i>		
enum	ERebarType = {	
	rtRibbed = 0	<i>ribbed rebar</i>
	rtWelded = 1	<i>welded rebar</i>
	rtSmooth = 2 }	<i>smooth rebar</i>
<i>Type of rebar</i>		

Error codes

enum	ECrackWidthsError = {	
	cweCOMError = -100001	<i>COM server error</i>
	cweLoadCaseIdIndexOutOfBoundsException = -100002	<i>Invalid LoadCaseID while reading results</i>
	cweLoadCombinationIdIndexOutOfBoundsException = -100003	<i>Invalid CombinationID while reading results</i>
	cweInvalidAnalysisType = -100004	<i>Invalid type of results for used analysis</i>
	cweCombinationTypeNotValidForCurrentNationalDesignCode = -100005	<i>The CombinationType cannot be used for the selected national code. Some design codes allow only certain ECombinationTypes (see here) or results not available for CombinationType</i>
	cweInvalidCombinationOfLoadCaseAndLoadLevel = -100006	<i>Invalid combination LoadCaseID and load level</i>
	cweInvalidCombinationOfLoadCombinationAndLoadLevel = -100007 }	<i>Invalid combination CombinationID and load level or this</i>

Records / structures

	RCrackWidthValues = (
double	Asbx	<i>Bottom reinforcement in x (or ξ) direction [m²/m]</i>
double	Asby	<i>Bottom reinforcement in y (or η) direction [m²/m]</i>
double	Astx	<i>Top reinforcement in x (or ξ) direction [m²/m]</i>
double	Asty	<i>Top reinforcement in y (or η) direction [m²/m]</i>
double	Nx	<i>Axial force in x direction [kN/m]</i>
double	Ny	<i>Axial force in y direction [kN/m]</i>
double	Nxy	<i>Axial force in xy direction [kN/m]</i>
double	Mx	<i>Bending moment in x direction [kNm/m]</i>
double	My	<i>Bending moment in y direction [kNm/m]</i>
double	Mxy	<i>Bending moment in x y direction [kNm/m]</i>
double	wR_p	<i>Angle of primary cracks relative to the local x direction [rad]</i>
double	wR_s	<i>Angle of secondary cracks relative to the local x direction [rad]</i>
double	wk_p	<i>Primary crack width at the axis of the rebar [m]</i>
double	wk_s	<i>Secondary crack width at the axis of the rebar [m]</i>
double	wk2_p	<i>Primary crack width at the surface level [m]</i>
double	wk2_s	<i>Secondary crack width at the surface level [m]</i>
double	xs2_p	<i>Position of neutral axis of primary crack relative to the edge on the compression side [m]</i>

Functions

long **GetCrackWidthsByLoadCaseId** ([in] long **Surfaceld**, [in] long **LoadCaseId**, [in] long **LoadLevel**,
[in] [EAnalysisType](#) **AnalysisType**, [i/o] [RCrackWidths](#)* **CrackWidths**, [out] BSTR* **Combination**)

Surfaceld index of the surface element
LoadCaseId load case index ($0 < LoadCaseId \leq \text{AxisVMLoadCases.Count}$)
LoadLevel load level (increment) index
AnalysisType Type of Analysis
CrackWidths Crack widths of the surface element
Combination String with name of load case.

If successful returns Surfaceld, otherwise an error code ([errDatabaseNotReady](#),
[errIndexOutOfBounds](#), [coeLoadCaseIdIndexOutOfBounds](#), [coeInvalidAnalysisType](#),
[coeInvalidCombinationOfLoadCaseAndLoadLevel](#)).

long **GetCrackWidthsByLoadCombinationId** ([in] long **Surfaceld**, [in] long **LoadCombinationId**,
[in] long **LoadLevel**, [in] [EAnalysisType](#) **AnalysisType**, [i/o] [RCrackWidths](#)* **CrackWidths**,
[out] BSTR* **Combination**)

Surfaceld index of the surface element
LoadCombinationId load combination index ($0 < LoadCombinationId \leq \text{AxisVMLoadCombinations.Count}$)
LoadLevel load level (increment) index
AnalysisType Type of Analysis
CrackWidths Crack widths of the surface element
Combination String with name of combination

If successful returns Surfaceld, otherwise an error code ([errDatabaseNotReady](#),
[errIndexOutOfBounds](#), [coeLoadCombinationIdIndexOutOfBounds](#), [coeInvalidAnalysisType](#),
[coeInvalidCombinationOfLoadCombinationAndLoadLevel](#)).

long **GetEnvelopeCrackWidths** ([in] long **Surfaceld**, [in] [EAnalysisType](#) **AnalysisType**,
[in] [ECrackWidth](#) **Component**, [i/o] [RCrackWidths](#)* **CrackWidths**, [out] BSTR* **Combination**)

Surfaceld index of the surface element
AnalysisType Type of Analysis
Component Location of cracking
CrackWidths Crack widths of the surface element
Combination Combination contain a multiline strings (separated with CR+LF)
where the lines belong to: ContourPoint1, ContourLineMidPoint1,
ContourPoint2, ContourLineMidPoint2, ContourPoint3,
ContourLineMidPoint3, [ContourPoint4, ContourLineMidPoint4],
CenterPoint

Retrieves envelope results. Envelope is identified by Component , EnvelopeUID and MinMaxType properties.
If successful returns Surfaceld, otherwise an error code ([errDatabaseNotReady](#),
[errIndexOutOfBounds](#), [coeInvalidAnalysisType](#)).

long **GetEnvelopeCrackWidths2** ([in] long **Surfaceld**, [in] [EAnalysisType](#) **AnalysisType**,
[in] [ECrackWidth](#) **Component**, [i/o] [RCrackWidths](#)* **CrackWidths**,
[out] long **LoadCaseOrCombinationId**, [out] long **LoadLevel**)

Surfaceld index of the surface element
AnalysisType Type of Analysis
Component Location of cracking
CrackWidths Crack widths of the surface element
LoadCaseOrCombinationId load case or load combination index of surface midpoint's results, if
index is > [IAxisVMLoadcases](#).count then Load combination index =
LoadCaseOrCombinationId - [IAxisVMLoadcases](#).count
LoadLevel load level of surface midpoint's results

Retrieves envelope results. Envelope is identified by Component , EnvelopeUID and MinMaxType properties.
If successful returns Surfaceld, otherwise an error code ([errDatabaseNotReady](#),
[errIndexOutOfBounds](#), [coeInvalidAnalysisType](#)).

long	GetCriticalCrackWidths ([in] long Surfaceld , [in] ECombinationType CombinationType , [in] EAnalysisType AnalysisType , [in] ECrackWidth Component , [i/o] RCrackWidths * CrackWidths , [out] BSTR* Combination)
	<p>Surfaceld <i>index of the surface element</i></p> <p>CombinationType <i>Combination Type</i></p> <p>AnalysisType <i>Type of Analysis</i></p> <p>Component <i>Location of cracking</i></p> <p>CrackWidths <i>Crack widths of the surface element</i></p> <p>Combination <i>Combination contain a multiline strings (separated with CR+LF) where the lines belong to: ContourPoint1, ContourLineMidPoint1, ContourPoint2, ContourLineMidPoint2, ContourPoint3, ContourLineMidPoint3, [ContourPoint4, ContourLineMidPoint4], CenterPoint</i></p>
	<i>If successful returns Surfaceld, otherwise an error code (errDatabaseNotReady, errNotSupportedByNationalDesignCode, errIndexOutOfBounds, coeInvalidAnalysisType).</i>
long	GetCriticalCrackWidths2 ([in] long Surfaceld , [in] ECombinationType CombinationType , [in] EAnalysisType AnalysisType , [in] ECrackWidth Component , [i/o] RCrackWidths * CrackWidths , [out] ECombinationType CriticalCombinationType , [out] SAFEARRAY(double) Factors , [out] SAFEARRAY(long) LoadCaselds)
	<p>Surfaceld <i>index of the surface element</i></p> <p>CombinationType <i>Combination Type</i></p> <p>AnalysisType <i>Type of Analysis</i></p> <p>Component <i>Location of cracking</i></p> <p>CrackWidths <i>Crack widths of the surface element</i></p> <p>CriticalCombinationType <i>combination type corresponding to of midpoint critical result</i></p> <p>Factors <i>load factors of midpoint's critical load combination</i></p> <p>LoadCaselds <i>load case indexes of midpoint's critical load combination</i></p>
	<i>If successful returns Surfaceld, otherwise an error code (errDatabaseNotReady, errNotSupportedByNationalDesignCode, errIndexOutOfBounds, coeInvalidAnalysisType).</i>
long	GetAllCrackWidthsByLoadCaseld ([in] long LoadCaseld , [in] long LoadLevel , [in] EAnalysisType AnalysisType , [out] SAFEARRAY(RCrackWidths)* CrackWidths , [out] SAFEARRAY(BSTR)* Combinations)
	<p>LoadCaseld <i>load case index ($0 < LoadCaseld \leq \text{AxisVMLoadCases.Count}$)</i></p> <p>LoadLevel <i>load level (increment) index</i></p> <p>AnalysisType <i>Type of Analysis</i></p> <p>CrackWidths <i>Crack widths of all surface elements</i></p> <p>Combinations <i>Array of strings with names of load case for all surface elements.</i></p>
	<i>If successful returns number of surface elements, otherwise an error code (errDatabaseNotReady, coeLoadCaseldIndexOutOfBounds, coeInvalidAnalysisType, coeInvalidCombinationOfLoadCaseAndLoadLevel).</i>
long	GetAllCrackWidthsByLoadCombinationId ([in] long LoadCombinationId , [in] long LoadLevel , [in] EAnalysisType AnalysisType , [out] SAFEARRAY(RCrackWidths)* CrackWidths , [out] SAFEARRAY(BSTR)* Combinations)
	<p>LoadCombinationId <i>load combination index ($0 < LoadCombinationId \leq \text{AxisVMLoadCombinations.Count}$)</i></p> <p>LoadLevel <i>load level (increment) index</i></p> <p>AnalysisType <i>Type of Analysis</i></p> <p>CrackWidths <i>Crack widths of all surface elements</i></p> <p>Combinations <i>Array of strings with combination names for all surface elements.</i></p>
	<i>If successful returns number of surface elements, otherwise an error code (errDatabaseNotReady, coeLoadCombinationIdIndexOutOfBounds, coeInvalidAnalysisType, coeInvalidCombinationOfLoadCombinationAndLoadLevel).</i>

long	GetAllEnvelopeCrackWidths ([in] EAnalysisType AnalysisType, [in] ECrackWidth Component, [out] SAFEARRAY(RCrackWidths)* CrackWidths, [out] SAFEARRAY(BSTR)* Combinations)
	<p>AnalysisType Type of Analysis</p> <p>Component Location of cracking</p> <p>CrackWidths Crack widths of all surface elements</p> <p>Combinations String array of multiline strings (separated with CR+LF) where the lines belong to: ContourPoint1, ContourLineMidPoint1, ContourPoint2, ContourLineMidPoint2, ContourPoint3, ContourLineMidPoint3, [ContourPoint4, ContourLineMidPoint4], CenterPoint for all surface elements</p>
	If successful returns number of surface elements, otherwise an error code (errDatabaseNotReady , errIndexOutOfBounds , coeInvalidAnalysisType).
long	GetAllCriticalCrackWidths ([in] ECombinationType CombinationType, [in] EAnalysisType AnalysisType, [in] ECrackWidth Component, [out] SAFEARRAY(RCrackWidths)* CrackWidths, [out] SAFEARRAY(BSTR)* Combinations)
	<p>CombinationType Combination Type</p> <p>AnalysisType Type of Analysis</p> <p>Component Location of cracking</p> <p>CrackWidths Crack widths of all surface elements</p> <p>Combinations String array of multiline strings (separated with CR+LF) where the lines belong to: ContourPoint1, ContourLineMidPoint1, ContourPoint2, ContourLineMidPoint2, ContourPoint3, ContourLineMidPoint3, [ContourPoint4, ContourLineMidPoint4], CenterPoint for all surface elements</p>
	If successful returns number of surface elements, otherwise an error code (errDatabaseNotReady , errNotSupportedByNationalDesignCode , errIndexOutOfBounds , coeInvalidAnalysisType).
long	CrackWidthsByLoadCaseId ([in] long SurfaceId, [i/o] RCrackWidths * CrackWidths, [out] BSTR* Combination)
	<p>SurfaceId index of the surface element</p> <p>CrackWidths Crack widths of the surface element</p> <p>Combination String with name of load case.</p>
	If successful returns SurfaceId, otherwise an error code (errDatabaseNotReady , errIndexOutOfBounds , coeLoadCaseIdIndexOutOfBounds , coeInvalidAnalysisType , coeInvalidCombinationOfLoadCaseAndLoadLevel).
long	CrackWidthsByLoadCombinationId ([in] long SurfaceId, [i/o] RCrackWidths * CrackWidths, [out] BSTR* Combination)
	<p>SurfaceId index of the surface element</p> <p>CrackWidths Crack widths of the surface element</p> <p>Combination String with name of combination</p>
	If successful returns SurfaceId, otherwise an error code (errDatabaseNotReady , errIndexOutOfBounds , coeLoadCombinationIdIndexOutOfBounds , coeInvalidAnalysisType , coeInvalidCombinationOfLoadCombinationAndLoadLevel).
long	EnvelopeCrackWidths ([in] long SurfaceId, [i/o] RCrackWidths * CrackWidths, [out] BSTR* Combination)
	<p>SurfaceId index of the surface element</p> <p>CrackWidths Crack widths of the surface element</p> <p>Combination Combination contain a multiline strings (separated with CR+LF) where the lines belong to: ContourPoint1, ContourLineMidPoint1, ContourPoint2, ContourLineMidPoint2, ContourPoint3, ContourLineMidPoint3, [ContourPoint4, ContourLineMidPoint4], CenterPoint</p>
	Retrieves envelope results. Envelope is identified by Component, EnvelopeUID and MinMaxType properties. If successful returns SurfaceId, otherwise an error code (errDatabaseNotReady , errIndexOutOfBounds , coeInvalidAnalysisType).

long	EnvelopeCrackWidths2 ([in] long Surfaceld , [i/o] RCrackWidths * CrackWidths , [out] long LoadCaseOrCombinationId , [out] long LoadLevel)
	<p>Surfaceld index of the surface element</p> <p>CrackWidths Crack widths of the surface element</p> <p>LoadCaseOrCombinationId load case or load combination index of surface midpoint's results, if index is > IAxisVMLoadcases.count then Load combination index = LoadCaseOrCombinationId - IAxisVMLoadcases.count</p> <p>LoadLevel load level of surface midpoint's results</p> <p>Retrieves envelope results. Envelope is identified by Component, EnvelopeUID and MinMaxType properties. If successful returns Surfaceld, otherwise an error code (errDatabaseNotReady, errIndexOutOfBounds, coeInvalidAnalysisType).</p>
long	CriticalCrackWidths ([in] long Surfaceld , [i/o] RCrackWidths * CrackWidths , [out] BSTR* Combination)
	<p>Surfaceld index of the surface element</p> <p>CrackWidths Crack widths of the surface element</p> <p>Combination Combination contain a multiline strings (separated with CR+LF) where the lines belong to: ContourPoint1, ContourLineMidPoint1, ContourPoint2, ContourLineMidPoint2, ContourPoint3, ContourLineMidPoint3, [ContourPoint4, ContourLineMidPoint4], CenterPoint</p> <p>If successful returns Surfaceld, otherwise an error code (errDatabaseNotReady, errNotSupportedByNationalDesignCode, errIndexOutOfBounds, coeInvalidAnalysisType).</p>
long	CriticalCrackWidths2 ([in] long Surfaceld , [i/o] RCrackWidths * CrackWidths , [out] ECombinationType CriticalCombinationType , [out] SAFEARRAY(double) Factors , [out] SAFEARRAY(long) LoadCaselds)
	<p>Surfaceld index of the surface element</p> <p>CrackWidths Crack widths of the surface element</p> <p>CriticalCombinationType combination type corresponding to of surface midpoint's critical result</p> <p>Factors load factors of midpoint's critical load combination</p> <p>LoadCaselds load case indexes of midpoint's critical load combination</p> <p>If successful returns Surfaceld, otherwise an error code (errDatabaseNotReady, errNotSupportedByNationalDesignCode, errIndexOutOfBounds, coeInvalidAnalysisType).</p>
long	AllCrackWidthsByLoadCaseld ([out] SAFEARRAY(RCrackWidths)* CrackWidths , [out] SAFEARRAY(BSTR)* Combinations)
	<p>CrackWidths Crack widths of all surface elements</p> <p>Combinations Array of strings with names of load case for all surface elements.</p> <p>If successful returns number of surface elements, otherwise an error code (errDatabaseNotReady, coeLoadCaseldIndexOutOfBounds, coeInvalidAnalysisType, coeInvalidCombinationOfLoadCaseAndLoadLevel).</p>
long	AllCrackWidthsByLoadCombinationId ([out] SAFEARRAY(RCrackWidths)* CrackWidths , [out] SAFEARRAY(BSTR)* Combinations)
	<p>CrackWidths Crack widths of all surface elements</p> <p>Combinations Array of strings with combination names for all surface elements.</p> <p>If successful returns number of surface elements, otherwise an error code (errDatabaseNotReady, coeLoadCombinationIdIndexOutOfBounds, coeInvalidAnalysisType, coeInvalidCombinationOfLoadCombinationAndLoadLevel).</p>

long	AllEnvelopeCrackWidths ([out] SAFEARRAY(RCrackWidths)* CrackWidths , [out] SAFEARRAY(BSTR)* Combinations)
	<p>CrackWidths Crack widths of all surface elements</p> <p>Combinations String array of multiline strings (separated with CR+LF) where the lines belong to: ContourPoint1, ContourLineMidPoint1, ContourPoint2, ContourLineMidPoint2, ContourPoint3, ContourLineMidPoint3, [ContourPoint4, ContourLineMidPoint4], CenterPoint for all surface elements</p>
Retrieves envelope results. Envelope is identified by Component , EnvelopeUID and MinMaxType properties. If successful returns number of surface elements, otherwise an error code (errDatabaseNotReady , errIndexOutOfBounds , coeInvalidAnalysisType).	
long	AllCriticalCrackWidths ([out] SAFEARRAY(RCrackWidths)* CrackWidths , [out] SAFEARRAY(BSTR)* Combinations)
	<p>CrackWidths Crack widths of all surface elements</p> <p>Combinations String array of multiline strings (separated with CR+LF) where the lines belong to: ContourPoint1, ContourLineMidPoint1, ContourPoint2, ContourLineMidPoint2, ContourPoint3, ContourLineMidPoint3, [ContourPoint4, ContourLineMidPoint4], CenterPoint for all surface elements</p>
If successful returns number of surface elements, otherwise an error code (errDatabaseNotReady , errNotSupportedByNationalDesignCode , errIndexOutOfBounds , coeInvalidAnalysisType).	
long	SetUserCreep ([in] EBoolean Creep)
	<p>Creep If lbTrue concrete creep will be considered in results of nonlinear analysis if national design code allows it. More here...</p>
Enable or disable consideration of concrete creep in nonlinear analysis results.. If successful returns 1, otherwise an error code (errDatabaseNotReady , errCreepNotSupported).	

Properties

EAnalysisType	AnalysisType •
	Get or set type of analysis
ECrackWidth	Component •
	Get or set location of crack
long	Count
	Get number of surface elements in the model, if 0 then results not calculated or invalid.
ECombinationType	CombinationType •
	Get or set the type of combination for critical results
long	EnvelopeUID •
	Get or set the unique index of the envelope used in functions for reading envelope results
long	LoadCaseId •
	Get or set load case index (0 < LoadCaseId ≤ AxisVMLoadCases .Count)
long	LoadCombinationId •
	Get or set load combination index (0 < LoadCombinationId ≤ AxisVMLoadCombinations .Count)
long	LoadLevel •
	Get or set load level (increment) index
EBoolean	UserCreep
	Returns lbTrue if nonlinear analysis results consider concrete creep. More here...

IAxisVMDisplacements

Interface containing nodal/line displacement results of the model and vibration or buckling shapes.

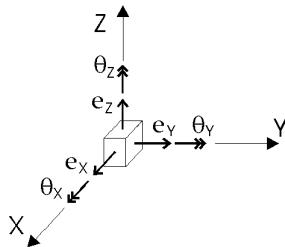
For further details of result objects see [Overview of AxisVM result objects](#).

If the model database is not available when calling functions or reading properties an [errDatabaseNotReady](#) error code is returned.

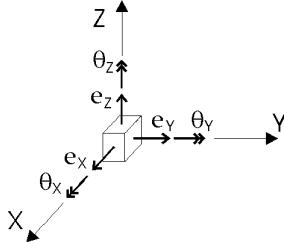
Error codes

enum EDisplacementError = {	
deNodeIndexOutOfBounds = -100001	<i>node index is out of bounds</i>
deLoadCaseIndexOutOfBounds = -100002	<i>LoadCaseId is out of bounds</i>
deLoadCombinationIndexOutOfBounds = -100003	<i>LoadCombinationId is out of bounds</i>
deCombinationTypeNotValidForCurrentNationalDesignCode = -100004	<i>used CombinationType is incompatible with the current design code</i>
deCOMError = -100005	<i>internal COM error when calling GetRecordInfoFromGUIDs, SafeArrayCreateEx, SafeArrayAccessData</i>
deNoNodesInTheModel = -100006	<i>AxisVMMODEL.Nodes.Count = 0</i>
deNoLoadCasesInTheModel = -100007	<i>AxisVMMODEL.LoadCases.Count = 0</i>
deNoLoadCombinationsInTheModel = -100008	<i>AxisVMMODEL.LoadCombinations.Count = 0</i>
deSectionIndexOutOfBounds = -100009	<i>cross-section index is out of bounds</i>
deLineIndexOutOfBounds = -100010	<i>line index is out of bounds</i>
deLineHasNoSections = -100011	<i>line element has no sections</i>
deNoValidLinesInTheModel = -100012	<i>no valid lines in the model</i>
deInvalidAnalysisType = -100013	<i>AnalysisType is incompatible with the function</i>
deInvalidCombinationOfLoadCaseAndLoadLevel = -100014	<i>no result is available for the given LoadCaseId and LoadLevelOrModeShape or thisCreepOfConcrete</i>
deInvalidCombinationOfLoadCombinationAndLoadLevel = -100015	<i>no result is available for the given LoadCombinationId and LoadLevelOrModeShape or this</i>
deNoResultBlocksInTheModel = -100016	<i>no result blocks in the model</i>
deInvalidLineType = -100017,	<i>Line type is not valid for the selected operation</i>
deMemberIndexOutOfBounds = -100018,	<i>member index is out of bounds</i>
deVirtualBeamIndexOutOfBounds = -100019,	<i>virtual beam or strip index is out of bounds</i>
deVirtualBeamChainIndexOutOfBounds = -100020,	<i>index of virtual beam or strip chain is out of bounds</i>
deVirtualBeamSectionIndexOutOfBounds = -100021 }	<i>index of virtual beam or strip section is out of bounds</i>

Enumerated types

enum EDisplacement = {	nodal displacements
d_eX = 0x01	<i>displacement in global X direction</i>
d_eY = 0x02	<i>displacement in global Y direction</i>
d_eZ = 0x03	<i>displacement in global Z direction</i>
d_fX = 0x04	<i>rotation about the global X axis</i>
d_fY = 0x05	<i>rotation about the global Y axis</i>
d_fZ = 0x06	<i>rotation about the global Z axis</i>
d_eR = 0x07	<i>global resultant displacement</i>
d_fR = 0x08 }	<i>global resultant rotation</i>
<i>Displacement component identifiers [m]</i>	
	
line and member displacements	
<i>displacement in global or local x direction (depends on DisplacementSystem property)</i>	
<i>displacement in global or local y direction (depends on DisplacementSystem property)</i>	
<i>displacement in global or local z direction (depends on DisplacementSystem property)</i>	
<i>rotation about the global or the local x axis (depends on DisplacementSystem property)</i>	
<i>rotation about the global or the local y axis (depends on DisplacementSystem property)</i>	
<i>rotation about the global or the local z axis (depends on DisplacementSystem property)</i>	
<i>global or local resultant displacement (depends on DisplacementSystem property)</i>	
<i>global or local resultant rotation (depends on DisplacementSystem property)</i>	

Displacement component identifiers [m]



enum EDisplacementSystem = {	
dsLocal = 0x01	<i>line or member displacements in local system</i>
dsGlobal }	<i>line or member displacements in global system</i>

Records / structures

RDisplacementValues = (

		nodal displacements	line and member displacements
double	Ex	<i>displacement in global X direction</i>	<i>displacement in global or local x direction (depends on DisplacementSystem property)</i>
double	Ey	<i>displacement in global Y direction</i>	<i>displacement in global or local y direction (depends on DisplacementSystem property)</i>
double	Ez	<i>displacement in global Z direction</i>	<i>displacement in global or local z direction (depends on DisplacementSystem property)</i>
double	Fx	<i>rotation about the global X axis</i>	<i>rotation about the global or the local x axis (depends on DisplacementSystem property)</i>
double	Fy	<i>rotation about the global Y axis</i>	<i>rotation about the global or the local y axis (depends on DisplacementSystem property)</i>
double	Fz	<i>rotation about the global Z axis</i>	<i>rotation about the global or the local z axis (depends on DisplacementSystem property)</i>
double	eR	<i>global resultant displacement</i>	<i>global or local resultant displacement (depends on DisplacementSystem property)</i>
double	fR	<i>global resultant rotation</i>	<i>global or local resultant rotation (depends on DisplacementSystem property)</i>

Important Note:

Mode shapes can be read only by using nodal displacement functions. Use **NodalDisplacementByLoadCaseId** function if mode shapes are analysed for load case and **NodalDisplacementByLoadCombinationId** function if mode shapes are analysed for load combination.

Nodal Displacements

Single LOCATION reader functions

long **NodalDisplacementByLoadCaseId** ([in] long **NodeId**,
[i/o] [RDisplacementValues](#) **Displacement**, [out] BSTR **Combination**)

NodeId *node index or line midpoint index
(0 < NodeId ≤ AxisVMMModel.Nodes.Count) or a value returned by AxisVMMModel.Lines.MidpointId[LineIndex]*

Displacement *displacement results*
Combination *name of the load case*

Retrieves displacements of a node or line midpoint according to the LoadCaseId (and LoadLevelOrModeShapeOrTimeStep) property. Returns NodeId or an error code ([errDatabaseNotReady](#) or see [EDisplacementError](#)).

long **NodalDisplacementByLoadCombinationId** ([in] long **NodeId**,
[i/o] [RDisplacementValues](#) **Displacement**, [out] BSTR **Combination**)

NodeId *node index or line midpoint index
(0 < NodeId ≤ AxisVMMModel.Nodes.Count) or a value returned by AxisVMMModel.Lines.MidpointId[LineIndex]*

Displacement *displacement results*
Combination *name of the load combination*

*Retrieves displacements of a node or line midpoint according to the LoadCombinationId (and LoadLevelOrModeShapeOrTimeStep) property.
Returns NodeId or an error code ([errDatabaseNotReady](#) or see [EDisplacementError](#)).*

long **EnvelopeNodalDisplacement** ([in] long **NodeId**, [i/o] [RDisplacementValues](#) **Displacement**,
[out] BSTR **Combination**)

NodeId *node index or line midpoint index (0 < NodeId ≤ AxisVMMModel.Nodes.Count) or a value returned by AxisVMMModel.Lines.MidpointId[LineIndex]*

Displacement *displacement results*
Combination *name of the load case or combination in which Component has its minimum or maximum*

Retrieves envelope results. Envelope is identified by Component, EnvelopeUID and MinMaxType properties.

Returns NodeId or an error code ([errDatabaseNotReady](#) or see [EDisplacementError](#)).

long **EnvelopeNodalDisplacement2** ([in] long **NodId**, [i/o] [RDisplacementValues](#) **Displacement**,
[out] long **LoadCaseOrCombinationId**, [out] long **LoadLevel**)

NodId node index or line midpoint index ($0 < \text{NodId} \leq \text{AxisVMMModel.Nodes.Count}$) or a value returned by
 $\text{AxisVMMModel.Lines.MidpointId[LinelIndex]}$
displacement results

Displacement
LoadCaseOrCombinationId load case or load combination index, if index is $> \text{IAxisVMLoadcases.count}$ then Load combination index =
 $\text{LoadCaseOrCombinationId} - \text{IAxisVMLoadcases.count}$
LoadLevel load level

Retrieves envelope results. Envelope is identified by Component, EnvelopeUID and MinMaxType properties.

Returns NodId or an error code ([errDatabaseNotReady](#) or see [EDisplacementError](#)).

long **CriticalNodalDisplacement** ([in] long **NodId**, [i/o] [RDisplacementValues](#) **Displacement**,
[out] BSTR **Combination**)

NodId node index or line midpoint index ($0 < \text{NodId} \leq \text{AxisVMMModel.Nodes.Count}$) or a value returned by
 $\text{AxisVMMModel.Lines.MidpointId[LinelIndex]}$
displacement results

Displacement
Combination critical combination in which Component has its minimum or maximum

Retrieves critical displacements of a node or line midpoint. Critical combination is identified by Component and MinMaxType properties (e.g. Ez min). Returns NodId or an error code ([errDatabaseNotReady](#) or see [EDisplacementError](#)).

long **CriticalNodalDisplacement2** ([in] long **NodId**, [i/o] [RDisplacementValues](#) **Displacement**,
[out] [ECombinationType](#) **CriticalCombinationType**, [out] SAFEARRAY(double) **Factors**,
[out] SAFEARRAY(long) **LoadCaselds**)

CriticalCombinationType combination type corresponding to critical load combination

Factors load factors of the critical load combination

LoadCaselds load case indexes of the critical load combination

Similar to CriticalNodalDisplacement with more parameters described above.

Multiple element reader functions

long **AllNodalDisplacementsByLoadCaseld** ([i/o] SAFEARRAY([RDisplacementValues](#))
Displacements)

Displacements displacement results for all nodes
Length of the array = $\text{AxisVMMModel.Nodes.Count} + \text{AxisVMMModel.Lines.MidpointCount}$

Retrieves displacements of all nodes and line midpoints according to the LoadCaseld (and LoadLevelOrModeShapeOrTimeStep) property. Returns the total number of displacement records in the array or an error code ([errDatabaseNotReady](#) or see [EDisplacementError](#)).

long **AllNodalDisplacementsByLoadCombinationId** ([i/o] SAFEARRAY([RDisplacementValues](#))
Displacements)

Displacements displacement results for all nodes
Length of the array = $\text{AxisVMMModel.Nodes.Count} + \text{AxisVMMModel.Lines.MidpointCount}$

Retrieves displacements of all nodes and line midpoints according to the LoadCombinationId (and LoadLevelOrModeShapeOrTimeStep) property. Returns the total number of displacement records in the array or an error code ([errDatabaseNotReady](#) or see [EDisplacementError](#)).

long **AllEnvelopeNodalDisplacements** ([i/o] SAFEARRAY([RDisplacementValues](#)) **Displacements**)

Displacements array of envelope nodal displacements for all nodes and all midpoints consecutively.

Length of the array is AxisVMMModel.Nodes.Count+AxisVMMModel.Lines.MidpointCount

Retrieves envelope results of all nodes and line midpoints. Envelope is identified by Component, EnvelopeUID and MinMaxType properties. Returns the total number of displacement records in the array or an error code ([errDatabaseNotReady](#) or see [EDisplacementError](#)).

long **AllCriticalNodalDisplacements** ([i/o] SAFEARRAY([RDisplacementValues](#)) **Displacements**)

Displacements array of critical nodal displacements for all nodes and all midpoints consecutively.

Length of the array is AxisVMMModel.Nodes.Count+AxisVMMModel.Lines.MidpointCount

Retrieves critical displacements of all nodes and line midpoints. Critical combination is identified by Component and MinMaxType properties (e.g. Ez min). Returns the total number of displacement records in the array or an error code ([errDatabaseNotReady](#) or see [EDisplacementError](#)).

Multiple BLOCK reader functions

long **NodalDisplacementsForResultBlocks** ([in] long **NodeID**,

[i/o] SAFEARRAY([RResultBlockInfo](#)) **ResultBlockInfo**,

[i/o] SAFEARRAY([RDisplacementValues](#)) **Displacements**)

ResultBlockInfo array of description records for result blocks
Displacements displacement results for all result blocks

Retrieves displacements of the given node or line midpoint in all result blocks consecutively. The number of result blocks depends on the AnalysisType property.

Returns the common length of ResultBlockInfo and Displacements arrays or an error code ([errDatabaseNotReady](#) or see [EDisplacementError](#)).

Line Displacements

Line displacement results are available only for beams and ribs.

If **Lineld** refers to a line of other type the number of cross-sections will be zero.

If a single reader functions is called [deSectionIndexOutOfBounds](#) error code is returned. If a line reader function is called a [deLineHasNoSections](#) error code is returned. If a multiple reader is called the SectionCounts array will contain zero at these line indexes.

Single LOCATION reader functions

long **LineDisplacementByLoadCaseld** ([in] long **Lineld**, [in] long **SectionId**,

[i/o] [RDisplacementValues](#) **Displacement**, [out] double **PosX**, [out] BSTR **Combination**)

Lineld line index ($0 < Lineld \leq AxisVMMModel.Lines.Count$)

SectionId section index ($0 < SectionId \leq AxisVMMModel.Lines[Lineld].SectionCount$)

Displacement displacement results

PosX position of SectionId in m according to the local x direction

Combination name of the load case

Retrieves displacements at a section of a line element according to the LoadCaseld (and LoadLevelOrModeShapeOrTimeStep) property. Returns Lineld or an error code ([errDatabaseNotReady](#) or see [EDisplacementError](#)).

long	LineDisplacementByLoadCombinationId ([in] long Lineld , [in] long SectionId , [i/o] RDisplacementValues Displacement , [out] double PosX , [out] BSTR Combination)
	<p style="margin-left: 20px;">Lineld <i>line index ($0 < \text{Lineld} \leq \text{AxisVMModel.Lines.Count}$)</i></p> <p style="margin-left: 20px;">SectionId <i>section index ($0 < \text{SectionId} \leq \text{AxisVMModel.Lines[Lineld].SectionCount}$)</i></p> <p style="margin-left: 20px;">Displacement <i>displacement results</i></p> <p style="margin-left: 20px;">PosX <i>position of SectionId in m according to the local x direction</i></p> <p style="margin-left: 20px;">Combination <i>name of the load combination</i></p>
	<p><i>Retrieves displacements at a section of a line element according to the LoadCombinationId (and LoadLevelOrModeShapeOrTimeStep) property. Returns Lineld or an error code (errDatabaseNotReady or see EDisplacementError).</i></p>
long	EnvelopeLineDisplacement ([in] long Lineld , [in] long SectionId , [i/o] RDisplacementValues Displacement , [out] double PosX , [out] BSTR Combination)
	<p style="margin-left: 20px;">Lineld <i>line index ($0 < \text{Lineld} \leq \text{AxisVMModel.Lines.Count}$)</i></p> <p style="margin-left: 20px;">SectionId <i>section index ($0 < \text{SectionId} \leq \text{AxisVMModel.Lines[Lineld].SectionCount}$)</i></p> <p style="margin-left: 20px;">Displacement <i>displacement results</i></p> <p style="margin-left: 20px;">PosX <i>position of SectionId in m according to the local x direction</i></p> <p style="margin-left: 20px;">Combination <i>load case or combination in which Component has its minimum or maximum</i></p>
	<p><i>Read envelope results at a section of a line element. Envelope is identified by Component, EnvelopeUID and MinMaxType properties. Displacement contains the result of the load case or combination in which Component is maximal or minimal. Returns Lineld or an error code (errDatabaseNotReady or see EDisplacementError).</i></p>
long	EnvelopeLineDisplacement2 ([in] long Lineld , [in] long SectionId , [i/o] RDisplacementValues Displacement , [out] double PosX , [out] long LoadCaseOrCombinationId , [out] long LoadLevel)
	<p style="margin-left: 20px;">Lineld <i>line index ($0 < \text{Lineld} \leq \text{AxisVMModel.Lines.Count}$)</i></p> <p style="margin-left: 20px;">SectionId <i>section index ($0 < \text{SectionId} \leq \text{AxisVMModel.Lines[Lineld].SectionCount}$)</i></p> <p style="margin-left: 20px;">Displacement <i>displacement results</i></p> <p style="margin-left: 20px;">PosX <i>position of SectionId in m according to the local x direction</i></p> <p style="margin-left: 20px;">LoadCaseOrCombinationId <i>load case or load combination index, if index is > IAxisVMLoadcases.count then Load combination index = LoadCaseOrCombinationId - IAxisVMLoadcases.count</i></p> <p style="margin-left: 20px;">LoadLevel <i>load level</i></p>
	<p><i>Retrieves envelope results. Envelope is identified by Component, EnvelopeUID and MinMaxType properties. Displacement contains the result of the load case or combination in which Component is maximal or minimal. Returns Lineld or an error code (errDatabaseNotReady or see EDisplacementError).</i></p>
long	CriticalLineDisplacement ([in] long Lineld , [in] long SectionId , [i/o] RDisplacementValues Displacement , [out] double PosX , [out] BSTR Combination)
	<p style="margin-left: 20px;">Lineld <i>line index ($0 < \text{Lineld} \leq \text{AxisVMModel.Lines.Count}$)</i></p> <p style="margin-left: 20px;">SectionId <i>section index ($0 < \text{SectionId} \leq \text{AxisVMModel.Lines[Lineld].SectionCount}$)</i></p> <p style="margin-left: 20px;">Displacement <i>displacement results</i></p> <p style="margin-left: 20px;">PosX <i>position of SectionId in m according to the local x direction</i></p> <p style="margin-left: 20px;">Combination <i>critical combination in which Component has its minimum or maximum</i></p>
	<p><i>Retrieves critical displacements at a section of a line element. Critical combination is identified by Component and MinMaxType properties (e.g. Ez min). Displacement contains the result of the critical combination in which Component is maximal or minimal. Returns Lineld or an error code (errDatabaseNotReady or see EDisplacementError).</i></p>

long **CriticalLineDisplacement2** ([in] long **Lineld**, [in] long **SectionId**,
[i/o] [RDisplacementValues](#) **Displacement**, [out] double **PosX**,
[out] [ECombinationType](#) **CriticalCombinationType**, [out] SAFEARRAY(double) **Factors**,
[out] SAFEARRAY(long) **LoadCaselds**)

CriticalCombinationType combination type corresponding to critical load combination

Factors load factors of the critical load combination

LoadCaselds load case indexes of the critical load combination

Similar to *CriticalLineDisplacement* with more parameters described above.

Single ELEMENT reader functions

long **LineDisplacementsByLoadCaseld** ([in] long **Lineld**,
[i/o] SAFEARRAY([RDisplacementValues](#)) **Displacements**, [out] SAFEARRAY(double) **PosX**)

Lineld line index ($0 < \text{Lineld} \leq \text{AxisVMMModel.Lines.Count}$)

Displacements array of line displacements for the line element.

Length of the array is *SectionCount[Lineld]*

PosX array of cross-section positions in *m* according to the local *x* direction
Length of the array is *SectionCount[Lineld]*

Retrieves displacements for each cross-section of a given element according to the *LoadCaseld* (and *LoadLevelOrModeShapeOrTimeStep*) property. Returns the total number of cross-sections or an error code ([errDatabaseNotReady](#) or see [EDisplacementError](#)).

long **LineDisplacementsByLoadCombinationId** ([in] long **Lineld**,
[i/o] SAFEARRAY([RDisplacementValues](#)) **Displacements**, [out] SAFEARRAY(double) **PosX**)

Lineld line index ($0 < \text{Lineld} \leq \text{AxisVMMModel.Lines.Count}$)

Displacements array of line displacements for the line element.

Length of the array is *SectionCount[Lineld]*

PosX array of cross-section positions in *m* according to the local *x* direction
Length of the array is *SectionCount[Lineld]*

Retrieves displacements for each cross-section of a given element according to the *LoadCombinationId* (and *LoadLevelOrModeShapeOrTimeStep*) property. Returns the total number of cross-sections or an error code ([errDatabaseNotReady](#) or see [EDisplacementError](#)).

long **EnvelopeLineDisplacements** ([in] long **Lineld**,
[i/o] SAFEARRAY([RDisplacementValues](#)) **Displacements**, [out] SAFEARRAY(double) **PosX**)

Lineld line index ($0 < \text{Lineld} \leq \text{AxisVMMModel.Lines.Count}$)

Displacements array of envelope line displacements for the line element.

Length of the array is *SectionCount[Lineld]*

PosX array of cross-section positions in *m* according to the local *x* direction
Length of the array is *SectionCount[Lineld]*

Retrieves envelope results. Envelope is identified by *Component*, *EnvelopeUID* and *MinMaxType* properties. *Displacements* array contains the result of the load case or combination in which *Component* is maximal or minimal. Load case or combination in which *Component* has its minimum or maximum can be read using the respective single element/single cross-section reader function.

Returns the total number of cross-sections or an error code ([errDatabaseNotReady](#) or see [EDisplacementError](#)).

long **CriticalLineDisplacements** ([in] long **LinElId**,
[i/o] SAFEARRAY([RDisplacementValues](#)) **Displacements**, [out] SAFEARRAY(double) **PosX**)

LinElId *line index ($0 < LinElId \leq AxisVMMModel.Lines.Count$)*
Displacements *array of critical line displacements for the line element.*
Length of the array is SectionCount[LinElId]
PosX *array of cross-section positions in m according to the local x direction*
Length of the array is SectionCount[LinElId]

Retrieves critical displacements in each cross-section of a given element. Critical combination is identified by Component and MinMaxType properties (e.g. Ez min). Displacements array contains the result of the critical combination in which Component is maximal or minimal. Critical combination in which Component has its minimum or maximum can be read using the respective single element/single cross-section reader function.

Returns the total number of cross-sections or an error code ([errDatabaseNotReady](#) or see [EDisplacementError](#)).

Multiple element reader functions

long **AllLineDisplacementsByLoadCasElId** ([out] SAFEARRAY(long) **SectionCounts**,
[i/o] SAFEARRAY([RDisplacementValues](#)) **Displacements**, [out] SAFEARRAY(double) **PosX**)

SectionCounts *array containing the section counts of line elements.*
Length of the array = AxisVMMModel.Lines.Count
Displacements *array of line displacements.*
For each line element it contains results for SectionCount[i] sections consecutively.
Length of the array is the sum of the SectionCounts array elements
PosX *array of cross-section positions in m according to the local x direction of each line element*
Length of the array is the sum of the SectionCounts array elements

Retrieves displacements of all lines and cross-sections consecutively according to the LoadCasElId (and LoadLevelOrModeShapeOrTimeStep) property. Returns the total number of displacement records in the array or an error code ([errDatabaseNotReady](#) or see [EDisplacementError](#)).

long **AllLineDisplacementsByLoadCombinationId** ([out] SAFEARRAY(long) **SectionCounts**,
[i/o] SAFEARRAY([RDisplacementValues](#)) **Displacements**, [out] SAFEARRAY(double) **PosX**)

SectionCounts *array containing the section counts of line elements.*
Length of the array = AxisVMMModel.Lines.Count
Displacements *array of line displacements.*
For each line element it contains results for SectionCount[i] sections consecutively.
Length of the array is the sum of the SectionCounts array elements
PosX *array of cross-section positions in m according to the local x direction of each line element*
Length of the array is the sum of the SectionCounts array elements

Retrieves displacements of all lines and cross-sections consecutively according to the LoadCombinationId (and LoadLevelOrModeShapeOrTimeStep) property. Returns the total number of displacement records in the array or an error code ([errDatabaseNotReady](#) or see [EDisplacementError](#)).

long	AllEnvelopeLineDisplacements ([out] SAFEARRAY(long) SectionCounts , [i/o] SAFEARRAY(RDisplacementValues) Displacements , [out] SAFEARRAY(double) PosX)
	<p>SectionCounts array containing the section counts of line elements. Length of the array = AxisVMMModel.Lines.Count</p> <p>Displacements array of envelope line displacements for all line elements. For each line element it contains results for SectionCount[i] sections consecutively. Length of the array is the sum of the SectionCounts array elements</p> <p>PosX array of cross-section positions in m according to the local x direction of each line element Length of the array is the sum of the SectionCounts array elements</p> <p><i>Retrieves envelope results of all line elements. Envelope is identified by Component , EnvelopeUID and MinMaxType properties. Displacements array contains the result of the load case or combination in which Component is maximal or minimal. Load case or combination in which Component has its minimum or maximum can be read using the respective single location reader function.</i></p> <p><i>Returns the total number of cross-sections or an error code (errDatabaseNotReady or see EDisplacementError).</i></p>
long	AllCriticalLineDisplacements ([out] SAFEARRAY(long) SectionCounts , [i/o] SAFEARRAY(RDisplacementValues) Displacements , [out] SAFEARRAY(double) PosX)
	<p>SectionCounts array containing the section counts of line elements. Length of the array = AxisVMMModel.Lines.Count</p> <p>Displacements array of critical line displacements for all line elements. For each line element it contains results for SectionCount[i] sections consecutively. Length of the array is the sum of the SectionCounts array elements</p> <p>PosX array of cross-section positions in m according to the local x direction of each line element Length of the array is the sum of the SectionCounts array elements</p> <p><i>Retrieves critical displacements of all line elements. Critical combination is identified by Component and MinMaxType properties (e.g. Ez min). Displacements array contains the result of the critical combination in which Component is maximal or minimal. Critical combination in which Component has its minimum or maximum can be read using the respective single location reader function.</i></p> <p><i>Returns the total number of cross-sections or an error code (errDatabaseNotReady or see EDisplacementError).</i></p>

Multiple BLOCK reader functions

long	LineDisplacementsForResultBlocks ([in] long Lineld , [in] long SectionId , [i/o] SAFEARRAY(RResultBlockInfo) ResultBlockInfo , [i/o] SAFEARRAY(RDisplacementValues) Displacements , [out] double PosX)
	<p>Lineld line index ($0 < \text{Lineld} \leq \text{AxisVMMModel.Lines.Count}$)</p> <p>SectionId section index ($0 < \text{SectionId} \leq \text{AxisVMMModel.Lines[Lineld].SectionCount}$)</p> <p>ResultBlockInfo array of description records for result blocks</p> <p>Displacements displacement results for all result blocks</p> <p>PosX cross-section position according to the local x direction of each line element</p> <p><i>Retrieves displacements at the given line and cross-section in all result blocks consecutively. The number of result blocks depends on the AnalysisType property.</i></p> <p><i>Returns the common length of ResultBlockInfo, Displacements and PosX arrays or an error code (errDatabaseNotReady or see EDisplacementError).</i></p>

Virtual Beam and Vitrual Strip Displacements

Single ELEMENT reader functions

long **VirtualBeamOrStripDisplacementsByLoadCaseld** ([in] long **Index**, [in] long **ChainId**,
[i/o] SAFEARRAY([RDisplacementValues](#)) **Displacements**, [i/o] SAFEARRAY([RPoint3d](#))* **Pos**)

Index	<i>virtual beam/strip index ($0 < Index \leq \text{AxisVMModel.VirtualBeams.Count}$)</i>
ChainId	<i>virtual beam's/strip's chain index ($0 < ChainId \leq \text{AxisVMModel.VirtualBeams.ChainCount[Index]}$)</i>
Displacements	<i>array of virtual beam/strip displacements Length of the array is AxisVMModel.VirtualBeams.SectionCount[Index, ChainId]</i>
Pos	<i>array of section positions in m according to the global coordinate system Length of the array is AxisVMModel.VirtualBeams.SectionCount[Index, ChainId]</i>

Retrieves displacements for each sections of a given chain of a virtual beam/strip according to the LoadCaseld (and LoadLevelOrTimeStep) property. Returns the total number of sections of the chain or an error code ([errDatabaseNotReady](#) or see [EDisplacementError](#)).

long **VirtualBeamOrStripDisplacementsByLoadCombinationId** ([in] long **Index**, [in] long **ChainId**,
[i/o] SAFEARRAY([RDisplacementValues](#)) **Displacements**, [i/o] SAFEARRAY([RPoint3d](#))* **Pos**)

Index	<i>virtual beam/strip index ($0 < Index \leq \text{AxisVMModel.VirtualBeams.Count}$)</i>
ChainId	<i>virtual beam's/strip's chain index ($0 < ChainId \leq \text{AxisVMModel.VirtualBeams.ChainCount[Index]}$)</i>
Displacements	<i>array of virtual beam/strip displacements Length of the array is AxisVMModel.VirtualBeams.SectionCount[Index, ChainId]</i>
Pos	<i>array of section positions in m according to the global coordinate system Length of the array is AxisVMModel.VirtualBeams.SectionCount[Index, ChainId]</i>

Retrieves displacements for each sections of a given chain of a virtual beam/strip according to the LoadCombinationId (and LoadLevelOrTimeStep) property. Returns the total number of sections of the chain or an error code ([errDatabaseNotReady](#) or see [EDisplacementError](#)).

long **EnvelopeVirtualBeamOrStripDisplacements** ([in] long **Index**, [in] long **ChainId**,
[i/o] SAFEARRAY([RDisplacementValues](#)) **Displacements**, [i/o] SAFEARRAY([RPoint3d](#))* **Pos**,
[out] SAFEARRAY(BSTR)* **Combination**)

Index	<i>virtual beam/strip index ($0 < Index \leq \text{AxisVMModel.VirtualBeams.Count}$)</i>
ChainId	<i>virtual beam's/strip's chain index ($0 < ChainId \leq \text{AxisVMModel.VirtualBeams.ChainCount[Index]}$)</i>
Displacements	<i>array of virtual beam/strip displacements Length of the array is AxisVMModel.VirtualBeams.SectionCount[Index, ChainId]</i>
Pos	<i>array of section positions in m according to the global coordinate system Length of the array is AxisVMModel.VirtualBeams.SectionCount[Index, ChainId]</i>
Combination	<i>load cases or combinations in which Component has its minimum or maximum</i>

Retrieves envelope displacements for each sections of a given chain of a virtual beam/strip. Envelope is identified by Component, MinMaxType and EnvelopeUID properties. Combination array contains the name of load cases or combinations in which Component is maximal or minimal. Returns the total number of sections or an error code ([errDatabaseNotReady](#) or see [EDisplacementError](#)).

long	EnvelopeVirtualBeamOrStripDisplacements2 ([in] long Index , [in] long ChainId , [<i>i/o</i>] SAFEARRAY(RDisplacementValues) Displacements , [<i>i/o</i>] SAFEARRAY(RPoint3d)* Pos , [<i>out</i>] SAFEARRAY(long)* LoadCaseOrCombinationIds , [<i>out</i>] SAFEARRAY(long)* LoadLevels)
	<p style="margin-left: 20px;">Index <i>virtual beam/strip index ($0 < \text{Index} \leq \text{AxisVMModel.VirtualBeams.Count}$)</i></p> <p style="margin-left: 20px;">ChainId <i>virtual beam's/strip's chain index ($0 < \text{ChainId} \leq \text{AxisVMModel.VirtualBeams.ChainCount[Index]}$)</i></p>
	<p style="margin-left: 20px;">Displacements <i>array of virtual beam/strip displacements Length of the array is AxisVMModel.VirtualBeams.SectionCount[Index, ChainId]</i></p>
	<p style="margin-left: 20px;">Pos <i>array of section positions in m according to the global coordinate system Length of the array is AxisVMModel.VirtualBeams.SectionCount[Index, ChainId]</i></p>
	<p style="margin-left: 20px;">LoadCaseOrCombinationIds <i>array of load case or load combination indices, if any is > IAxisVMLoadcases.count then Load combination index = LoadCaseOrCombinationId – IAxisVMLoadcases.count</i></p>
	<p style="margin-left: 20px;">LoadLevels <i>array of load levels according to LoadCaseOrCombinationIds</i></p> <p><i>Retrieves envelope displacements for each sections of a given chain of a virtual beam/strip. Envelope is identified by Component, MinMaxType and EnvelopeUID properties. LoadCaseOrLoadCombinationIds array contains the index of load cases or combinations in which Component is maximal or minimal. Returns the total number of sections or an error code (errDatabaseNotReady or see EDisplacementError).</i></p>
long	CriticalVirtualBeamOrStripDisplacements ([in] long Index , [in] long ChainId , [<i>i/o</i>] SAFEARRAY(RDisplacementValues) Displacements , [<i>i/o</i>] SAFEARRAY(RPoint3d)* Pos , [<i>out</i>] SAFEARRAY(BSTR)* Combination)
	<p style="margin-left: 20px;">Index <i>virtual beam/strip index ($0 < \text{Index} \leq \text{AxisVMModel.VirtualBeams.Count}$)</i></p> <p style="margin-left: 20px;">ChainId <i>virtual beam's/strip's chain index ($0 < \text{ChainId} \leq \text{AxisVMModel.VirtualBeams.ChainCount[Index]}$)</i></p>
	<p style="margin-left: 20px;">Displacements <i>array of virtual beam/strip displacements Length of the array is AxisVMModel.VirtualBeams.SectionCount[Index, ChainId]</i></p>
	<p style="margin-left: 20px;">Pos <i>array of section positions in m according to the global coordinate system Length of the array is AxisVMModel.VirtualBeams.SectionCount[Index, ChainId]</i></p>
	<p style="margin-left: 20px;">Combination <i>load cases or combinations in which Component has its minimum or maximum</i></p> <p><i>Retrieves critical displacements for each sections of a given chain of a virtual beam/strip. Critical combination is identified by Component and MinMaxType properties. Displacements array contains the result of critical combination in which Component is maximal or minimal. Returns the total number of sections or an error code (errDatabaseNotReady or see EDisplacementError).</i></p>
long	CriticalVirtualBeamOrStripDisplacement ([in] long Index , [in] long ChainId , [in] long SectionId , [<i>i/o</i>] RDisplacementValues Displacements , [<i>i/o</i>] RPoint3d Pos , [<i>out</i>] BSTR Combination)
	<p style="margin-left: 20px;">Index <i>virtual beam/strip index ($0 < \text{Index} \leq \text{AxisVMModel.VirtualBeams.Count}$)</i></p> <p style="margin-left: 20px;">ChainId <i>virtual beam's/strip's chain index ($0 < \text{ChainId} \leq \text{AxisVMModel.VirtualBeams.ChainCount[Index]}$)</i></p>
	<p style="margin-left: 20px;">SectionId <i>chain's section index ($0 < \text{SectionId} \leq \text{AxisVMModel.VirtualBeams.SectionCount[Index, ChainId]}$)</i></p>
	<p style="margin-left: 20px;">Displacements <i>virtual beam displacements</i></p>
	<p style="margin-left: 20px;">Pos <i>section position in m according to the global coordinate system</i></p>
	<p style="margin-left: 20px;">Combination <i>critical combination in which Component has its minimum or maximum</i></p> <p><i>Retrieves critical displacements for a section of a given chain of a virtual beam/strip. Critical combination is identified by Component and MinMaxType properties. Returns the section index or an error code (errDatabaseNotReady or see EDisplacementError).</i></p>

long	CriticalVirtualBeamOrStripDisplacement2 ([in] long Index , [in] long ChainId , [in] long SectionId , [i/o] RDisplacementValues Displacements , [i/o] RPoint3d Pos , [out] ECombinationType CriticalCombinationType , [out] SAFEARRAY(double)* Factors , [out] SAFEARRAY(long)* LoadCaselds)
	Index virtual beam/strip index ($0 < \text{Index} \leq \text{AxisVMModel.VirtualBeams.Count}$)
	ChainId virtual beam's/strip's chain index ($0 < \text{ChainId} \leq \text{AxisVMModel.VirtualBeams.ChainCount}[\text{Index}]$)
	SectionId chain's section index ($0 < \text{SectionId} \leq \text{AxisVMModel.VirtualBeams.SectionCount}[\text{Index}, \text{ChainId}]$)
	Displacements virtual beam displacements
	Pos section position in m according to the global coordinate system
	CriticalCombinationType combination type corresponding to critical load combination
	Factors load factors of the critical load combination
	LoadCaselds load case indexes of the critical load combination
	<i>Retrieves critical displacements for a section of a given chain of a virtual beam/strip. Critical combination is identified by Component and MinMaxType properties. Similar to CriticalVirtualBeamDisplacement with more parameters described above. Returns the section index or an error code (errDatabaseNotReady or see EDisplacementError).</i>

Save results to MetaFile functions

long	SaveCriticalVirtualBeamOrStripDisplacementsToMetaFile ([in] BSTR FileName , [in] long ID , [in] ECombinationType CombinationType , [in] EAnalysisType AnalysisType , [in] long Width , [in] long Height , [in] ELongBoolean EnvelopeOnly , [in] ELongBoolean RelativeToLeft , [in] ELongBoolean RelativeToRight , [in] double Position , [in] EWindowColourMode ColourMode)
	FileName <i>name of the file with extension emf</i>
	ID <i>index of the virtual beam or strip</i>
	CombinationType <i>combination type</i>
	AnalysisType <i>analysis type</i>
	Width <i>picture's size in pixel (minimal acceptable value is 640)</i>
	Height <i>picture's size in pixel (minimal acceptable value is 580)</i>
	EnvelopeOnly <i>only Envelope</i>
	RelativeToLeft <i>relative deformations to the left endpoint (see the notes below)</i>
	RelativeToRight <i>relative deformations to the 358ight endpoint (see the notes below)</i>
	Position <i>position of the investigated cross section along the virtual beam or strip</i>
	ColourMode <i>colour mode</i>

It creates a metafile from the virtual beam's or strip's critical displacements. It returns ID or an error code ([EGeneralError](#) or see [EDisplacementError](#)).

long	SaveEnvelopeVirtualBeamOrStripDisplacementsToMetaFile ([in] BSTR FileName , [in] long ID , [in] long EnvelopeUID , [in] EAnalysisType AnalysisType , [in] long Width , [in] long Height , [in] ELongBoolean EnvelopeOnly , [in] ELongBoolean RelativeToLeft , [in] ELongBoolean RelativeToRight , [in] double Position , [in] EWindowColourMode ColourMode)
	FileName <i>name of the file with extension emf</i>
	ID <i>index of the virtual beam or strip</i>
	EnvelopeUID <i>unique envelope index</i>
	AnalysisType <i>analysis type</i>
	Width <i>picture's size in pixel (minimal acceptable value is 640)</i>
	Height <i>picture's size in pixel (minimal acceptable value is 580)</i>
	EnvelopeOnly <i>only Envelope</i>
	RelativeToLeft <i>relative deformations to the left endpoint (see the notes below)</i>
	RelativeToRight <i>relative deformations to the 358ight endpoint (see the notes below)</i>
	Position <i>position of the investigated cross section along the virtual beam or strip</i>
	ColourMode <i>colour mode</i>

It creates a metafile from the virtual beam's or strip's displacements envelope. It returns ID or an error code ([EGeneralError](#) or see [EDisplacementError](#)).

long	SaveVirtualBeamOrStripDisplacementsToMetaFileByLoadCaseID ([in] BSTR FileName , [in] long ID , [in] long LoadCaseID , [in] long LoadLevelOrTimeStep , [in] EAnalysisType AnalysisType , [in] long Width , [in] long Height , [in] ELongBoolean RelativeToLeft , [in] ELongBoolean RelativeToRight , [in] double Position , [in] EWindowColourMode ColourMode)
	FileName <i>name of the file with extension emf</i>
	ID <i>index of the virtual beam or strip</i>
	LoadCaseID <i>load case index</i>
	LoadLevelOrTimeStep <i>for load level ($0 < \text{LoadLevelOrTimeStep} \leq \text{LoadLevelCount}$)</i>
	<i>for timestep ($0 < \text{LoadLevelOrTimeStep} \leq \text{TimeStepCount}$)</i>
	AnalysisType <i>analysis type</i>
	Width <i>picture's size in pixel (minimal acceptable value is 640)</i>
	Height <i>picture's size in pixel (minimal acceptable value is 580)</i>
	RelativeToLeft <i>relative deformations to the left endpoint (see the notes below)</i>
	RelativeToRight <i>relative deformations to the 358ight endpoint (see the notes below)</i>
	Position <i>position of the investigated cross section along the virtual beam or strip</i>
	ColourMode <i>colour mode</i>

It saves the virtual beam's or strip's displacements by LoadCaseID into a metafile. It returns ID or an error code ([EGeneralError](#) or see [EDisplacementError](#)).

long **SaveVirtualBeamOrStripDisplacementsToMetaFileByLoadCombinationID** ([in] BSTR
 FileName, [in] long **ID**, [in] long **LoadCombinationId**, [in] long **LoadLevelOrTimeStep**, [in]
 EAnalysisType **AnalysisType**, [in] long **Width**, [in] long **Height**, [in] **EBoolean**
 RelativeToLeft, [in] **EBoolean** **RelativeToRight**, [in] double **Position**, [in]
 EWindowColourMode **ColourMode**)

FileName *name of the file with extension emf*
 ID *index of the virtual beam or strip*
 LoadCombinationId *load combination index*
 LoadLevelOrTimeStep *for load level ($0 < LoadLevelOrTimeStep \leq LoadLevelCount$)*
 for timestep ($0 < LoadLevelOrTimeStep \leq TimeStepCount$)
 AnalysisType *analysis type*
 Width *picture's size in pixel (minimal acceptable value is 640)*
 Height *picture's size in pixel (minimal acceptable value is 580)*
 RelativeToLeft *relative deformations to the left endpoint (see the [notes](#) below)*
 RelativeToRight *relative deformations to the right endpoint (see the [notes](#) below)*
 Position *position of the investigated cross section along the virtual beam or strip*
 ColourMode *colour mode*

It saves the virtual beam's or strip's displacements by LoadCombinationId into a metafile. It returns ID or an error code ([EGeneralError](#) or see [EDisplacementError](#)).

End release deformations

Single LOCATION reader functions

long	EndReleasesDeformationsByLoadCaselD ([in] long LinelD, [i/o] RDIsplacementValues StartDisplacement, [i/o] RDIsplacementValues EndDisplacement, [out] BSTR StartCombination, [out] BSTR EndCombination)
	LinelD line index ($0 < \text{LinelD} \leq \text{AxisVMMModel.Lines.Count}$) StartDisplacement deformations of the release at beginning of the line EndDisplacement deformations of the release at the end of the line StartCombination name of the load case for deformations of the release at the end of the line EndCombination name of the load case for deformations of the release at the begining of the line
	<i>Retrieves deformations of end releases of a line element according to the LoadCaselD (and LoadLevelOrModeShapeOrTimeStep) property. Returns LinelD or an error code (errDatabaseNotReady , deInvalidCombinationOfLoadCaseAndLoadLevel , deInvalidLineType , deLoadCaseIndexOutOfBounds , deLineIndexOutOfBounds or deInvalidAnalysisType).</i>
long	EndReleasesDeformationsByByLoadCombinationId ([in] long LinelD, [i/o] RDIsplacementValues StartDisplacement, [i/o] RDIsplacementValues EndDisplacement, [out] BSTR StartCombination, [out] BSTR EndCombination)
	LinelD line index ($0 < \text{LinelD} \leq \text{AxisVMMModel.Lines.Count}$) StartDisplacement deformations of the release at beginning of the line EndDisplacement deformations of the release at the end of the line StartCombination name of the load case for deformations of the release at the end of the line EndCombination name of the load case for deformations of the release at the begining of the line
	<i>Retrieves deformations of end releases according to the LoadCombinationId (and LoadLevelOrModeShapeOrTimeStep) property. Returns LinelD or an error code (errDatabaseNotReady , deInvalidCombinationOfLoadCaseAndLoadLevel , deInvalidLineType , deLoadCombinationIndexOutOfBounds , deLineIndexOutOfBounds or deInvalidAnalysisType).</i>
long	EnvelopeEndReleasesDeformations ([in] long LinelD, [i/o] RDIsplacementValues StartDisplacement, [i/o] RDIsplacementValues EndDisplacement, [out] BSTR StartCombination, [out] BSTR EndCombination)
	LinelD line index ($0 < \text{LinelD} \leq \text{AxisVMMModel.Lines.Count}$) StartDisplacement deformations of the release at beginning of the line EndDisplacement deformations of the release at the end of the line StartCombination name of the load case or combination for deformations of the release at the end of the line EndCombination name of the load case or combination for deformations of the release at the begining of the line
	<i>Retrieves envelope deformations of end releases. Envelope is identified by Component, MinMaxType and EnvelopeUID properties . Returns LinelD or an error code (errDatabaseNotReady , deInvalidLineType , deLineIndexOutOfBounds or deInvalidAnalysisType).</i>
long	CriticalEndReleasesDeformations ([in] long LinelD, [i/o] RDIsplacementValues StartDisplacement, [i/o] RDIsplacementValues EndDisplacement, [out] BSTR StartCombination, [out] BSTR EndCombination)
	LinelD line index ($0 < \text{LinelD} \leq \text{AxisVMMModel.Lines.Count}$) StartDisplacement deformations of the release at beginning of the line EndDisplacement deformations of the release at the end of the line StartCombination name of the load case or combination for deformations of the release at the end of the line EndCombination name of the load case or combination for deformations of the release at the begining of the line
	<i>Retrieves critical deformations of end releases. Critical combination is identified by Component and MinMaxType properties (e.g. Ez min). Returns LinelD or an error code (errDatabaseNotReady , deCombinationTypeNotValidForCurrentNationalDesignCode , deInvalidLineType , deLineIndexOutOfBounds or deInvalidAnalysisType).</i>

Multiple element reader functions

long	AllEndReleasesDeformationsByLoadCaseId ([i/o] SAFEARRAY(RDisplacementValues) StartDisplacements , [i/o] SAFEARRAY(RDisplacementValues) EndDisplacements) StartDisplacements displacement results for all end releases at the beginning of the lines (Length of the array = 2 * AxisVMMModel.Lines.Count) EndDisplacements displacement results for all end releases at the end of the lines (Length of the array = 2 * AxisVMMModel.Lines.Count) Retrieves all deformations of end releases according to the LoadCaseId (and LoadLevelOrModeShapeOrTimeStep) property. Returns length of each array or an error code (errDatabaseNotReady , deInvalidCombinationOfLoadCaseAndLoadLevel , deLoadCaseIndexOutOfBounds or deInvalidAnalysisType).
long	AllEndReleasesDeformationsByLoadCombinationId ([i/o] SAFEARRAY(RDisplacementValues) StartDisplacements , [i/o] SAFEARRAY(RDisplacementValues) EndDisplacements) StartDisplacements displacement results for all end releases at the beginning of the lines (Length of the array = 2 * AxisVMMModel.Lines.Count) EndDisplacements displacement results for all end releases at the end of the lines (Length of the array = 2 * AxisVMMModel.Lines.Count) Retrieves all deformations of end releases according to the LoadCombinationId (and LoadLevelOrModeShapeOrTimeStep) property. Returns length of each array or an error code (errDatabaseNotReady , deInvalidCombinationOfLoadCaseAndLoadLevel , deLoadCombinationIndexOutOfBounds or deInvalidAnalysisType).
long	AllEnvelopeEndReleasesDeformations ([i/o] SAFEARRAY(RDisplacementValues) StartDisplacements , [i/o] SAFEARRAY(RDisplacementValues) EndDisplacements) StartDisplacements displacement results for all end releases at the beginning of the lines (Length of the array = 2 * AxisVMMModel.Lines.Count) EndDisplacements displacement results for all end releases at the end of the lines (Length of the array = 2 * AxisVMMModel.Lines.Count) Retrieves envelope of all end releases deformations. Envelope is identified by Component, MinMaxType and EnvelopeUID properties. Returns length of each array or an error code (errDatabaseNotReady or deInvalidAnalysisType).
long	AllCriticalEndReleasesDeformations ([i/o] SAFEARRAY(RDisplacementValues) StartDisplacements , [i/o] SAFEARRAY(RDisplacementValues) EndDisplacements) StartDisplacements displacement results for all end releases at the beginning of the lines (Length of the array = 2 * AxisVMMModel.Lines.Count) EndDisplacements displacement results for all end releases at the end of the lines (Length of the array = 2 * AxisVMMModel.Lines.Count) Retrieves critical of all end releases deformations. Critical combination is identified by Component and MinMaxType properties. Returns length of each array or an error code (errDatabaseNotReady , deCombinationTypeNotValidForCurrentNationalDesignCode or deInvalidAnalysisType).

Member Deformations

long **GetMemberDisplacementsByLoadCaseId** ([in] long **MemberID**, [in] long **LoadCaseId**,
[in] long **LoadLevelOrModeShapeOrTimeStep**, [in] [EAnalysisType](#) **AnalysisType**,
[in] [EBoolean](#) **WithReinforcement**, [out] SAFEARRAY([RDisplacementValues](#)) **Displacements**,
[out] SAFEARRAY(double) **PosX**

MemberID member index ($0 < MemberId \leq AxisVMMModel.Members.Count$)

LoadCaseId load case index

LoadLevelOrModeShapeOrTimeStep load level or mode shape or time step, according to the type of analysis. See [LoadLevelOrModeShapeOrTimeStep](#) parameter.

AnalysisType analysis type

WithReinforcement Get or set this to determine which displacements results should be read. Considering reinforcement (*lbTrue*) or not (*lbFalse*) (this setting is valid only if **AnalysisType** = *atNonLinearStatic*)

Displacements array of member displacements

PosX array of cross-section positions in *m* according to the local *x* direction

Retrieves displacements for each cross-section of a given element according to the **LoadCaseId** and **LoadLevelOrModeShapeOrTimeStep**. Returns the total number of cross-sections or an error code ([errDatabaseNotReady](#) or see [EDisplacementError](#)).

long **GetMemberDisplacementsByLoadCombinationId** ([in] long **MemberID**,
[in] long **LoadCombinationId**, [in] long **LoadLevelOrModeShapeOrTimeStep**,
[in] [EAnalysisType](#) **AnalysisType**, [in] [EBoolean](#) **WithReinforcement**,
[out] SAFEARRAY([RDisplacementValues](#)) **Displacements**, [out] SAFEARRAY(double) **PosX**)

MemberID member index ($0 < MemberId \leq AxisVMMModel.Members.Count$)

LoadCombinationId load combination index

LoadLevelOrModeShapeOrTimeStep load level or mode shape or time step, according to the type of analysis. See [LoadLevelOrModeShapeOrTimeStep](#) parameter.

AnalysisType analysis type

WithReinforcement Get or set this to determine which displacements results should be read. Considering reinforcement (*lbTrue*) or not (*lbFalse*) (this setting is valid only if **AnalysisType** = *atNonLinearStatic*)

Displacements array of member displacements

PosX array of cross-section positions in *m* according to the local *x* direction

Retrieves displacements for each cross-section of a given element according to the **LoadCombinationId** and **LoadLevelOrModeShapeOrTimeStep**. Returns the total number of cross-sections or an error code ([errDatabaseNotReady](#) or see [EDisplacementError](#)).

long	GetEnvelopeMemberDisplacements ([in] long MemberID, [in] long EnvelopeUID, [in] EMinMaxType MinMaxType, [in] EAnalysisType AnalysisType, [in] EDisplacement Component, [in] ELongBoolean WithReinforcement, [out] SAFEARRAY(RDDisplacementValues) Displacements, [out] SAFEARRAY(long) LoadCaseOrCombinationIds, [out] SAFEARRAY(long) LoadLevels, [out] SAFEARRAY(double) PosX)
	MemberID member index ($0 < MemberId \leq \text{AxisVMModel.Members.Count}$) EnvelopeUID unique envelope index MinMaxType minimum or maximum value AnalysisType analysis type Component force component WithReinforcement Get or set this to determine which displacements results should be read. Considering reinforcement (lbTrue) or not (lbFalse) (this setting is valid only if AnalysisType = atNonLinearStatic)
	Displacements array of member displacements LoadCaseOrCombinationIds array of load case or load combination indexes, if index is $> \text{IAxisVMLoadcases.count}$ then Load combination index = LoadCaseOrCombinationId – $\text{IAxisVMLoadcases.count}$
	LoadLevels array of load levels PosX array of cross-section positions in m according to the local x direction
	<i>Retrieves envelope displacements for each cross-section of a member. Envelope is identified by MinMaxType, Component and EnvelopeUID. Displacements array contains the result of the load case or combination in which Component is maximal or minimal. Returns the total number of cross-sections or an error code (errDatabaseNotReady or see EDisplacementError).</i>
long	GetCriticalMemberDisplacements ([in] long MemberID, [in] EMinMaxType MinMaxType, [in] ECombinationType CombinationType, [in] EAnalysisType AnalysisType, [in] EDisplacement Component, [out] SAFEARRAY(RDDisplacementValues) Displacements, [out] SAFEARRAY(double) PosX)
	MemberID member index ($0 < MemberId \leq \text{AxisVMModel.Members.Count}$) MinMaxType Minimum or maximum value CombinationType combination type AnalysisType analysis type Component force component Displacements array of member displacements PosX array of cross-section positions in m according to the local x direction
	<i>Retrieves critical displacements in each cross-section of a member. Critical combination is identified by Component and MinMaxType properties (e.g. IsSmin). Displacements array contains the result of the critical combination in which Component is maximal or minimal. Returns the total number of cross-sections or an error code (errDatabaseNotReady or see EDisplacementError).</i>

Save results to MetaFile functions

long	SaveCriticalMemberDisplacementsToMetaFile ([in] BSTR FileName , [in] long MemberId , [in] ECombinationType CombinationType , [in] EAnalysisType AnalysisType , [in] long Width , [in] long Height , [in] ELongBoolean EnvelopeOnly , [in] ELongBoolean RelativeToLeft , [in] ELongBoolean RelativeToRight , [in] double Position , [in] EWindowColourMode ColourMode)
FileName	<i>name of the file with extension emf</i>
MemberId	<i>member index</i>
CombinationType	<i>combination type</i>
AnalysisType	<i>analysis type</i>
Width	<i>picture's size in pixel (minimal acceptable value is 640)</i>
Height	<i>picture's size in pixel (minimal acceptable value is 580)</i>
EnvelopeOnly	<i>only Envelope</i>
RelativeToLeft	<i>relative deformations to the left endpoint (see the notes below)</i>
RelativeToRight	<i>relative deformations to the right endpoint (see the notes below)</i>
Position	<i>position of the investigated cross section along the member</i>
ColourMode	<i>colour mode</i>

It creates a metafile from the member's critical displacements. It returns MemberId or an error code ([EgeneralError](#) or see [EdisplacementError](#)).

long	SaveEnvelopeMemberDisplacementsToMetaFile ([in] BSTR FileName , [in] long MemberId , [in] long EnvelopeUID , [in] EAnalysisType AnalysisType , [in] long Width , [in] long Height , [in] ELongBoolean EnvelopeOnly , [in] ELongBoolean RelativeToLeft , [in] ELongBoolean RelativeToRight , [in] double Position , [in] EwindowColourMode ColourMode)
FileName	<i>name of the file with extension emf</i>
MemberId	<i>member index</i>
EnvelopeUID	<i>unique envelope index</i>
AnalysisType	<i>analysis type</i>
Width	<i>picture's size in pixel (minimally acceptable value is 640)</i>
Height	<i>picture's size in pixel (minimally acceptable value is 580)</i>
EnvelopeOnly	<i>only Envelope</i>
RelativeToLeft	<i>relative deformations to the left endpoint (see the notes below)</i>
RelativeToRight	<i>relative deformations to the right endpoint (see the notes below)</i>
Position	<i>position of the investigated cross section along the member</i>
ColourMode	<i>colour mode</i>

It creates a metafile from the member's displacements envelope. It returns MemberId or an error code ([EgeneralError](#) or see [EdisplacementError](#)).

long **SaveMemberDisplacementsToMetaFileByLoadCaseID** ([in] BSTR **FileName**, [in] long **MemberId**, [in] long **LoadCaseId**, [in] long **LoadLevelOrTimeStep**, [in] [EAnalysisType](#) **AnalysisType**, [in] long **Width**, [in] long **Height**, [in] [ElongBoolean](#) **RelativeToLeft**, [in] [ElongBoolean](#) **RelativeToRight**, [in] double **Position**, [in] [EWindowColourMode](#) **ColourMode**)

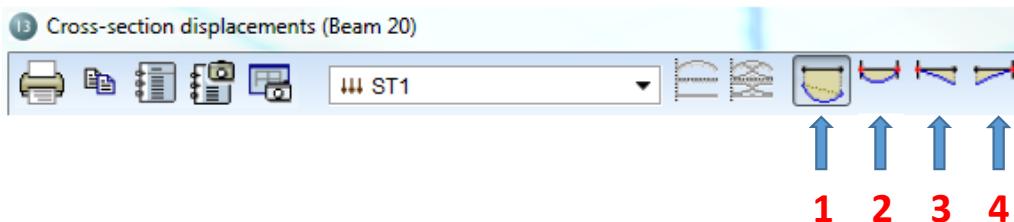
FileName	<i>name of the file with extension emf</i>
MemberId	<i>member index</i>
LoadCaseId	<i>load case index</i>
LoadLevelOrTimeStep	<i>for load level ($0 < \text{LoadLevelOrTimeStep} \leq \text{LoadLevelCount}$) for timestep ($0 < \text{LoadLevelOrTimeStep} \leq \text{TimeStepCount}$)</i>
AnalysisType	<i>analysis type</i>
Width	<i>picture's size in pixel (minimal acceptable value is 640)</i>
Height	<i>picture's size in pixel (minimal acceptable value is 580)</i>
RelativeToLeft	<i>relative deformations to the left endpoint (see the notes below)</i>
RelativeToRight	<i>relative deformations to the right endpoint (see the notes below)</i>
Position	<i>position of the investigated cross section along the member</i>
ColourMode	<i>colour mode</i>

It saves the member's displacements by LoadCaseId into a metafile. It returns MemberId or an error code ([EGeneralError](#) or see [EdisplacementError](#)).

long **SaveMemberDisplacementsToMetaFileByLoadCombinationID** ([in] BSTR **FileName**, [in] long **MemberId**, [in] long **LoadCombinationId**, [in] long **LoadLevelOrTimeStep**, [in] [EAnalysisType](#) **AnalysisType**, [in] long **Width**, [in] long **Height**, [in] [ElongBoolean](#) **RelativeToLeft**, [in] [ElongBoolean](#) **RelativeToRight**, [in] double **Position**, [in] [EWindowColourMode](#) **ColourMode**)

FileName	<i>name of the file with extension emf</i>
MemberId	<i>member index</i>
LoadCombinationId	<i>load combination index</i>
LoadLevelOrTimeStep	<i>for load level ($0 < \text{LoadLevelOrTimeStep} \leq \text{LoadLevelCount}$) for timestep ($0 < \text{LoadLevelOrTimeStep} \leq \text{TimeStepCount}$)</i>
AnalysisType	<i>analysis type</i>
Width	<i>picture's size in pixel (minimal acceptable value is 640)</i>
Height	<i>picture's size in pixel (minimal acceptable value is 580)</i>
RelativeToLeft	<i>relative deformations to the left endpoint (see the notes below)</i>
RelativeToRight	<i>relative deformations to the right endpoint (see the notes below)</i>
Position	<i>position of the investigated cross section along the member</i>
ColourMode	<i>colour mode</i>

It saves the member's displacements by LoadCombinationId into a metafile. It returns MemberId or an error code ([EGeneralError](#) or see [EdisplacementError](#)).



Notes to *RelativeToLeft* and *RelativeToRight* *ElongBooleans*:

1. *Actual displacements: RelativeToLeft := lbFalse and RelativeToRight := lbFalse;*
2. *Deformations relative to displaced endpoints: RelativeToLeft := lbTrue and RelativeToRight := lbTrue;*
3. *Deformations relative to displaced left endpoint: RelativeToLeft := lbTrue and RelativeToRight := lbFalse;*
4. *Deformations relative to displaced right endpoint: RelativeToLeft := lbFalse and RelativeToRight := lbTrue.*

long	SetUserCreep ([in] EBoolean Creep)
	Creep If <i>IbTrue</i> concrete creep will be considered in results of nonlinear analysis if national design code allows it. More here...
	Enable or disable consideration of concrete creep in nonlinear analysis results.. If successful returns 1, otherwise an error code (errDatabaseNotReady , errCreepNotSupported).

Properties

EAnalysisType	AnalysisType • Get or set the type of analysis (linear/nonlinear/vibration/buckling)
EDisplacement	Component • Get or set the displacement component for envelope or critical results
ECombinationType	CombinationType • Get or set the type of combination for critical results
long	EnvelopeUID • Get or set the unique index of the envelope used in functions for reading envelope results
EDisplacementSystem	DisplacementSystem • Get or set this to determine whether line's or member's displacement results are in global or local system
long	LoadCaseId • Get or set the load case index (0 < LoadCaseId ≤ AxisVMModel.LoadCases.Count)
long	LoadCombinationId • Get or set the load combination index (0 < LoadCombinationId ≤ AxisVMModel.LoadCombinations.Count)
long	LoadLevelOrModeShapeOrTimeStep • Get or set the load level or mode shape or time step, according to the type of analysis. See LoadLevelOrModeShapeOrTimeStep parameter.
EMinMaxType	MinMaxType • Get or set the if minimum or maximum values of the component will be read
EBoolean	UserCreep
	Returns <i>IbTrue</i> if nonlinear analysis results consider concrete creep. More here...
EBoolean	WithReinforcement • Get or set this to determine which nodal displacements results should be read. Considering reinforcement (<i>IbTrue</i>) or not (<i>IbFalse</i>) (this setting is valid only if AnalysisType = atNonLinearStatic)

IAxisVMForces

Interface containing internal forces within the model.

For further details of result objects see [Overview of AxisVM result objects](#).

If the model database is not available when calling functions or reading properties an [errDatabaseNotReady](#) error code is returned.

Error codes

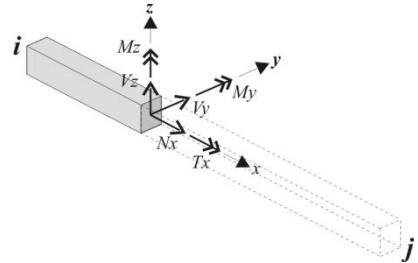
enum EForcesError = {	
feLineIndexOutOfBounds = -100001	<i>line index is out of bounds</i>
feLoadCaseIndexOutOfBounds = -100002	<i>LoadCaseId is out of bounds</i>
feLoadCombinationIndexOutOfBounds = -100003	<i>LoadCombinationId is out of bounds</i>
feNotValidLineType = -100004	<i>IAxisVMLine.LineType is not compatible with the reader function</i>
feSectionIndexOutOfBounds = -100005	<i>section index is out of bounds</i>
feCombinationTypeNotValidForCurrentNationalDesignCode = -100006	<i>CombinationType cannot be used for the selected design code or results not available for CombinationType</i>
feCOMError = -100007	<i>internal COM error when calling GetRecordInfoFromGUIDs, SafeArrayCreateEx, SafeArrayAccesssData</i>
feLineHasNoSections = -100008	<i>line element has no cross-sections (e.g. link element)</i>
feNoValidLinesInTheModel = -100009	<i>no valid line elements in the model</i>
feSurfaceIndexOutOfBounds = -100010	<i>surface index is out of bounds</i>
feInvalidSurfaceVertexType = -100011	<i>surface vertex type is invalid</i>
feInvalidAnalysisType = -100012	<i>AnalysisType is incompatible with the function</i>
feInvalidCombinationOfLoadCaseAndLoadLevel = -100013	<i>no result is available for the given LoadCaseId and LoadLevelOrModeShape or this</i>
feInvalidCombinationOfLoadCombinationAndLoadLevel = -100014	<i>no result is available for the given LoadCombinationId and LoadLevelOrModeShape or this</i>
feNoResultBlocksInTheModel = -100015	<i>no result blocks in the model</i>
feNodeIndexOutOfBounds = -100016	<i>node index is out of bounds</i>
feNoSurfacesInTheModel = -100017	<i>no surface elements in the model</i>
feNodalSupportIndexOutOfBounds = -100018	<i>nodal support index is out of bounds</i>
feLineSupportIndexOutOfBounds = -100019	<i>line support index is out of bounds</i>
feNoNodalSupportsInTheModel = -100020	<i>no nodal supports in the model</i>
feNoLineSupportsInTheModel = -100021	<i>no line supports in the model</i>
feSurfaceSupportIndexOutOfBounds = -100022	<i>surface support index is out of bounds</i>
feNoSurfaceSupportsInTheModel = -100023	<i>no surface supports in the model</i>
feInvalidLineType = -100024	<i>line type is invalid</i>
feNoSpringsInTheModel = -100025	<i>no springs in the model</i>
feNoGapsInTheModel = -100026	<i>no gaps in the model</i>
feEdgeConnectionIndexOutOfBounds = -100027	<i>edge connection index is out of bounds</i>
feNoEdgeConnectionsInTheModel = -100028	<i>no edge connections in the model</i>
feLinkElementIndexOutOfBounds = -100029	<i>link element index is out of bounds</i>
feNoLinkElementsInTheModel = -100030	<i>no link elements in the model</i>
feMemberIndexOutOfBounds = -100031	<i>member index is out of bounds</i>
feInvalidEnvelopeUID = -100032	<i>EnvelopeUID is invalid</i>
feZeroValidLineNumber = -100033 }	<i>there is no selected line based on EConnectionToNodeType</i>
feVirtualBeamIndexOutOfBounds = -100034	<i>virtual beam index is out of bounds</i>
feVirtualBeamChainIndexOutOfBounds = -100035	<i>virtual beam's chain index is out of bounds</i>
feVirtualBeamSectionIndexOutOfBounds = -100036	<i>chain's section index is out of bounds</i>
feWindowIdNotValid = -100037	<i>the given WindowId is not valid</i>
feMembersSupportIndexOutOfBounds = -100037 }	<i>Member support index is out of range</i>

Enumerated types

enum **ELineForce** = {

- IfNx** = 0x00 *Nx normal force [kN]*
- IfVy** = 0x01 *Vy shear force [kN]
(only for beams and ribs)*
- IfVz** = 0x02 *Vz shear force [kN]
(only for beams and ribs)*
- IfTx** = 0x03 *Tx twisting moment [kNm]
(only for beams and ribs)*
- IfMy** = 0x04 *My bending moment [kNm]
(only for beams and ribs)*
- IfMz** = 0x05 *Mz bending moment [kNm]
(only for beams and ribs)*
- IfMyD** = 0x06 } *My bending moment considering rib eccentricity [kNm]
(only for ribs)*

Line force component identifiers



enum **EVirtualBeamForce** = {

- vbfNx** = 0x00 *Nx normal force [kN]*
- vbfVy** = 0x01 *Vy shear force [kN]*
- vbfVz** = 0x02 *Vz shear force [kN]*
- vbfTx** = 0x03 *Tx twisting moment [kNm]*
- vbfMy** = 0x04 *My bending moment [kNm]*
- vbfMz** = 0x05 *Mz bending moment [kNm]*

Virtual beam/strip force component identifiers

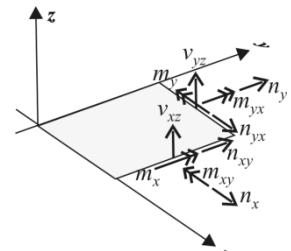
enum **EConnectedToNodeType** = {

- ctntAll** = 0x00 *connection forces related to all the connected line elements*
- ctntSelected** = 0x01 *connection forces related to only the selected connected line elements*
- ctntVisible** = 0x02 } *connection forces related to only the visible (in window with index WindowId) connected line elements (see e.g. IAxisVMLogicalParts)*

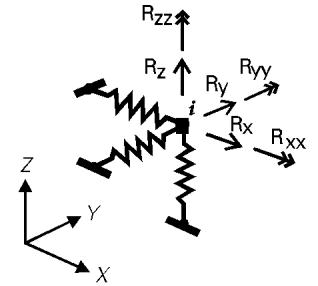
enum **ESurfaceForce** = {

- sfNx** = 0x00 *nx cross-section force [kN/m]*
- sfNy** = 0x01 *ny cross-section force [kN/m]*
- sfNxy** = 0x02 *nxy cross-section torsional force [kN/m]*
- sfMx** = 0x03 *mx bending moment [kNm/m]*
- sfMy** = 0x04 *my bending moment [kNm/m]*
- sfMxy** = 0x05 *mxy torsional moment [kNm/m]*
- sfVxz** = 0x06 *vxz shear force [kN/m]*
- sfVyz** = 0x07 *vyz shear force [kN/m]*
- sfVSz** = 0x08 *vSz resultant shear force [kN/m]*
- sfN1** = 0x09 *n1 1st principal force [kN/m]*
- sfN2** = 0x0A *n2 2nd principal force [kN/m]*
- sfAn** = 0x0B *an principal force direction [°]*
- sfM1** = 0x0C *m1 1st principal moment [kNm/m]*
- sfM2** = 0x0D *m2 2nd principal moment [kNm/m]*
- sfAm** = 0x0E *am principal moment direction [°]*
- sfNxD** = 0x0F *nxD design force [kN/m]*
- sfNyD** = 0x10 *nyD design force [kN/m]*
- sfMxDp** = 0x11 *mxD design moment(plus +) [kNm/m]*
- sfMxDm** = 0x12 *mxD design moment(minus -) [kNm/m]*
- sfMyDp** = 0x13 *myD design moment (plus +)[kNm/m]*
- sfMyDm** = 0x14 } *myD design moment (minus -)[kNm/m]*

Surface force component identifiers



enum **ENodalSupportForce** = {
nsfRx = 0x00 support force in local x direction [kN]
nsfRy = 0x01 support force in local y direction [kN]
nsfRz = 0x02 support force in local z direction [kN]
nsfRxx = 0x03 support moment about local x axis [kNm]
nsfRyy = 0x04 support moment about local y axis [kNm]
nsfRzz = 0x05 support moment about local z axis [kNm]
nsfRr = 0x06 resultant support force [kN]
nsfRrr = 0x07 resultant support moment [kNm]
nsfRalpha = 0x08 } ratio of force component in loc. xy plane to force
 in loc. z direction



Nodal support force component identifiers

enum **ELineSupportForce** = {
lsfRx = 0x00 support force in local x direction [kN/m]
lsfRy = 0x01 support force in local y direction [kN/m]
lsfRz = 0x02 support force in local z direction [kN/m]
lsfRxx = 0x03 support moment about local x axis [kNm/m]
lsfRyy = 0x04 support moment about local y axis [kNm/m]
lsfRzz = 0x05 } support moment about local z axis [kNm/m]

Line support force component identifiers

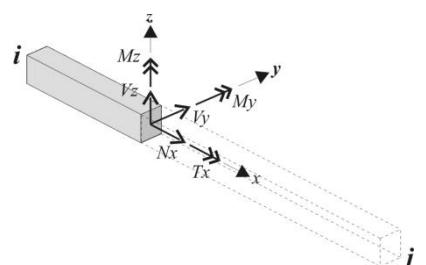
enum **ESurfaceSupportForce** = {
ssfRx = 0x00 support force in local x axis [kN/m²]
ssfRy = 0x01 support force in local y axis [kN/m²]
ssfRz = 0x02 } support force in local z axis [kN/m²]

Surface support force component identifiers

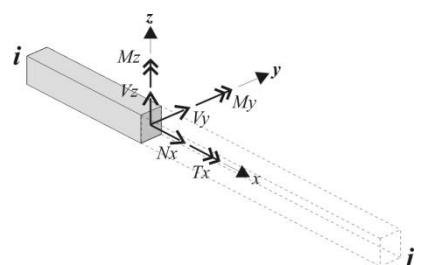
enum **ESpringForce** = {
sfRx = 0x00 force in local x direction [kN]
sfRy = 0x01 force in local y direction [kN]
sfRz = 0x02 force in local z direction [kN]
sfRxx = 0x03 moment about local x axis [kNm]
sfRyy = 0x04 moment about local y axis [kNm]
sfRzz = 0x05 } moment about local z axis [kNm]

Spring force component identifiers

enum **EEdgeConnectionForce** = {
ecfNx = 0x00 Nx normal force [kN/m]
ecfVy = 0x01 Vy shear force [kN/m]
ecfVz = 0x02 Vz shear force [kN/m]
ecfTx = 0x03 Tx twisting moment [kNm/m]
ecfMy = 0x04 My bending moment [kNm/m]
ecfMz = 0x05 } Mz bending moment [kNm/m]



enum **ELinkElementForce** = {
lefNx = 0x00 Nx normal force
 ([kN] for NN links and [kNm] for LL links)
lefVy = 0x01 Vy shear force
 ([kN] for NN links and [kNm] for LL links)
lefVz = 0x02 Vz shear force
 ([kN] for NN links and [kNm] for LL links)
lefTx = 0x03 Tx twisting moment
 ([kNm] for NN links and [kNm/m] for LL links)
lefMy = 0x04 My bending moment
 ([kNm] for NN links and [kNm/m] for LL links)



lefMz = 0x05 }	<i>Mz bending moment ([kNm] for NN links and [kNm/m] for LL links)</i>
enum ECapacityCurveType = {	
cctImportantCharacteristics = 0x00	<i>Instead of curve, typical characteristics are returned in array X in this order:</i>
	<i>Γ – transformation factor</i>
	<i>m* - mass of the equivalent SDOF system</i>
	<i>Fy* - yield force of the equivalent SDOF system</i>
	<i>dm* - maximal displacement of the equivalent bilinear system</i>
	<i>dy* - yield displacement of the equivalent SDOF system</i>
	<i>T* - natural period of vibration of the equivalent SDOF system</i>
	<i>det* - target displacement of a perfectly elastic system</i>
	<i>dt* - target displacement of the equivalent inelastic SDOF system</i>
	<i>dt - target displacement of the inelastic MDOF system ,</i>
	<i>du - maximum considered displacement of the MDOF system (in accordance with the 85% capacity rule in Italy),</i>
	<i>du* - maximum considered displacement of the equivalent SDOF system (in accordance with the 85% capacity rule in Italy),</i>
	<i>dmax - maximum expected displacement of the MDOF system,</i>
	<i>dmax* - maximum expected displacement of the equivalent SDOF system</i>
cctMDOF = 0x01	<i>MDOF capacity curve</i>
cctSDOF = 0x02	<i>Equivalent SDOF capacity curve</i>
cctEquivalentBilinear = 0x03	<i>Equivalent elastic bilinear capacity curve</i>
cctElasticADRS = 0x04	<i>elastic ADRS spectrum</i>
cctInelasticADRS = 0x05	<i>inelastic ADRS spectrum</i>
cctBilinearAD = 0x06	<i>Plastic-elastic acceleration-displacement curve</i>
cctSDOFinADSpace = 0x07	<i>Equivalent SDOF acceleration-displacement curve</i>
cctInitialPeriod = 0x08	<i>Straight line corresponding to initial stiffness period</i>
}	
enum ESubsoilClass = {	
scA_Type1 = 0x01	<i>Type 1 class A</i>
scB_Type1 = 0x02	<i>Type 1 class B</i>
scC_Type1 = 0x03	<i>Type 1 class C</i>
scD_Type1 = 0x04	<i>Type 1 class D</i>
scE_Type1 = 0x05	<i>Type 1 class E</i>
scA_Type2 = 0x06	<i>Type 2 class A</i>
scB_Type2 = 0x07	<i>Type 2 class B</i>
scC_Type2 = 0x08	<i>Type 2 class C</i>
scD_Type2 = 0x09	<i>Type 2 class D</i>
scE_Type2 = 0x10	<i>Type 2 class E</i>
}	
enum ETopographicCategory = {	
tcT1 = 0x01	<i>Category T1</i>
tcT2 = 0x02	<i>Category T2</i>
tcT3 = 0x03	<i>Category T3</i>
tcT4 = 0x04	<i>Category T4</i>
}	

```

enum ESeismicComponentSumType
{
    scstUsual = 0x00
        simple sum considering +/- sign of seismic component as it is
        from the static analysis
    scstCritical = 0x01
        critical sum that results greater value in absolute value
    scstNMyMz = 0x02
        it returns multiple results considering both +/- signs for N, My
        and Mz seismic components in summation
    scstMyVz = 0x03
        it returns multiple results considering both +/- signs for My and
        Vz seismic components in summation
    scstMzVy = 0x04
        it returns multiple results considering both +/- signs for Mz and
        Vy seismic components in summation
    scstN = 0x05
        it returns multiple results considering both +/- signs for N
        seismic component in summation
    scstNVyVz = 0x06
        it returns multiple results considering both +/- signs for N, Vy
        and Vz seismic components in summation
    scstNMyMx = 0x07
        it returns multiple results considering both +/- signs for N, My
        and Mx seismic components in summation
    scstNMzMx = 0x08
        it returns multiple results considering both +/- signs for N, Mz
        and Mx seismic components in summation
}

```

Summation rule of seismic internal force components.

Records / structures

RLineForceValues = (

ELineType

double **IfvNx**

double **IfvVy**

double **IfvVz**

double **IfvTx**

double **IfvMy**

double **IfvMz**

double **IfvMyD**

IfvLineType

line type

Nx normal force [kN]

Vy shear force [kN] (only for beams and ribs)

Vz shear force [kN] (only for beams and ribs)

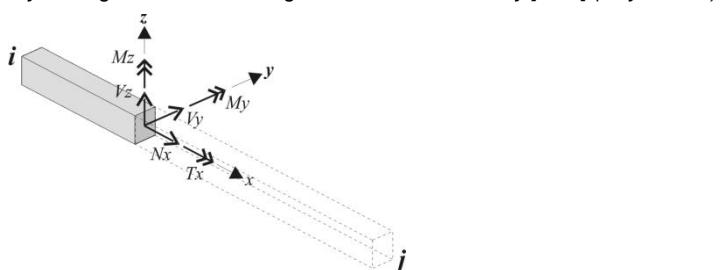
Tx twisting moment [kNm] (only for beams and ribs)

My bending moment [kNm] (only for beams and ribs)

Mz bending moment [kNm] (only for beams and ribs)

MyD design moment including the effect of eccentricity [kNm] (only for ribs)

)



RVirtualBeamForceValues = (

double **vbfvNx**

double **vbfvVy**

double **vbfvVz**

double **vbfvTx**

double **vbfvMy**

double **vbfvMz**

vbfvLineType

Nx normal force [kN]

Vy shear force [kN]

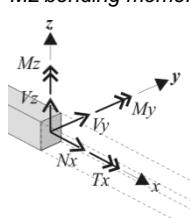
Vz shear force [kN]

Tx twisting moment [kNm]

My bending moment [kNm]

Mz bending moment [kNm]

)



RNodalSupportForceValues = (

double **Rx**

support force in local *x* direction [kN]

double **Ry**

support force in local *y* direction [kN]

double **Rz**

support force in local *z* direction [kN]

double **Rxx**

support moment about the local *x* axis [kNm]

double **Ryy**

support moment about the local *y* axis [kNm]

double **Rzz**

support moment about the local *z* axis [kNm]

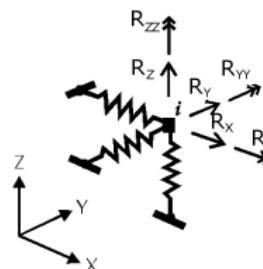
double **Rr**

resultant force [kN]

double **Rrr**

resultant moment [kNm]

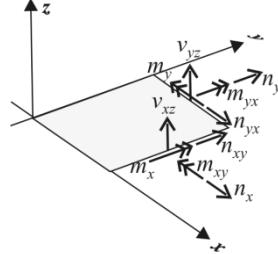
)



RSurfaceForceValues = (

double	sfvNx	<i>nx cross-section force [kN/m]</i>
double	sfvNy	<i>ny cross-section force [kN/m]</i>
double	sfvNxy	<i>nxy cross-section torsional force [kNm/m]</i>
double	sfvMx	<i>mx bending moment [kNm/m]</i>
double	sfvMy	<i>my bending moment [kNm/m]</i>
double	sfvMxy	<i>mxy torsional moment [kNm/m]</i>
double	sfvVxz	<i>vxz shear force [kN/m]</i>
double	sfvVyz	<i>vyz shear force [kN/m]</i>
double	sfvVs	<i>vSz resultant shear force [kN/m]</i>
double	sfvN1	<i>n1 1st principal force [kN/m]</i>
double	sfvN2	<i>n2 2nd principal force [kN/m]</i>
double	sfvAn	<i>an principal force direction [°]</i>
double	sfvM1	<i>m1 1st principal moment [kNm/m]</i>
double	sfvM2	<i>m2 2nd principal moment [kNm/m]</i>
double	sfvAm	<i>am principal moment direction [°]</i>
double	sfvNxD	<i>nxD design force [kN/m]</i>
double	sfvNyD	<i>nyD design force [kN/m]</i>
double	sfvMxDp	<i>mxD design moment (plus) [kNm/m]</i>
double	sfvMxDm	<i>mxD design moment (minus) [kNm/m]</i>
double	sfvMyDp	<i>myD design moment (plus) [kNm/m]</i>
double	sfvMyDm	<i>myD design moment (minus) [kNm/m]</i>

)



RSurfaceForces = (

long	ContourPointCount	<i>number of vertices of the surface polygon (3 or 4)</i>
long	ContourPoint1Id	<i>1st contour node index</i>
long	ContourPoint2Id	<i>2nd contour node index</i>
long	ContourPoint3Id	<i>3rd contour node index</i>
long	ContourPoint4Id	<i>4th contour node index</i>
long	ContourLine1Id	<i>1st contour line index</i>
long	ContourLine2Id	<i>2nd contour line index</i>
long	ContourLine3Id	<i>3rd contour line index</i>
long	ContourLine4Id	<i>4th contour line index</i>
	sfvCenterPoint	<i>forces at the surface centerpoint</i>
	sfvContourPoint1	<i>forces at the 1st contour node</i>
	sfvContourPoint2	<i>forces at the 2nd contour node</i>
	sfvContourPoint3	<i>forces at the 3rd contour node</i>
	sfvContourPoint4	<i>forces at the 4th contour node</i>
	sfvContourLineMidPoint1	<i>forces at the 1st contour line midpoint</i>
	sfvContourLineMidPoint2	<i>forces at the 2nd contour line midpoint</i>
	sfvContourLineMidPoint3	<i>forces at the 3rd contour line midpoint</i>
	sfvContourLineMidPoint4	<i>forces at the 4th contour line midpoint</i>

)

RLineSupportForceValues = (

double	Rx	<i>support force in local x direction [kN/m]</i>
double	Ry	<i>support force in local y direction [kN/m]</i>
double	Rz	<i>support force in local z direction [kN/m]</i>
double	Rxx	<i>support moment about the local x axis [kNm/m]</i>
double	Ryy	<i>support moment about the local y axis [kNm/m]</i>
double	Rzz	<i>support moment about the local z axis [kNm/m]</i>
double	Rr	<i>resultant force [kN/m]</i>
double	Rrr	<i>resultant moment [kNm/m]</i>

)

RSurfaceSupportForceValues = (

double	Rx	<i>support force in local x direction [kN/m²]</i>
double	Ry	<i>support force in local y direction [kN/m²]</i>
double	Rz	<i>support force in local z direction [kN/m²]</i>
)	

RSurfaceSupportForces = (

long	ContourPointCount	<i>number of vertices of the surface polygon (3 or 4)</i>
long	ContourPoint1Id	<i>1st contour node index</i>
long	ContourPoint2Id	<i>2nd contour node index</i>
long	ContourPoint3Id	<i>3rd contour node index</i>
long	ContourPoint4Id	<i>4th contour node index</i>
long	ContourLine1Id	<i>1st contour line index</i>
long	ContourLine2Id	<i>2nd contour line index</i>
long	ContourLine3Id	<i>3rd contour line index</i>
long	ContourLine4Id	<i>4th contour line index</i>
RSurfaceSupportForceValues	ssfvCenterPoint	<i>support forces at the surface centerpoint</i>
RSurfaceSupportForceValues	ssfvContourPoint1	<i>support forces at the 1st contour node</i>
RSurfaceSupportForceValues	ssfvContourPoint2	<i>support forces at the 2nd contour node</i>
RSurfaceSupportForceValues	ssfvContourPoint3	<i>support forces at the 3rd contour node</i>
RSurfaceSupportForceValues	ssfvContourPoint4	<i>support forces at the 4th contour node</i>
RSurfaceSupportForceValues	ssfvContourLineMidPoint1	<i>support forces at the 1st contour line midpoint</i>
RSurfaceSupportForceValues	ssfvContourLineMidPoint2	<i>support forces at the 2nd contour line midpoint</i>
RSurfaceSupportForceValues	ssfvContourLineMidPoint3	<i>support forces at the 3rd contour line midpoint</i>
RSurfaceSupportForceValues	ssfvContourLineMidPoint4	<i>support forces at the 4th contour line midpoint</i>
)	

RSpringForceValues = (

double	Rx	<i>force in local x direction [kN]</i>
double	Ry	<i>force in local y direction [kN]</i>
double	Rz	<i>force in local z direction [kN]</i>
double	Rxx	<i>moment about the local x axis [kNm]</i>
double	Ryy	<i>moment about the local y axis [kNm]</i>
double	Rzz	<i>moment about the local z axis [kNm]</i>
)	

REdgeConnectionForceValues = (

double	ecfvNx	<i>force in local x direction [kN]</i>
double	ecfvVy	<i>force in local y direction [kN]</i>
double	ecfvVz	<i>force in local z direction [kN]</i>
double	ecfvTx	<i>moment about the local x axis [kNm]</i>
double	ecfvMy	<i>moment about the local y axis [kNm]</i>
double	ecfvMz	<i>moment about the local z axis [kNm]</i>
)	

REdgeConnectionForces = (

REdgeConnectionForceValues	ecfSection1	<i>edge connection forces at the beginning of edge</i>
REdgeConnectionForceValues	ecfSection2	<i>edge connection forces at middle of edge</i>
REdgeConnectionForceValues	ecfSection3	<i>edge connection forces at the end of edge</i>
)	

RLinkElementForceValues = (

double	lefvNx	<i>force in local x direction [kN]</i>
double	lefvVy	<i>force in local y direction [kN]</i>
double	lefvVz	<i>force in local z direction [kN]</i>
double	lefvTx	<i>moment about the local x axis [kNm]</i>
double	lefvMy	<i>moment about the local y axis [kNm]</i>
double	lefvMz	<i>moment about the local z axis [kNm]</i>
)	

RLinkElementForces = (

ELinkElementType	lefLinkElementType	<i>type of the link element</i>
RLinkElementForceValues	lefSection1	<i>link element forces at the beginning of link</i>
RLinkElementForceValues	lefSection2	<i>link element forces at middle of link</i>
RLinkElementForceValues	lefSection3	<i>link element forces at the end of link</i>
)	

Gap Forces

Single EleMENt reader functions

long **GapForceByLoadCaseId** ([in] long **LinId**, [i/o] double **Force**, [out] BSTR **Combination**)

LinId line index ($0 < \text{LinId} \leq \text{AxisVMModel.Lines.Count}$)

Force gap force

Combination name of the load case

Retrieves gap force according to the LoadCaseId (and LoadLevelOrTimeStep) property. Returns LinId or an error code ([errDatabaseNotReady](#) or see [EForcesError](#)).

long **GapForceByLoadCombinationId** ([in] long **LinId**, [i/o] double **Force**, [out] BSTR **Combination**)

LinId line index ($0 < \text{LinId} \leq \text{AxisVMModel.Lines.Count}$)

Force gap force

Combination name of the load case

Retrieves gap force according to the LoadCombinationId (and LoadLevelOrTimeStep) property. Returns LinId or an error code ([errDatabaseNotReady](#) or see [EForcesError](#)).

long **EnvelopeGapForce** ([in] long **LinId**, [i/o] double **Force**, [out] BSTR **Combination**)

LinId line index ($0 < \text{LinId} \leq \text{AxisVMModel.Lines.Count}$)

Force gap force

Combination load case or combination in which gap force has its minimum or maximum

Retrieves envelope gap force. Envelope is identified by MinMaxType and EnvelopeUID properties. Force is minimum or maximum gap force. Returns LinId or an error code ([errDatabaseNotReady](#) or see [EForcesError](#)).

long **EnvelopeGapForce2** ([in] long **LinId**, [i/o] double **Force**, [out] long **LoadCaseOrCombinationId**,

[out] long **LoadLevel**)

LinId line index ($0 < \text{LinId} \leq \text{AxisVMModel.Lines.Count}$)

Force gap force

LoadCaseOrCombinationId load case or load combination index, if index is $> \text{IAxisVMLoadcases.count}$ then Load combination index = LoadCaseOrCombinationId - [IAxisVMLoadcases.count](#)

LoadLevel load level

Retrieves envelope gap force. Envelope is identified byMinMaxType and EnvelopeUID properties. Force is minimum or maximum gap force. Returns LinId or an error code ([errDatabaseNotReady](#) or see [EForcesError](#)).

long **CriticalGapForce** ([in] long **LinId**, [i/o] double **Force**, [out] BSTR **Combination**)

LinId line index ($0 < \text{LinId} \leq \text{AxisVMModel.Lines.Count}$)

Force gap force

Combination critical combination in which gap force has its minimum or maximum

Retrieves critical gap force. Critical combination is identified by the MinMaxType property. Force is minimum or maximum gap force. Returns LinId or an error code ([errDatabaseNotReady](#) or see [EForcesError](#)).

long **CriticalGapForce2** ([in] long **LinId**, [i/o] double **Force**, [out] **ECombinationType**

CriticalCombinationType, [out] SAFEARRAY(double) **Factors**, [out] SAFEARRAY(long) **LoadCaselds**)

CriticalCombinationType combination type corresponding to critical load combination

Factors load factors of the critical load combination

LoadCaselds load case indexes of the critical load combination

Similar to CriticalGapForce with more parameters described above.

Multiple element reader functions

long **AllGapForcesByLoadCaseId** ([i/o] SAFEARRAY(double) **Forces**)

Forces array of gap forces.

Length of the array is the number of gap elements in the model.

Retrieves all gap forces consecutively according to the LoadCaseId (and LoadLevelOrTimeStep) property. Returns the total number of records in the array or an error code ([errDatabaseNotReady](#) or see [EForcesError](#)).

long **AllGapForcesByLoadCombinationId** ([i/o] SAFEARRAY(double) **Forces**)

Forces array of gap forces.

Length of the array is the number of gap elements in the model.

Retrieves all gap forces consecutively according to the LoadCombinationId (and LoadLevelOrTimeStep) property. Returns the total number of records in the array or an error code ([errDatabaseNotReady](#) or see [EForcesError](#)).

long **AllEnvelopeGapForces** ([i/o] SAFEARRAY(double) **Forces**)

Forces array of gap forces.

Length of the array is the number of gap elements in the model.

Retrieves all envelope gap forces. Envelope is identified by MinMaxType and EnvelopeUID properties. Forces array contains minimum or maximum gap forces consecutively. Load case or combination in which gap force has its minimum or maximum can be read using the respective single element reader function.

Returns the array length or an error code ([errDatabaseNotReady](#) or see [EForcesError](#)).

long **AllCriticalGapForces** ([i/o] SAFEARRAY(double) **Forces**)

Forces array of gap forces.

Length of the array is the number of gap elements in the model.

Retrieves all critical gap forces. Critical combination is identified by the MinMaxType property. Forces array contains minimum or maximum gap forces. Critical combination in which gap force has its minimum or maximum can be read using the respective single element reader function. Returns the array length or an error code ([errDatabaseNotReady](#) or see [EForcesError](#)).

Multiple BLOCK reader functions

long **GapForcesForResultBlocks** ([in] long **LinId**,
[i/o] SAFEARRAY(**RResultBlockInfo**) **ResultBlockInfo**, [i/o] SAFEARRAY(double) **Forces**)

LinId line index ($0 < \text{LinId} \leq \text{AxisVMModel.Lines.Count}$)

ResultBlockInfo array of description records for result blocks

Forces force results for all result blocks

Retrieves gap forces in all result blocks consecutively. The number of result blocks depends on the AnalysisType property.

Returns the common length of ResultBlockInfo and Forces arrays or an error code ([errDatabaseNotReady](#) or see [EForcesError](#)).

Line Forces

Line forces are available only for trusses, beams and ribs.

If **Lineld** refers to a line of other type

- 1) the number of cross-sections will be zero.
- 2) If a single location reader functions is called [deSectionIndexOutOfBounds](#) error code is returned.
- 3) If a single element reader function is called a [deLineHasNoSections](#) error code is returned. If a multiple reader is called the SectionCounts array will contain zero at these line indexes.
- 4) AxisVM displays only the Member indexes. Use GetLines function from [IAxisVMMember](#) interface.

Single LOCATION reader functions

long **LineForceByLoadCaseld** ([in] long **Lineld**, [in] long **SectionId**,
[i/o] [RLineForceValues](#) **Force**, [out] double **PosX**, [out] BSTR **Combination**)

Lineld line index ($0 < \text{Lineld} \leq \text{AxisVMMModel.Lines.Count}$)

SectionId section index ($0 < \text{SectionId} \leq \text{AxisVMMModel.Lines[Lineld].SectionCount}$)

Force force results

PosX position of **SectionId** in m according to the local x direction

Combination name of the load case

Retrieves forces at a section of a line element according to the LoadCaseld (and LoadLevelOrTimeStep) property. Returns Lineld or an error code ([errDatabaseNotReady](#) or see [EForcesError](#)).

long **LineForceByLoadCombinationId** ([in] long **Lineld**, [in] long **SectionId**,
[i/o] [RLineForceValues](#) **Force**, [out] double **PosX**, [out] BSTR **Combination**)

Lineld line index ($0 < \text{Lineld} \leq \text{AxisVMMModel.Lines.Count}$)

SectionId section index ($0 < \text{SectionId} \leq \text{AxisVMMModel.Lines[Lineld].SectionCount}$)

Force force results

PosX position of **SectionId** in m according to the local x direction

Combination name of the load case

Retrieves forces at a section of a line element according to the LoadCombinationId (and LoadLevelOrTimeStep) property. Returns Lineld or an error code ([errDatabaseNotReady](#) or see [EForcesError](#)).

long **EnvelopeLineForce** ([in] long **Lineld**, [in] long **SectionId**,
[i/o] [RLineForceValues](#) **Force**, [out] double **PosX**, [out] BSTR **Combination**)

Lineld line index ($0 < \text{Lineld} \leq \text{AxisVMMModel.Lines.Count}$)

SectionId section index ($0 < \text{SectionId} \leq \text{AxisVMMModel.Lines[Lineld].SectionCount}$)

Force force results

PosX position of **SectionId** in m according to the local x direction

Combination load case or combination in which Component has its minimum or maximum

Retrieves envelope forces at a section of a line element. Envelope is identified by LineForceComponent, MinMaxType and EnvelopeUID properties. Force contains the result of the load case or combination in which Component is maximal or minimal. Returns Lineld or an error code ([errDatabaseNotReady](#) or see [EForcesError](#)).

long **EnvelopeLineForce2** ([in] long **Lineld**, [in] long **SectionId**,
[i/o] [RLineForceValues](#) **Force**, [out] double **PosX**, [out] long **LoadCaseOrCombinationId**,
[out] long **LoadLevel**)

Lineld line index ($0 < \text{Lineld} \leq \text{AxisVMMModel.Lines.Count}$)

SectionId section index ($0 < \text{SectionId} \leq \text{AxisVMMModel.Lines[Lineld].SectionCount}$)

Force force results

PosX position of **SectionId** in m according to the local x direction

LoadCaseOrCombinationId load case or load combination index, if index is $>$ [IAxisVMLoadcases](#).count then Load combination index = LoadCaseOrCombinationId - [IAxisVMLoadcases](#).count

LoadLevel load level

Retrieves envelope forces at a section of a line element. Envelope is identified by LineForceComponent, MinMaxType and EnvelopeUID properties. Force contains the result of the load case or combination in which Component is maximal or minimal. Returns Lineld or an error code ([errDatabaseNotReady](#) or see [EForcesError](#)).

long	CriticalLineForce ([in] long Lineld , [in] long SectionId , [i/o] RLineForceValues Force , [out] double PosX , [out] BSTR Combination)
	Lineld <i>line index ($0 < \text{Lineld} \leq \text{AxisVMMModel.Lines.Count}$)</i>
	SectionId <i>section index ($0 < \text{SectionId} \leq \text{AxisVMMModel.Lines[Lineld].SectionCount}$)</i>
	Force <i>force results</i>
	PosX <i>position of SectionId in m according to the local x direction</i>
	Combination <i>critical combination in which Component has its minimum or maximum</i>
	<i>Retrieves critical forces at a section of a line element. Critical combination is identified by Component and MinMaxType properties (e.g. Nx max). Force contains the result of the critical combination in which Component is maximal or minimal. Returns Lineld or an error code (errDatabaseNotReady or see EForcesError).</i>
long	CriticalLineForce2 ([in] long Lineld , [in] long SectionId , [i/o] RLineForceValues Force , [out] double PosX , [out] ECombinationType CriticalCombinationType , [out] SAFEARRAY(double) Factors , [out] SAFEARRAY(long) LoadCaselds)
	CriticalCombinationType <i>combination type corresponding to critical load combination</i>
	Factors <i>load factors of the critical load combination</i>
	LoadCaselds <i>load case indexes of the critical load combination</i>
	<i>Similar to CriticalLineForce with more parameters described above.</i>
long	LineForceByLoadCombinationIdEQ ([in] long Lineld , [in] long SectionId , [i/o] RLineForceValues Force , [out] double PosX , [out] BSTR Combination)
	Lineld <i>line index ($0 < \text{Lineld} \leq \text{AxisVMMModel.Lines.Count}$)</i>
	SectionId <i>section index ($0 < \text{SectionId} \leq \text{AxisVMMModel.Lines[Lineld].SectionCount}$)</i>
	Force <i>force results</i>
	PosX <i>position of SectionId in m according to the local x direction</i>
	Combination <i>name of the load case</i>
	<i>Retrieves forces at a section of a line element according to the LoadCombinationId (and LoadLevelOrTimeStep) property. Internal forces from seismic actions are taken into consideration based on SeismicComponentSumType property. Returns the length of Forces array or an error code (errDatabaseNotReady or see EForcesError).</i>
long	EnvelopeLineForceEQ ([in] long Lineld , [in] long SectionId , [i/o] RLineForceValues Force , [out] double PosX , [out] BSTR Combination)
	Lineld <i>line index ($0 < \text{Lineld} \leq \text{AxisVMMModel.Lines.Count}$)</i>
	SectionId <i>section index ($0 < \text{SectionId} \leq \text{AxisVMMModel.Lines[Lineld].SectionCount}$)</i>
	Force <i>force results</i>
	PosX <i>position of SectionId in m according to the local x direction</i>
	Combination <i>load case or combination in which Component has its minimum or maximum</i>
	<i>Retrieves envelope forces at a section of a line element. Envelope is identified by LineForceComponent, MinMaxType and EnvelopeUID properties. Force contains the result of the load case or combination in which Component is maximal or minimal. Internal forces from seismic actions are taken into consideration based on SeismicComponentSumType property. Returns the length of Forces array or an error code (errDatabaseNotReady or see EForcesError).</i>

long **CriticalLineForceEQ** ([in] long **Lineld**, [in] long **SectionId**,
 [i/o] [RLineForceValues](#) **Force**, [out] double **PosX**, [out] BSTR **Combination**)

Lineld line index ($0 < \text{Lineld} \leq \text{AxisVMMModel.Lines.Count}$)
SectionId section index ($0 < \text{SectionId} \leq \text{AxisVMMModel.Lines[Lineld].SectionCount}$)
Force force results
PosX position of **SectionId** in *m* according to the local x direction
Combination critical combination in which Component has its minimum or maximum

Retrieves critical forces at a section of a line element. Critical combination is identified by Component and MinMaxType properties (e.g. Nx max). Force contains the result of the critical combination in which Component is maximal or minimal. Internal forces from seismic actions are taken into consideration based on SeismicComponentSumType property. Returns the length of Forces array or an error code ([errDatabaseNotReady](#) or see [EForcesError](#)).

Single ELEMENT reader functions

long **LineForcesByLoadCaseld** ([in] long **Lineld**,
 [i/o] SAFEARRAY([RLineForceValues](#)) **Forces**, [out] SAFEARRAY(double) **PosX**)

Lineld line index ($0 < \text{Lineld} \leq \text{AxisVMMModel.Lines.Count}$)
Forces array of line forces for the line element.
 Length of the array is AxisVMMModel.Lines[Lineld].SectionCount
PosX array of cross-section positions in *m* according to the local x direction
 Length of the array is AxisVMMModel.Lines[Lineld].SectionCount

Retrieves forces for each cross-section of a given element according to the LoadCaseld (and LoadLevelOrTimeStep) property. Returns the total number of cross-sections or an error code ([errDatabaseNotReady](#) or see [EForcesError](#)).

long **LineForcesByLoadCombinationId** ([in] long **Lineld**,
 [i/o] SAFEARRAY([RLineForceValues](#)) **Forces**, [out] SAFEARRAY(double) **PosX**)

Lineld line index ($0 < \text{Lineld} \leq \text{AxisVMMModel.Lines.Count}$)
Forces array of line forces for the line element.
 Length of the array is AxisVMMModel.Lines[Lineld].SectionCount
PosX array of cross-section positions in *m* according to the local x direction
 Length of the array is AxisVMMModel.Lines[Lineld].SectionCount

Retrieves forces for each cross-section of a given element according to the LoadCombinationId (and LoadLevelOrTimeStep) property. Returns the total number of cross-sections or an error code ([errDatabaseNotReady](#) or see [EForcesError](#)).

long **EnvelopeLineForces** ([in] long **Lineld**,
 [i/o] SAFEARRAY([RLineForceValues](#)) **Forces**, [out] SAFEARRAY(double) **PosX**)

Lineld line index ($0 < \text{Lineld} \leq \text{AxisVMMModel.Lines.Count}$)
Forces array of envelope line forces for the line element.
 Length of the array is AxisVMMModel.Lines[Lineld].SectionCount
PosX array of cross-section positions in *m* according to the local x direction
 Length of the array is AxisVMMModel.Lines[Lineld].SectionCount

Retrieves envelope forces for each cross-section of a given element. Envelope is identified by LineForceComponent, MinMaxType and EnvelopeUID properties. Forces array contains the result of the load case or combination in which Component is maximal or minimal. Load case or combination in which Component has its minimum or maximum can be read using the respective single element/single cross-section reader function.

Returns the total number of cross-sections or an error code ([errDatabaseNotReady](#) or see [EForcesError](#)).

long **CriticalLineForces** ([in] long **Lineld**,
[i/o] SAFEARRAY([RLineForceValues](#)) **Forces**, [out] SAFEARRAY(double) **PosX**)

Lineld line index ($0 < \text{Lineld} \leq \text{AxisVMMModel.Lines.Count}$)
Forces array of critical line forces for the line element.
Length of the array is `AxisVMMModel.Lines[Lineld].SectionCount`
PosX array of cross-section positions in m according to the local x direction
Length of the array is `AxisVMMModel.Lines[Lineld].SectionCount`

Retrieves critical forces in each cross-section of a given element. Critical combination is identified by Component and MinMaxType properties (e.g. Nx max). Forces array contains the result of the critical combination in which Component is maximal or minimal. Critical combination in which Component has its minimum or maximum can be read using the respective single element/single cross-section reader function.

Returns the total number of cross-sections or an error code ([errDatabaseNotReady](#) or see [EForcesError](#)).

long **LineForcesByLoadCaselDConnectedToNode** ([in] long **Nodeld**, [in] long **Windowld**,
[in] [EConnectedToNodeType](#) **ConnectedToNodeType**,
[out] SAFEARRAY([RLineForceValues](#)) **Forces**, [out] SAFEARRAY(long) **ConnectedLinelds**,
[out] SAFEARRAY(BSTR) **Combinations**)

Nodeld node index the lines connected to ($0 < \text{Nodeld} \leq \text{AxisVMMModel.Nodes.Count}$)
Windowld active window index ($0 < \text{Windowld} \leq \text{AxisVMMModel.Windows.Count}$)
(Note: used only if `EConnectedToNodeType = ctntVisible`)
ConnectedToNodeType selection type
Forces array of line force results
ConnectedLinelds array of connected line indices
Combinations array of names of the load cases

Retrieves line force values of connecting line elements (beam, truss and rib elements are supported) at end node identified with Nodeld index according to the LoadCaselD (and LoadLevelOrTimeStep) property. Returns the number of connected lines based on selection type or an error code ([errDatabaseNotReady](#) or see [EForcesError](#)).

Note: the function retrieves the line force values according to the local coordinate system of the element!

long **LineForcesByLoadCombinationIdConnectedToNode** ([in] long **Nodeld**,
[in] long **Windowld**, [in] [EConnectedToNodeType](#) **ConnectedToNodeType**,
[out] SAFEARRAY([RLineForceValues](#)) **Forces**,
[out] SAFEARRAY(long) **ConnectedLinelds**, [out] SAFEARRAY(BSTR) **Combinations**)

Nodeld node index the lines connected to ($0 < \text{Nodeld} \leq \text{AxisVMMModel.Nodes.Count}$)
Windowld active window index ($0 < \text{Windowld} \leq \text{AxisVMMModel.Windows.Count}$)
(Note: used only if `EConnectedToNodeType = ctntVisible`)
ConnectedToNodeType selection type
Forces array of line force results
ConnectedLinelds array of connected line indices
Combinations array of names of the load combinations

Retrieves line force values of connecting line elements (beam, truss and rib elements are supported) at end node identified with Nodeld index according to the LoadCombinationId (and LoadLevelOrTimeStep) property. Internal forces from seismic actions are taken into consideration based on SeismicComponentSumType property. Returns the length of Forces array or an error code ([errDatabaseNotReady](#) or see [EForcesError](#)).

Note: the function retrieves the line force values according to the local coordinate system of the element!

long **EnvelopeLineForcesConnectedToNode** ([in] long **NodId**, [in] long **WindowId**,
[in] [EConnectedToNodeType](#) **ConnectedToNodeType**, [in] long **LinId**,
[out] SAFEARRAY([RLineForceValues](#)) **Forces**,
[out] SAFEARRAY(long) **ConnectedLinIds**, [out] SAFEARRAY(BSTR) **Combinations**)

NodId node index the lines connected to ($0 < \text{NodId} \leq \text{AxisVMMModel.Nodes.Count}$)

WindowId active window index ($0 < \text{WindowId} \leq \text{AxisVMMModel.Windows.Count}$)
(Note: used only if **EConnectedToNodeType** = **ctntVisible**)

ConnectedToNodeType selection type

Component reference component (N_x, V_y , etc.)

LinId reference line connected to the node

Forces array of envelope line force results

ConnectedLinIds array of connected line indices

Combinations array of names of the load cases/load combinations

*Retrieves envelope line force values of connecting line elements (beam, truss and rib elements are supported) at end node identified with **NodId** index. Reference component (**LineForceComponent** property) and line index need to be given because coherent results are collected related to other components and lines. Load combinations/ load cases are identified by **LineForceComponent** and **MinMaxType** properties (e.g. N_x max) related to line with **LinId** index. Forces array contains the results of the load combination/ load case in which **Component** is maximal or minimal. Internal forces from seismic actions are taken into consideration based on **SeismicComponentSumType** property. Returns the length of **Forces** array or an error code ([errDatabaseNotReady](#) or see [EForcesError](#)).*

Note: the function retrieves the line force values according to the local coordinate system of the element!

long **CriticalLineForcesConnectedToNode** ([in] long **NodId**, [in] long **WindowId**,
[in] [EConnectedToNodeType](#) **ConnectedToNodeType**, [in] long **LinId**,
[out] SAFEARRAY([RLineForceValues](#)) **Forces**,
[out] SAFEARRAY(long) **ConnectedLinIds**, [out] SAFEARRAY(BSTR) **Combinations**)

NodId node index the lines connected to ($0 < \text{NodId} \leq \text{AxisVMMModel.Nodes.Count}$)

WindowId active window index ($0 < \text{WindowId} \leq \text{AxisVMMModel.Windows.Count}$)
(Note: used only if **EConnectedToNodeType** = **ctntVisible**)

ConnectedToNodeType selection type

LinId reference line connected to the node

Forces array of critical line force results

ConnectedLinIds array of connected line indices

Combinations array of names of the load cases/load combinations

*Retrieves critical line force values of connecting line elements (beam, truss and rib elements are supported) at end node identified with **NodId** index. Reference component (**LineForceComponent** property) and line index need to be given because coherent results are collected related to other components and lines. Critical combination is identified by **LineForceComponent** and **MinMaxType** properties (e.g. N_x max) related to line with **LinId** index. Forces array contains the results of the critical combination in which **Component** is maximal or minimal. Internal forces from seismic actions are taken into consideration based on **SeismicComponentSumType** property. Returns the length of **Forces** array or an error code ([errDatabaseNotReady](#) or see [EForcesError](#)).*

Note: the function retrieves the line force values according to the local coordinate system of the element!

Multiple element reader functions

long **AllLineForcesByLoadCaseId** ([out] SAFEARRAY(long) **SectionCounts**,
[i/o] SAFEARRAY([RLineForceValues](#)) **Forces**, [out] SAFEARRAY(double) **PosX**)

SectionCounts array containing the section counts of line elements.
Length of the array = AxisVMMModel.Lines.Count

Forces array of line forces.
For each line element it contains results for SectionCount sections consecutively.
Length of the array is the sum of the SectionCounts array elements

PosX array of cross-section positions in m according to the local x direction of each line element
Length of the array is the sum of the SectionCounts array elements

Retrieves forces of all lines and cross-sections consecutively according to the LoadCaseId (and LoadLevelOrTimeStep) property. Returns the total number of records in the array or an error code ([errDatabaseNotReady](#) or see [EForcesError](#)).

long **AllLineForcesByLoadCombinationId** ([out] SAFEARRAY(long) **SectionCounts**,
[i/o] SAFEARRAY([RLineForceValues](#)) **Forces**, [out] SAFEARRAY(double) **PosX**)

SectionCounts array containing the section counts of line elements.
Length of the array = AxisVMMModel.Lines.Count

Forces array of line forces.
For each line element it contains results for SectionCount sections consecutively.
Length of the array is the sum of the SectionCounts array elements

PosX array of cross-section positions in m according to the local x direction of each line element
Length of the array is the sum of the SectionCounts array elements

Retrieves forces of all lines and cross-sections consecutively according to the LoadCombinationId (and LoadLevelOrTimeStep) property. Returns the total number of records in the array or an error code ([errDatabaseNotReady](#) or see [EForcesError](#)).

long **AllEnvelopeLineForces** ([out] SAFEARRAY(long) **SectionCounts**,
[i/o] SAFEARRAY([RLineForceValues](#)) **Forces**, [out] SAFEARRAY(double) **PosX**)

SectionCounts array containing the section counts of line elements.
Length of the array = AxisVMMModel.Lines.Count

Forces array of envelope line forces for all line elements.
For each line element it contains results for SectionCount sections consecutively.
Length of the array is the sum of the SectionCounts array elements

PosX array of cross-section positions in m according to the local x direction of each line element
Length of the array is the sum of the SectionCounts array elements

Retrieves envelope forces of all line elements. Envelope is identified by LineForceComponent, MinMaxType and EnvelopeUID properties. Forces array contains the result of the load case or combination in which Component is maximal or minimal. Load case or combination in which Component has its minimum or maximum can be read using the respective single location reader function.

Returns the total number of cross-sections or an error code ([errDatabaseNotReady](#) or see [EForcesError](#)).

long	AllCriticalLineForces ([out] SAFEARRAY(long) SectionCounts , [i/o] SAFEARRAY(RLineForceValues) Forces , [out] SAFEARRAY(double) PosX)
	SectionCounts array containing the section counts of line elements. <i>Length of the array = AxisVMModel.Lines.Count</i>
	Forces array of critical line forces for all line elements. <i>For each line element it contains results for SectionCount sections consecutively.</i> <i>Length of the array is the sum of the SectionCounts array elements</i>
	PosX array of cross-section positions in m according to the local x direction of each line element <i>Length of the array is the sum of the SectionCounts array elements</i>
	<i>Retrieves critical forces of all line elements. Critical combination is identified by Component and MinMaxType properties (e.g. Nx max). Forces array contains the result of the critical combination in which Component is maximal or minimal. Critical combination in which Component has its minimum or maximum can be read using the respective single location reader function.</i> <i>Returns the total number of cross-sections or an error code (errDatabaseNotReady or see EForcesError).</i>

Multiple BLOCK reader functions

long	LineForcesForResultBlocks ([in] long Lineld , [in] long SectionId , [i/o] SAFEARRAY(RResultBlockInfo) ResultBlockInfo , [i/o] SAFEARRAY(RLineForceValues) Forces , [out] SAFEARRAY(double) PosX)
	Lineld line index ($0 < \text{Lineld} \leq \text{AxisVMModel.Lines.Count}$)
	SectionId section index ($0 < \text{SectionId} \leq \text{AxisVMModel.Lines[Lineld].SectionCount}$)
	ResultBlockInfo array of description records for result blocks
	Forces force results for all result blocks
	PosX array of cross-section positions in m according to the local x direction of each line element
	<i>Retrieves forces at the given line and cross-section in all result blocks consecutively. The number of result blocks depends on the AnalysisType property.</i> <i>Returns the common length of ResultBlockInfo, Forces and PosX arrays or an error code (errDatabaseNotReady or see EForcesError).</i>

Virtual Beam and Virtual Strip Forces

Single ELEMENT reader functions

long **VirtualBeamOrStripForcesByLoadCaselid** ([in] long **Index**, [in] long **ChainId**,
[i/o] SAFEARRAY([RVirtualBeamForceValues](#))* **Forces**, [i/o] SAFEARRAY([RPoint3d](#))* **Pos**)

Index virtual beam/strip index ($0 < Index \leq \text{AxisVMModel.VirtualBeams.Count}$)
ChainId virtual beam's/strip's chain index
($0 < ChainId \leq \text{AxisVMModel.VirtualBeams.ChainCount[Index]}$)
Forces array of virtual beam/strip forces
Length of the array is $\text{AxisVMModel.VirtualBeams.SectionCount[Index, ChainId]}$
Pos array of section positions in m according to the global coordinate system
Length of the array is $\text{AxisVMModel.VirtualBeams.SectionCount[Index, ChainId]}$

Retrieves forces for each sections of a given chain of a virtual beam/strip according to the LoadCaselid (and LoadLevelOrTimeStep) property. Returns the total number of sections of the chain or an error code ([errDatabaseNotReady](#) or see [EForcesError](#)).

long **VirtualBeamOrStripForcesByLoadCombinationId** ([in] long **Index**, [in] long **ChainId**,
[i/o] SAFEARRAY([RVirtualBeamForceValues](#))* **Forces**, [i/o] SAFEARRAY([RPoint3d](#))* **Pos**)

Index virtual beam/strip index ($0 < Index \leq \text{AxisVMModel.VirtualBeams.Count}$)
ChainId virtual beam's/strip's chain index
($0 < ChainId \leq \text{AxisVMModel.VirtualBeams.ChainCount[Index]}$)
Forces array of virtual beam/strip forces
Length of the array is $\text{AxisVMModel.VirtualBeams.SectionCount[Index, ChainId]}$
Pos array of section positions in m according to the global coordinate system
Length of the array is $\text{AxisVMModel.VirtualBeams.SectionCount[Index, ChainId]}$

Retrieves forces for each sections of a given chain of a virtual beam/strip according to the LoadCombinationId (and LoadLevelOrTimeStep) property. Returns the total number of sections of the chain or an error code ([errDatabaseNotReady](#) or see [EForcesError](#)).

long **EnvelopeVirtualBeamOrStripForces** ([in] long **Index**, [in] long **ChainId**,
[i/o] SAFEARRAY([RVirtualBeamForceValues](#))* **Forces**, [i/o] SAFEARRAY([RPoint3d](#))* **Pos**,
[out] SAFEARRAY(BSTR)* **LoadCasesOrLoadCombinations**)

Index virtual beam/strip index ($0 < Index \leq \text{AxisVMModel.VirtualBeams.Count}$)
ChainId virtual beam's/strip's chain index
($0 < ChainId \leq \text{AxisVMModel.VirtualBeams.ChainCount[Index]}$)
Forces array of virtual beam/strip forces
Length of the array is $\text{AxisVMModel.VirtualBeams.SectionCount[Index, ChainId]}$
Pos array of section positions in m according to the global coordinate system
Length of the array is $\text{AxisVMModel.VirtualBeams.SectionCount[Index, ChainId]}$

LoadCasesOrLoadCombinations load cases or combinations in which VirtualBeamForceComponent has its minimum or maximum
Retrieves envelope forces for each sections of a given chain of a virtual beam/strip. Envelope is identified by VirtualBeamForceComponent, MinMaxType and EnvelopeUID properties.
LoadCasesOrLoadCombinations array contains the name of load cases or combinations in which Component is maximal or minimal. Returns the total number of sections or an error code ([errDatabaseNotReady](#) or see [EForcesError](#)).

long	EnvelopeVirtualBeamOrStripForces2 ([in] long Index , [in] long ChainId , [i/o] SAFEARRAY(RVirtualBeamForceValues)* Forces , [i/o] SAFEARRAY(RPoint3d)* Pos , [out] SAFEARRAY(long)* LoadCaseOrCombinationIds , [out] SAFEARRAY(long)* LoadLevels)
	Index virtual beam/strip index ($0 < \text{Index} \leq \text{AxisVMModel.VirtualBeams.Count}$) ChainId virtual beam's/strip's chain index $(0 < \text{ChainId} \leq \text{AxisVMModel.VirtualBeams.ChainCount[Index]}$) Forces array of virtual beam/strip forces $\text{Length of the array is AxisVMModel.VirtualBeams.SectionCount[Index], ChainId}$ Pos array of section positions in m according to the global coordinate system $\text{Length of the array is AxisVMModel.VirtualBeams.SectionCount[Index], ChainId}$ LoadCaseOrCombinationIds array of load case or load combination indices, if any is > IAxisVMLoadcases .count then Load combination index = LoadCaseOrCombinationId – IAxisVMLoadcases .count LoadLevels array of load levels according to LoadCaseOrCombinationIds
	<p>Retrieves envelope forces for each sections of a given chain of a virtual beam/strip. Envelope is identified by VirtualBeamForceComponent, MinMaxType and EnvelopeUID properties.</p> <p>LoadCaseOrLoadCombinationIds array contains the index of load cases or combinations in which Component is maximal or minimal. Returns the total number of sections or an error code (errDatabaseNotReady or see EForcesError).</p>
long	CriticalVirtualBeamOrStripForce ([in] long Index , [in] long ChainId , [in] long SectionId , [i/o] RVirtualBeamForceValues Forces , [i/o] RPoint3d Pos , [out] BSTR Combination)
	Index virtual beam/strip index ($0 < \text{Index} \leq \text{AxisVMModel.VirtualBeams.Count}$) ChainId virtual beam's/strip's chain index $(0 < \text{ChainId} \leq \text{AxisVMModel.VirtualBeams.ChainCount[Index]}$) SectionId chain's section index ($0 < \text{SectionId} \leq \text{AxisVMModel.VirtualBeams.SectionCount[Index], ChainId}$) Force virtual beam forces Pos array of section position in m according to the global coordinate system Combination critical combination in which VirtualBeamForceComponent has its minimum or maximum
	<p>Retrieves critical forces for a section of a given chain of a virtual beam/strip. Critical combination is identified by VirtualBeamForceComponent and MinMaxType properties (e.g. Nx max).</p> <p>Returns the section index or an error code (errDatabaseNotReady or see EForcesError).</p>
long	CriticalVirtualBeamOrStripForce2 ([in] long Index , [in] long ChainId , [in] long SectionId , [i/o] RVirtualBeamForceValues Forces , [i/o] RPoint3d Pos , [out] ECombinationType CriticalCombinationType , [out] SAFEARRAY(double)* Factors , [out] SAFEARRAY(long)* LoadCaselds)
	CriticalCombinationType combination type corresponding to critical load combination Factors load factors of the critical load combination LoadCaselds load case indexes of the critical load combination
	<p>Similar to CriticalVirtualBeamForce with more parameters described above. Returns the numbers of load cases considered in critical combination or an error code (errDatabaseNotReady or see EForcesError).</p>

long **CriticalVirtualBeamOrStripForces** ([in] long **Index**, [in] long **ChainId**,
[i/o] SAFEARRAY([RVirtualBeamForceValues](#))* **Forces**, [i/o] SAFEARRAY([RPoint3d](#))* **Pos**, [out]
SAFEARRAY(BSTR)* **Combinations**)

Index virtual beam/strip index ($0 < \text{Index} \leq \text{AxisVMMModel.VirtualBeams.Count}$)

ChainId virtual beam's/strip's chain index

($0 < \text{ChainId} \leq \text{AxisVMMModel.VirtualBeams.ChainCount[Index]}$)

Forces array of virtual beam/strip forces

Length of the array is $\text{AxisVMMModel.VirtualBeams.SectionCount[Index, ChainId]}$

Pos array of section positions in m according to the global coordinate system

Length of the array is $\text{AxisVMMModel.VirtualBeams.SectionCount[Index, ChainId]}$

Combinations array of critical combinations in which *VirtualBeamForceComponent* has its minimum or maximum

Retrieves critical forces for each sections of a given chain of a virtual beam/strip. Critical combination is identified by *VirtualBeamForceComponent* and *MinMaxType* properties (e.g. Nx max). Forces array contains the result of critical combination in which Component is maximal or minimal.

Returns the total number of sections or an error code ([errDatabaseNotReady](#) or see [EForcesError](#)).

Save results to MetaFile functions

long **SaveCriticalVirtualBeamOrStripForcesToMetaFile** ([in] BSTR **FileName**, [in] long **ID**,
[in] [ECombinationType](#) **CombinationType**, [in] [EAnalysisType](#) **AnalysisType**, [in] long **Width**,
[in] long **Height**, [in] [ELongBoolean](#) **EnvelopeOnly**, [in] double **Position**,
[in] [EWindowColourMode](#) **ColourMode**)

FileName	<i>name of the file with extension emf</i>
ID	<i>index of the virtual beam or strip</i>
CombinationType	<i>combination type</i>
AnalysisType	<i>analysis type</i>
Width	<i>picture's size in pixel (minimal acceptable value is 640)</i>
Height	<i>picture's size in pixel (minimal acceptable value is 580)</i>
EnvelopeOnly	<i>only Envelope</i>
Position	<i>position of the investigated cross section along the virtual beam or strip</i>
ColourMode	<i>colour mode</i>

It creates a metafile from the virtual beam's or strip's critical forces. It returns ID or an error code ([EGeneralError](#) or see [EForcesError](#)).

long **SaveEnvelopeVirtualBeamOrStripForcesToMetaFile** ([in] BSTR **FileName**, [in] long **ID**,
[in] long **EnvelopeUID**, [in] [EAnalysisType](#) **AnalysisType**, [in] long **Width**, [in] long **Height**,
[in] [ELongBoolean](#) **EnvelopeOnly**, [in] double **Position**, [in] [EWindowColourMode](#) **ColourMode**)

FileName	<i>name of the file with extension emf</i>
ID	<i>index of the virtual beam or strip</i>
EnvelopeUID	<i>unique envelope index</i>
AnalysisType	<i>analysis type</i>
Width	<i>picture's size in pixel (minimal acceptable value is 640)</i>
Height	<i>picture's size in pixel (minimal acceptable value is 580)</i>
EnvelopeOnly	<i>only Envelope</i>
Position	<i>position of the investigated cross section along the virtual beam or strip</i>
ColourMode	<i>colour mode</i>

It creates a metafile from the virtual beam's or strip's forces envelope. It returns ID or an error code ([EGeneralError](#) or see [EForcesError](#)).

long **SaveVirtualBeamOrStripForcesToMetaFileByLoadCaseID** ([in] BSTR **FileName**, [in] long **ID**,
[in] long **LoadCaseID**, [in] long **LoadLevelOrTimeStep**, [in] [EAnalysisType](#) **AnalysisType**,
[in] long **Width**, [in] long **Height**, [in] double **Position**, [in] [EWindowColourMode](#) **ColourMode**)

FileName	<i>name of the file with extension emf</i>
ID	<i>index of the virtual beam or strip</i>
LoadCaseID	<i>load case index</i>
LoadLevelOrTimeStep	<i>for load level ($0 < \text{LoadLevelOrTimeStep} \leq \text{LoadLevelCount}$)</i> <i>for timestep ($0 < \text{LoadLevelOrTimeStep} \leq \text{TimeStepCount}$)</i>
AnalysisType	<i>analysis type</i>
Width	<i>picture's size in pixel (minimal acceptable value is 640)</i>
Height	<i>picture's size in pixel (minimal acceptable value is 580)</i>
Position	<i>position of the investigated cross section along the virtual beam or strip</i>
ColourMode	<i>colour mode</i>

It saves the virtual beam's or strip's forces by LoadCaseID into a metafile. It returns ID or an error code ([EGeneralError](#) or see [EForcesError](#)).

long **SaveVirtualBeamOrStripForcesToMetaFileByLoadCombinationID** ([in] BSTR **FileName**, [in] long **ID**, [in] long **LoadCombinationId**, [in] long **LoadLevelOrTimeStep**, [in] **EAnalysisType AnalysisType**, [in] long **Width**, [in] long **Height**, [in] double **Position**, [in] **EWindowColourMode ColourMode**)

FileName *name of the file with extension emf*

ID *index of the virtual beam or strip*

LoadCombinationId *load combination index*

LoadLevelOrTimeStep *for load level ($0 < LoadLevelOrTimeStep \leq LoadLevelCount$)*

for timestep ($0 < LoadLevelOrTimeStep \leq TimeStepCount$)

AnalysisType *analysis type*

Width *picture's size in pixel (minimal acceptable value is 640)*

Height *picture's size in pixel (minimal acceptable value is 580)*

Position *position of the investigated cross section along the virtual beam or strip*

ColourMode *colour mode*

It saves the virtual beam's or strip's forces by LoadCombinationId into a metafile. It returns ID or an error code ([EGeneralError](#) or see [EForcesError](#)).

Line Support Forces

Single LOCATION reader functions

long **LineSupportForceByLoadCaseId** ([in] long **LineSupportId**, [in] long **LineSectionId**,
[i/o] [RLineSupportForceValues](#) **Force**, [out] double **PosX**, [out] BSTR **Combination**)

LineSupportId line support index ($0 < \text{LineSupportId} \leq \text{AxisVMMModel.LineSupports.Count}$)

LineSectionId section index ($0 < \text{SectionId} \leq \text{AxisVMMModel.LineSupports.SectionCount}[\text{LineSupportId}, \text{AnalysisType}]$)

Force force results

PosX position of SectionId in m according to the local x direction

Combination name of the load case

Retrieves line support forces at a section of a line element according to the LoadCaseId (and LoadLevelOrTimeStep) property. Returns LineSupportId or an error code ([errDatabaseNotReady](#) or see [EForcesError](#)).

long **LineSupportForceByLoadCombinationId** ([in] long **LineSupportId**, [in] long **LineSectionId**,
[i/o] [RLineSupportForceValues](#) **Force**, [out] double **PosX**, [out] BSTR **Combination**)

LineSupportId line support index ($0 < \text{LineSupportId} \leq \text{AxisVMMModel.LineSupports.Count}$)

LineSectionId section index ($0 < \text{SectionId} \leq \text{AxisVMMModel.LineSupports.SectionCount}[\text{LineSupportId}, \text{AnalysisType}]$)

Force force results

PosX position of SectionId in m according to the local x direction

Combination name of the load case

Retrieves line support forces at a section of a line element according to the LoadCombinationId (and LoadLevelOrTimeStep) property. Returns LineSupportId or an error code ([errDatabaseNotReady](#) or see [EForcesError](#)).

long **EnvelopeLineSupportForce** ([in] long **LineSupportId**, [in] long **LineSectionId**,
[i/o] [RLineSupportForceValues](#) **Force**, [out] double **PosX**, [out] BSTR **Combination**)

LineSupportId line support index ($0 < \text{LineSupportId} \leq \text{AxisVMMModel.LineSupports.Count}$)

LineSectionId section index ($0 < \text{SectionId} \leq \text{AxisVMMModel.LineSupports.SectionCount}[\text{LineSupportId}, \text{AnalysisType}]$)

Force force results

PosX position of SectionId in m according to the local x direction

Combination load case or combination in which Component has its minimum or maximum

Retrieves envelope line support forces at a section of a line element. Envelope is identified by LineSupportForceComponent, MinMaxType and EnvelopeUID properties. Force contains the result of the load case or combination in which Component is maximal or minimal. Returns LineSupportId or an error code ([errDatabaseNotReady](#) or see [EForcesError](#)).

long **EnvelopeLineSupportForce2** ([in] long **LineSupportId**, [in] long **LineSectionId**,
[i/o] [RLineSupportForceValues](#) **Force**, [out] double **PosX**, [out] long **LoadCaseOrCombinationId**,
[out] long **LoadLevel**)

LineSupportId line support index ($0 < \text{LineSupportId} \leq \text{AxisVMMModel.LineSupports.Count}$)

LineSectionId section index ($0 < \text{SectionId} \leq \text{AxisVMMModel.LineSupports.SectionCount}[\text{LineSupportId}, \text{AnalysisType}]$)

Force force results

PosX position of SectionId in m according to the local x direction

LoadCaseOrCombinationId load case or load combination index, if index is >

[IAxisVMLoadcases](#).count then Load combination index =

LoadCaseOrCombinationId – [IAxisVMLoadcases](#).count

LoadLevel load level

Retrieves envelope line support forces at a section of a line element. Envelope is identified by LineSupportForceComponent, MinMaxType and EnvelopeUID properties. Force contains the result of the load case or combination in which Component is maximal or minimal. Returns LineSupportId or an error code ([errDatabaseNotReady](#) or see [EForcesError](#)).

long **CriticalLineSupportForce** ([in] long **LineSupportId**, [in] long **LineSectionId**,
[i/o] [RLineSupportForceValues](#) **Force**, [out] double **PosX**, [out] BSTR **Combination**)

LineSupportId	<i>line support index ($0 < LineSupportId \leq AxisVMMModel.LineSupports.Count$)</i>
LineSectionId	<i>section index ($0 < SectionId \leq AxisVMMModel.LineSupports.SectionCount[LineSupportId, AnalysisType]$)</i>
Force	<i>force results</i>
PosX	<i>position of SectionId in m according to the local x direction</i>
Combination	<i>critical combination in which Component has its minimum or maximum</i>

Retrieves critical line support forces at a section of a line element. Critical combination is identified by Component and MinMaxType properties (e.g. Rx max). Force contains the result of the critical combination in which Component is maximal or minimal. Returns LineSupportId or an error code ([errDatabaseNotReady](#) or see [EForcesError](#)).

long **CriticalLineSupportForce2** ([in] long **LineSupportId**, [in] long **LineSectionId**,
[i/o] [RLineSupportForceValues](#) **Force**, [out] double **PosX**,
[out] [ECombinationType](#) **CriticalCombinationType**, [out] SAFEARRAY(double) **Factors**,
[out] SAFEARRAY(long) **LoadCaselds**)

CriticalCombinationType	<i>combination type corresponding to critical load combination</i>
Factors	<i>load factors of the critical load combination</i>
LoadCaselds	<i>load case indexes of the critical load combination</i>

Similar to CriticalLineSupportForce with more parameters described above.

Single ELEMENT reader functions

long **LineSupportForcesByLoadCaseld** ([in] long **LineSupportId**,
[i/o] SAFEARRAY([RLineSupportForceValues](#)) **Forces**, [out] SAFEARRAY(double) **PosX**)

LineSupportId	<i>line support index ($0 < LineSupportId \leq AxisVMMModel.LineSupports.Count$)</i>
Forces	<i>array of line support forces Length of the array is AxisVMMModel.LineSupports.SectionCount[LineSupportId, AnalysisType]</i>
PosX	<i>array of cross-section positions in m according to the local x direction - Length of the array is AxisVMMModel.LineSupports.SectionCount[LineSupportId, AnalysisType]</i>

Retrieves line support forces for each cross-section of a given element according to the LoadCaseld (and LoadLevelOrTimeStep) property. Returns the total number of cross-sections or an error code ([errDatabaseNotReady](#) or see [EForcesError](#)).

long **LineSupportForcesByLoadCombinationId** ([in] long **LineSupportId**,
[i/o] SAFEARRAY([RLineSupportForceValues](#)) **Forces**, [out] SAFEARRAY(double) **PosX**)

LineSupportId	<i>line support index ($0 < LineSupportId \leq AxisVMMModel.LineSupports.Count$)</i>
Forces	<i>array of line support forces Length of the array is AxisVMMModel.LineSupports.SectionCount[LineSupportId, AnalysisType]</i>
PosX	<i>array of cross-section positions in m according to the local x direction Length of the array is SectionCount[LineId]</i>

Retrieves line support forces for each cross-section of a given element according to the LoadCombinationId (and LoadLevelOrTimeStep) property. Returns the total number of cross-sections or an error code ([errDatabaseNotReady](#) or see [EForcesError](#)).

long	EnvelopeLineSupportForces ([in] long LineSupportId , [i/o] SAFEARRAY(RLineSupportForceValues) Forces , [out] SAFEARRAY(double) PosX) LineSupportId line support index ($0 < LineSupportId \leq AxisVMMModel.LineSupports.Count$) Forces array of line support forces <i>Length of the array is AxisVMMModel.LineSupports.SectionCount[LineSupportId,AnalysisType]</i> PosX array of cross-section positions in m according to the local x direction <i>Length of the array is SectionCount[Linelid]</i>
	<p><i>Retrieves envelope line support forces for each cross-section of a given element Envelope is identified by LineSupportForceComponent, MinMaxType and EnvelopeUID properties. Forces array contains the result of the load case or combination in which Component is maximal or minimal. Load case or combination in which Component has its minimum or maximum can be read using the respective single element/single cross-section reader function.</i></p> <p><i>Returns the total number of cross-sections or an error code (errDatabaseNotReady or see EForcesError).</i></p>
long	CriticalLineSupportForces ([in] long LineSupportId , [i/o] SAFEARRAY(RLineSupportForceValues) Forces , [out] SAFEARRAY(double) PosX) LineSupportId line support index ($0 < LineSupportId \leq AxisVMMModel.LineSupports.Count$) Forces array of line support forces <i>Length of the array is AxisVMMModel.LineSupports.SectionCount[LineSupportId,AnalysisType]</i> PosX array of cross-section positions in m according to the local x direction <i>Length of the array is SectionCount[Linelid]</i>
	<p><i>Retrieves critical line support forces in each cross-section of a given element. Critical combination is identified by Component and MinMaxType properties (e.g. Nx max). Forces array contains the result of the critical combination in which Component is maximal or minimal. Critical combination in which Component has its minimum or maximum can be read using the respective single element/single cross-section reader function.</i></p> <p><i>Returns the total number of cross-sections or an error code (errDatabaseNotReady or see EForcesError).</i></p>

Multiple element reader functions

long	AllLineSupportForcesByLoadCaselid ([out] SAFEARRAY(long) SectionCounts , [i/o] SAFEARRAY(RLineSupportForceValues) Forces , [out] SAFEARRAY(double) PosX) SectionCounts array containing the section counts of line supports. <i>Length of the array = AxisVMMModel.LineSupports.Count</i> Forces array of line support forces. <i>For each line support it contains results for SectionCount sections consecutively.</i> <i>Length of the array is the sum of the SectionCounts array elements</i> PosX array of cross-section positions in m according to the local x direction of each line element <i>Length of the array is the sum of the SectionCounts array elements</i>
	<p><i>Retrieves forces of all line supports and cross-sections consecutively according to the LoadCaselid (and LoadLevelOrTimeStep) property. Returns the total number of records in the array or an error code (errDatabaseNotReady or see EForcesError).</i></p>
long	AllLineSupportForcesByLoadCombinationId ([out] SAFEARRAY(long) SectionCounts , [i/o] SAFEARRAY(RLineSupportForceValues) Forces , [out] SAFEARRAY(double) PosX) SectionCounts array containing the section counts of line supports. <i>Length of the array = AxisVMMModel.LineSupports.Count</i> Forces array of line support forces. <i>For each line support it contains results for SectionCount sections consecutively.</i> <i>Length of the array is the sum of the SectionCounts array elements</i> PosX array of cross-section positions in m according to the local x direction of each line element <i>Length of the array is the sum of the SectionCounts array elements</i>
	<p><i>Retrieves forces of all line supports and cross-sections consecutively according to the LoadCombinationId (and LoadLevelOrTimeStep) property. Returns the total number of records in the array or an error code (errDatabaseNotReady or see EForcesError).</i></p>

long	AllEnvelopeLineSupportForces ([out] SAFEARRAY(long) SectionCounts , [i/o] SAFEARRAY(RLineSupportForceValues) Forces , [out] SAFEARRAY(double) PosX)
	SectionCounts array containing the section counts of line supports. Length of the array = AxisVMModel.LineSupports.Count
	Forces array of envelope line support forces. For each line support it contains results for SectionCount sections consecutively. Length of the array is the sum of the SectionCounts array elements
	PosX array of cross-section positions in m according to the local x direction of each line element Length of the array is the sum of the SectionCounts array elements
	Retrieves envelope forces of all line supports. Envelope is identified by LineSupportForceComponent, MinMaxType and EnvelopeUID properties. Forces array contains the result of the load case or combination in which Component is maximal or minimal. Load case or combination in which Component has its minimum or maximum can be read using the respective single location reader function. Returns the total number of cross-sections or an error code (errDatabaseNotReady or see EForcesError).
long	AllCriticalLineSupportForces ([out] SAFEARRAY(long) SectionCounts , [i/o] SAFEARRAY(RLineSupportForceValues) Forces , [out] SAFEARRAY(double) PosX)
	SectionCounts array containing the section counts of line supports. Length of the array = AxisVMModel.LineSupports.Count
	Forces array of critical line support forces. For each line support it contains results for SectionCount sections consecutively. Length of the array is the sum of the SectionCounts array elements
	PosX array of cross-section positions in m according to the local x direction of each line element. Length of the array is the sum of the SectionCounts array elements
	Retrieves critical forces of all line supports. Critical combination is identified by Component and MinMaxType properties (e.g. Rx max). Forces array contains the result of the critical combination in which Component is maximal or minimal. Critical combination in which Component has its minimum or maximum can be read using the respective single location reader function. Returns the total number of cross-sections or an error code (errDatabaseNotReady or see EForcesError).

Multiple BLOCK reader functions

long	LineSupportForcesForResultBlocks ([in] long LineSupportId , [in] long LineSectionId , [i/o] SAFEARRAY(RResultBlockInfo) ResultBlockInfo , [i/o] SAFEARRAY(RLineSupportForceValues) Forces , [out] SAFEARRAY(double) PosX)
	LineSupportId line support index ($0 < \text{LineSupportId} \leq \text{AxisVMLineSupports.Count}$)
	LineSectionId section index ($0 < \text{SectionId} \leq \text{AxisVMLineSupports.SectionCount}[\text{LineSupportId}, \text{AnalysisType}]$)
	ResultBlockInfo array of description records for result blocks
	Forces force results for all result blocks
	PosX array of cross-section positions in m according to the local x direction of each line element
	Retrieves forces at the given line support and cross-section in all result blocks consecutively. The number of result blocks depends on the AnalysisType property. Returns the common length of ResultBlockInfo, Forces and PosX arrays or an error code (errDatabaseNotReady or see EForcesError).

Member Support Forces

long **GetMembersSupportForcesByLoadCaseId** ([in] long **MembersSupportId**,
[in] long **LoadCaseId**, [in] long **LoadLevelOrTimeStep**, [in] [EAnalysisType](#) **AnalysisType**,
[out] SAFEARRAY([RLineSupportForceValues](#)) **Forces**, [out] SAFEARRAY(double) **PosX**)

MembersSupportId member support index
($0 < \text{MembersSupportId} \leq \text{AxisVMModel.MembersSupports.Count}$)
LoadCaseId load case index
LoadLevelOrTimeStep load level or time step, according to the type of analysis. See
[LoadLevelOrModeShapeOrTimeStep](#) parameter.
AnalysisType analysis type
Forces array of member forces for the member
PosX array of cross-section positions in m according to the local x direction

Retrieves support forces (reactions) for each section along member (subject to meshing) of a given element according to the LoadCaseId and LoadLevelOrTimeStep. Returns the total number of force values (Forces array length) or an error code ([errDatabaseNotReady](#) or see [EForcesError](#)).

long **GetMembersSupportForcesByLoadCombinationId** ([in] long **MembersSupportId**,
[in] long **LoadCombinationId**, [in] long **LoadLevelOrTimeStep**,
[in] [EAnalysisType](#) **AnalysisType**, [out] SAFEARRAY([RLineSupportForceValues](#)) **Forces**,
[out] SAFEARRAY(double) **PosX**)

MembersSupportId member support index
($0 < \text{MembersSupportId} \leq \text{AxisVMModel.MembersSupports.Count}$)
LoadCombinationId load combination index
LoadLevelOrTimeStep load level or time step, according to the type of analysis. See
[LoadLevelOrModeShapeOrTimeStep](#) parameter.
AnalysisType analysis type
Forces array of member support forces for the member
PosX array of cross-section positions in m according to the local x direction

Retrieves support forces (reactions) for each section along member (subject to meshing) of a given element according to the LoadCombinationId and LoadLevelOrTimeStep. Returns the total number of force values (Forces array length) or an error code ([errDatabaseNotReady](#) or see [EForcesError](#)).

long **GetEnvelopeMembersSupportForces** ([in] long **MembersSupportId**,
[in] [EMinMaxType](#) **MinMaxType**, [in] [EAnalysisType](#) **AnalysisType**,
[in] [ELineSupportForce](#) **Component**, [out] SAFEARRAY([RLineSupportForceValues](#)) **Forces**,
[out] SAFEARRAY(double) **PosX**)

MembersSupportId member support index
($0 < \text{MembersSupportId} \leq \text{AxisVMModel.MembersSupports.Count}$)
MinMaxType only min or max allowed
Component support force component
AnalysisType analysis type
Forces array of member support forces for the member
PosX array of cross-section positions in m according to the local x direction

Retrieves support forces (reactions) for each section along member (subject to meshing) of a given element according to MinMaxType and Component parameters and EnvelopeID property of the interface. Returns the total number of force values (Forces array length) or an error code ([errDatabaseNotReady](#) or see [EForcesError](#)).

long **GetCriticalMembersSupportForces** ([in] long **MembersSupportId**,
 [in] [EMinMaxType](#) **MinMaxType**, [in] [ECombinationType](#) **CombinationType**,
 [in] [EAnalysisType](#) **AnalysisType**, [in] [ELineSupportForce](#) **Component**,
 [out] SAFEARRAY([RLineSupportForceValues](#)) **Forces**,
 [out] SAFEARRAY(double) **PosX**)

MembersSupportId	<i>member support index ($0 < \text{MembersSupportId} \leq \text{AxisVMModel.MembersSupports.Count}$)</i>
MinMaxType	<i>only min or max allowed</i>
CombinationType	<i>load combination type allowed by national design code</i>
AnalysisType	<i>analysis type</i>
Component	<i>support force component</i>
Forces	<i>array of member support forces for the member</i>
PosX	<i>array of cross-section positions in m according to the local x direction</i>

Retrieves support forces (reactions) for each section along member (subject to meshing) of a given element according to MinMaxType, Component and CombinationType parameters. Returns the total number of force values (Forces array length) or an error code ([errDatabaseNotReady](#) or see [EForcesError](#)).

Nodal Support Forces

Single Element reader functions

long **NodalSupportForceByLoadCaseId** ([in] long **NodalSupportId**,
 [i/o] [RNodalSupportForceValues](#) **Force**, [out] BSTR **Combination**)

NodalSupportId	<i>nodal support index ($0 < \text{NodalSupportId} \leq \text{AxisVMNodalSupports.Count}$)</i>
Force	<i>force results</i>
Combination	<i>name of the load case</i>

Retrieves forces of a nodal support according to the LoadCaseId (and LoadLevelOrTimeStep) property. Returns NodalSupportId or an error code ([errDatabaseNotReady](#) or see [EForcesError](#)).

long **NodalSupportForceByLoadCombinationId** ([in] long **NodalSupportId**,
 [i/o] [RNodalSupportForceValues](#) **Force**, [out] BSTR **Combination**)

NodalSupportId	<i>nodal support index ($0 < \text{NodalSupportId} \leq \text{AxisVMNodalSupports.Count}$)</i>
Force	<i>force results</i>
Combination	<i>name of the load case</i>

Retrieves forces of a nodal support according to the LoadCombinationId (and LoadLevelOrTimeStep) property. Returns NodalSupportId or an error code ([errDatabaseNotReady](#) or see [EForcesError](#)).

long **EnvelopeNodalSupportForce** ([in] long **NodalSupportId**,
 [i/o] [RNodalSupportForceValues](#) **Force**, [out] BSTR **Combination**)

NodalSupportId	<i>nodal support index ($0 < \text{NodalSupportId} \leq \text{AxisVMNodalSupports.Count}$)</i>
Force	<i>force results</i>
Combination	<i>load case or combination in which Component has its minimum or maximum</i>

Retrieves envelope forces of a nodal support. Envelope is identified by NodalSupportForceComponent, MinMaxType and EnvelopeUID properties. Force contains the result of the load case or combination in which Component is maximal or minimal. Returns NodalSupportId or an error code ([errDatabaseNotReady](#) or see [EForcesError](#)).

long	EnvelopeNodalSupportForce2 ([in] long NodalSupportId , [i/o] RNodalSupportForceValues Force , [out] long LoadCaseOrCombinationId , [out] long LoadLevel)
	NodalSupportId <i>nodal support index ($0 < \text{NodalSupportId} \leq \text{AxisVMNodalSupports.Count}$)</i>
	Force <i>force results</i>
	LoadCaseOrCombinationId <i>load case or load combination index, if index is > IAxisVMLoadcases.count then Load combination index = $\text{LoadCaseOrCombinationId} - \text{IAxisVMLoadcases.count}$</i>
	LoadLevel <i>load level</i>
	<i>Retrieves envelope forces of a nodal support. Envelope is identified by NodalSupportForceComponent, MinMaxType and EnvelopeUID properties. Force contains the result of the load case or combination in which Component is maximal or minimal. Returns NodalSupportId or an error code (errDatabaseNotReady or see EForcesError).</i>
long	CriticalNodalSupportForce ([in] long NodalSupportId , [i/o] RNodalSupportForceValues Force , [out] BSTR Combination)
	NodalSupportId <i>nodal support index ($0 < \text{NodalSupportId} \leq \text{AxisVMNodalSupports.Count}$)</i>
	Force <i>force results</i>
	Combination <i>critical combination in which Component has its minimum or maximum</i>
	<i>Retrieves critical forces of a nodal support. Critical combination is identified by Component and MinMaxType properties (e.g. Rx max). Force contains the result of the critical combination in which Component is maximal or minimal. Returns NodalSupportId or an error code (errDatabaseNotReady or see EForcesError).</i>
long	CriticalNodalSupportForce2 ([in] long NodalSupportId , [i/o] RNodalSupportForceValues Force , [out] ECombinationType CriticalCombinationType , [out] SAFEARRAY(double) Factors , [out] SAFEARRAY(long) LoadCaselds)
	CriticalCombinationType <i>combination type corresponding to critical load combination</i>
	Factors <i>load factors of the critical load combination</i>
	LoadCaselds <i>load case indexes of the critical load combination</i>
	<i>Similar to CriticalNodalSupportForce with more parameters described above.</i>

Multiple element reader functions

long	AllNodalSupportForcesByLoadCaseld (
	[i/o] SAFEARRAY(RNodalSupportForceValues) Forces)
	Forces <i>array of nodal support forces. Length of the array is AxisVMNodalSupports.Count.</i>
	<i>Retrieves forces of all nodal supports consecutively according to the LoadCaseld (and LoadLevelOrTimeStep) property. Returns the total number of records in the array or an error code (errDatabaseNotReady or see EForcesError).</i>
long	AllNodalSupportForcesByLoadCombinationId (
	[i/o] SAFEARRAY(RNodalSupportForceValues) Forces)
	Forces <i>array of nodal support forces. Length of the array is AxisVMNodalSupports.Count.</i>
	<i>Retrieves forces of all nodal supports consecutively according to the LoadCombinationId (and LoadLevelOrTimeStep) property. Returns the total number of records in the array or an error code (errDatabaseNotReady or see EForcesError).</i>

long **AllEnvelopeNodalSupportForces** ([i/o] SAFEARRAY([RNodalSupportForceValues](#)) **Forces**)

Forces array of nodal support forces.

Length of the array is AxisVMNodalSupports.Count.

Retrieves envelope forces of all nodal supports. Envelope is identified by NodalSupportForceComponent, MinMaxType and EnvelopeUID properties. Forces array contains the result of the load case or combination in which Component is maximal or minimal. Load case or combination in which Component has its minimum or maximum can be read using the respective single location reader function.

Returns the array length or an error code ([errDatabaseNotReady](#) or see [EForcesError](#)).

long **AllCriticalNodalSupportForces** ([i/o] SAFEARRAY([RNodalSupportForceValues](#)) **Forces**)

Forces array of nodal support forces.

Length of the array is AxisVMNodalSupports.Count.

Retrieves critical forces of all nodal supports. Critical combination is identified by Component and MinMaxType properties (e.g. Rx max). Forces array contains the result of the critical combination in which Component is maximal or minimal. Critical combination in which Component has its minimum or maximum can be read using the respective single location reader function.

Returns the array length or an error code ([errDatabaseNotReady](#) or see [EForcesError](#)).

Multiple BLOCK reader functions

long **NodalSupportForcesForResultBlocks** ([in] long **NodalSupportId**,

[i/o] SAFEARRAY([RResultBlockInfo](#)) **ResultBlockInfo**,

[i/o] SAFEARRAY([RLineForceValues](#)) **Forces**)

NodalSupportId nodal support index

($0 < \text{NodalSupportId} \leq \text{AxisVMNodalSupports.Count}$)

ResultBlockInfo array of description records for result blocks

Forces force results for all result blocks

Retrieves forces at the given nodal support in all result blocks consecutively. The number of result blocks depends on the AnalysisType property.

Returns the common length of ResultBlockInfo and Forces arrays or an error code ([errDatabaseNotReady](#) or see [EForcesError](#)).

Spring Forces

Single EleMENT reader functions

long **SpringForceByLoadCaseId** ([in] long **LineId**, [i/o] [RSpringForceValues](#) **Force**, [out] BSTR **Combination**)

LineId line index ($0 < \text{LineId} \leq \text{AxisVMMModel.Lines.Count}$)

Force force results

Combination name of the load case

Retrieves spring forces according to the LoadCaseId (and LoadLevelOrTimeStep) property.

Returns LineId or an error code ([errDatabaseNotReady](#) or see [EForcesError](#)).

long **SpringForceByLoadCombinationId** ([in] long **LineId**, [i/o] [RSpringForceValues](#) **Force**, [out] BSTR **Combination**)

LineId line index ($0 < \text{LineId} \leq \text{AxisVMMModel.Lines.Count}$)

Force force results

Combination name of the load case

Retrieves spring forces according to the LoadCombinationId (and LoadLevelOrTimeStep) property.

Returns LineId or an error code ([errDatabaseNotReady](#) or see [EForcesError](#)).

long	EnvelopeSpringForce ([in] long LineId , [i/o] RSpringForceValues Force , [out] BSTR Combination)
	<p style="margin-left: 20px;">LineId <i>line index ($0 < \text{LineId} \leq \text{AxisVMModel.Lines.Count}$)</i></p> <p style="margin-left: 20px;">Force <i>force results</i></p> <p style="margin-left: 20px;">Combination <i>load case or combination in which Component has its minimum or maximum</i></p>
	<p><i>Retrieves envelope spring forces. Envelope is identified by SpringtForceComponent, MinMaxType and EnvelopeUID properties. Force contains the result of the load case or combination in which Component is maximal or minimal. Returns LineId or an error code (errDatabaseNotReady or see EForcesError).</i></p>
long	EnvelopeSpringForce2 ([in] long LineId , [i/o] RSpringForceValues Force , [out] long LoadCaseOrCombinationId , [out] long LoadLevel)
	<p style="margin-left: 20px;">LineId <i>line index ($0 < \text{LineId} \leq \text{AxisVMModel.Lines.Count}$)</i></p> <p style="margin-left: 20px;">Force <i>force results</i></p> <p style="margin-left: 20px;">LoadCaseOrCombinationId <i>load case or load combination index, if index is $> \text{IAxisVMLoadcases.count}$ then Load combination index = $\text{LoadCaseOrCombinationId} - \text{IAxisVMLoadcases.count}$</i></p> <p style="margin-left: 20px;">LoadLevel <i>load level</i></p>
	<p><i>Retrieves envelope spring forces Envelope is identified by SpringtForceComponent, MinMaxType and EnvelopeUID properties. Force contains the result of the load case or combination in which Component is maximal or minimal. Returns LineId or an error code (errDatabaseNotReady or see EForcesError).</i></p>
long	CriticalSpringForce ([in] long LineId , [i/o] RSpringForceValues Force , [out] BSTR Combination)
	<p style="margin-left: 20px;">LineId <i>line index ($0 < \text{LineId} \leq \text{AxisVMModel.Lines.Count}$)</i></p> <p style="margin-left: 20px;">Force <i>force results</i></p> <p style="margin-left: 20px;">Combination <i>critical combination in which Component has its minimum or maximum</i></p>
	<p><i>Retrieves critical spring forces. Critical combination is identified by Component and MinMaxType properties (e.g. Rx max). Force contains the result of the critical combination in which Component is maximal or minimal. Returns LineId or an error code (errDatabaseNotReady or see EForcesError).</i></p>
long	CriticalSpringForce2 ([in] long LineId , [i/o] RSpringForceValues Force , [out] ECombinationType CriticalCombinationType , [out] SAFEARRAY(double) Factors , [out] SAFEARRAY(long) LoadCaselds)
	<p style="margin-left: 20px;">CriticalCombinationType <i>combination type corresponding to critical load combination</i></p> <p style="margin-left: 20px;">Factors <i>load factors of the critical load combination</i></p> <p style="margin-left: 20px;">LoadCaselds <i>load case indexes of the critical load combination</i></p>
	<p><i>Similar to CriticalSpringForce with more parameters described above.</i></p>

Multiple element reader functions

long	AllSpringForcesByLoadCaseld ([i/o] SAFEARRAY(RSpringForceValues) Forces)
	<p style="margin-left: 20px;">Forces <i>array of spring forces.</i></p> <p style="margin-left: 20px;"><i>Length of the array is the number of spring elements in the model.</i></p>
	<p><i>Retrieves all spring forces consecutively according to the LoadCaseld (and LoadLevelOrTimeStep) property. Returns the total number of records in the array or an error code (errDatabaseNotReady or see EForcesError).</i></p>

long	AllSpringForcesByLoadCombinationId ([i/o] SAFEARRAY(RSpringForceValues) Forces)
	Forces array of spring forces. <i>Length of the array is the number of spring elements in the model.</i>
<i>Retrieves all spring forces consecutively according to the LoadCombinationId (and LoadLevelOrTimeStep) property. Returns the total number of records in the array or an error code (errDatabaseNotReady or see EForcesError).</i>	
long	AllEnvelopeSpringForces ([i/o] SAFEARRAY(RSpringForceValues) Forces)
	Forces array of spring forces. <i>Length of the array is the number of spring elements in the model.</i>
<i>Retrieves all envelope spring forces. Envelope is identified by SpringForceComponent, MinMaxType and EnvelopeUID properties. Forces array contains the result of the load case or combination in which Component is maximal or minimal. Load case or combination in which Component has its minimum or maximum can be read using the respective single location reader function.</i>	
<i>Returns the array length or an error code (errDatabaseNotReady or see EForcesError).</i>	
long	AllCriticalSpringForces ([i/o] SAFEARRAY(RSpringForceValues) Forces)
	Forces array of spring forces. <i>Length of the array is the number of spring elements in the model.</i>
<i>Retrieves all critical spring forces. Critical combination is identified by Component and MinMaxType properties (e.g. Rx max). Forces array contains the result of the critical combination in which Component is maximal or minimal. Critical combination in which Component has its minimum or maximum can be read using the respective single location reader function.</i>	
<i>Returns the array length or an error code (errDatabaseNotReady or see EForcesError).</i>	

Multiple BLOCK reader functions

long	SpringForcesForResultBlocks ([in] long Lineld , [i/o] SAFEARRAY(RResultBlockInfo) ResultBlockInfo , [i/o] SAFEARRAY(RSpringForceValues) Forces)
	Lineld line index ($0 < \text{Lineld} \leq \text{AxisVMModel.Lines.Count}$)
	ResultBlockInfo array of description records for result blocks
	Forces force results for all result blocks
<i>Retrieves spring forces in all result blocks consecutively. The number of result blocks depends on the AnalysisType property.</i>	
<i>Returns the common length of ResultBlockInfo and Forces arrays or an error code (errDatabaseNotReady or see EForcesError).</i>	

Surface Forces

Single location reader functions retrieve forces at a certain point of a surface identified by **SurfaceVertexType** (svtContourPoint, svtContourLineMidPoint or svtCenterPoint) and **SurfaceVertexId**.

SurfaceVertexType	Meaning of SurfaceVertexId
svtContourPoint	node index ($0 < \text{SurfaceVertexId} \leq \text{AxisVMNodes.Count}$)
svtContourLinePoint	line index ($0 < \text{SurfaceVertexId} \leq \text{AxisVMLines.Count}$)
svtCenterPoint	-

Single element reader functions retrieve forces in all available points of a surface. If the surface element is triangular the *ContourPointCount* field of [RSurfaceForces](#) = 3, record fields *ContourPoint4Id*, *ContourLine4Id*, *sfvContourPoint4* and *sfvContourLineMidPoint4* contain no meaningful values. If the surface element is quadrilateral the *ContourPointCount* field of [RSurfaceForces](#) = 4 and each field has a meaningful value.

Single LOCATION reader functions

long **SurfaceForceByLoadCaseId** ([in] long **Surfaceld**, [in] [ESurfaceVertexType](#) **SurfaceVertexType**, [in] long **SurfaceVertexId**, [i/o] [RSurfaceForceValues](#) **Force**, [out] BSTR **Combination**)

Surfaceld surface index ($0 < \text{Surfaceld} \leq \text{AxisVMSurfaces.Count}$)
SurfaceVertexType vertex type
SurfaceVertexId vertex identifier
Force force results
Combination name of the load case

Retrieves forces at a point of a surface according to the LoadCaseId (and LoadLevelOrTimeStep) property. Returns Surfaceld or an error code ([errDatabaseNotReady](#) or see [EForcesError](#)).

long **SurfaceForceByLoadCombinationId** ([in] long **Surfaceld**, [in] [ESurfaceVertexType](#) **SurfaceVertexType**, [in] long **SurfaceVertexId**, [i/o] [RSurfaceForceValues](#) **Force**, [out] BSTR **Combination**)

Surfaceld surface index ($0 < \text{Surfaceld} \leq \text{AxisVMSurfaces.Count}$)
SurfaceVertexType vertex type
SurfaceVertexId vertex identifier
Force force results
Combination name of the load combination

Retrieves forces at a point of a surface according to the LoadCombinationId (and LoadLevelOrTimeStep) property. Returns Surfaceld or an error code ([errDatabaseNotReady](#) or see [EForcesError](#)).

long **EnvelopeSurfaceForce** ([in] long **Surfaceld**, [in] [ESurfaceVertexType](#) **SurfaceVertexType**, [in] long **SurfaceVertexId**, [i/o] [RSurfaceForceValues](#) **Force**, [out] BSTR **Combination**)

Surfaceld surface index ($0 < \text{Surfaceld} \leq \text{AxisVMSurfaces.Count}$)
SurfaceVertexType vertex type
SurfaceVertexId vertex identifier
Force force results
Combination load case or combination in which Component has its minimum or maximum

Retrieves envelope forces at one point of a surface element. Envelope is identified by SurfaceForceComponent, MinMaxType and EnvelopeUID properties. Force contains the result of the load case or combination in which Component is maximal or minimal. Returns Surfaceld or an error code ([errDatabaseNotReady](#) or see [EForcesError](#)).

long **CriticalSurfaceForce** ([in] long **Surfaceld**, [in] [ESurfaceVertexType](#) **SurfaceVertexType**,
[in] long **SurfaceVertexId**, [i/o] [RSurfaceForceValues](#) **Force**, [out] BSTR **Combination**)

Surfaceld surface index ($0 < \text{Surfaceld} \leq \text{AxisVMSurfaces.Count}$)
SurfaceVertexType vertex type
SurfaceVertexId vertex identifier
Force force results
Combination critical combination in which Component has its minimum or maximum

Retrieves critical forces at one point of a surface element. Critical combination is identified by Component and MinMaxType properties (e.g. Nxy max). Force contains the result of the critical combination in which Component is maximal or minimal. Returns Surfaceld or an error code ([errDatabaseNotReady](#) or see [EForcesError](#)).

long **CriticalSurfaceForce2** ([in] long **Surfaceld**, [in] [ESurfaceVertexType](#) **SurfaceVertexType**,
[in] long **SurfaceVertexId**, [i/o] [RSurfaceForceValues](#) **Force**,
[out] [ECombinationType](#) **CriticalCombinationType**, [out] SAFEARRAY(double) **Factors**,
[out] SAFEARRAY(long) **LoadCaselds**)

CriticalCombinationType combination type corresponding to critical load combination
Factors load factors of the critical load combination
LoadCaselds load case indexes of the critical load combination

Similar to CriticalSurfaceForce with more parameters described above.

Single ELEMENT reader functions

long **SurfaceForcesByLoadCaseld** ([in] long **Surfaceld**, [i/o] [RSurfaceForces](#) **Forces**)

Surfaceld surface index ($0 < \text{Surfaceld} \leq \text{AxisVMMModel.Surfaces.Count}$)
Forces surface forces on the element

Retrieves forces in all available points of a surface element according to the LoadCaseld (and LoadLevelOrTimeStep) property. Returns Surfaceld or an error code ([errDatabaseNotReady](#) or see [EForcesError](#)).

long **SurfaceForcesByLoadCombinationId** ([in] long **Surfaceld**, [i/o] [RSurfaceForces](#) **Forces**)

Surfaceld surface index ($0 < \text{Surfaceld} \leq \text{AxisVMMModel.Surfaces.Count}$)
Forces surface forces on the element

Retrieves forces in all available points of a surface element according to the LoadCombinationId (and LoadLevelOrTimeStep) property. Returns Surfaceld or an error code ([errDatabaseNotReady](#) or see [EForcesError](#)).

long **EnvelopeSurfaceForces** ([in] long **Surfaceld**, [i/o] [RSurfaceForces](#) **Forces**)

Surfaceld surface index ($0 < \text{Surfaceld} \leq \text{AxisVMMModel.Surfaces.Count}$)
Forces surface forces on the element

Retrieves envelope forces in all available points of a surface element. Envelope is identified by SurfaceForceComponent, MinMaxType and EnvelopeUID properties. At each point Forces contain the result of the load case or combination in which Component is maximal or minimal. Returns Surfaceld or an error code ([errDatabaseNotReady](#) or see [EForcesError](#)).

long **CriticalSurfaceForces** ([in] long **Surfaceld**, [i/o] [RSurfaceForces](#) **Forces**)

Surfaceld surface index ($0 < \text{Surfaceld} \leq \text{AxisVMMModel.Surfaces.Count}$)
Forces surface forces on the element

Retrieves critical forces in all available points of a surface element. Critical combination is identified by Component and MinMaxType properties (e.g. Nxy max). At each point Forces contain the result of the critical combination in which Component is maximal or minimal. Returns Surfaceld or an error code ([errDatabaseNotReady](#) or see [EForcesError](#)).

Multiple element reader functions

long **AllSurfaceForcesByLoadCaseId** ([i/o] SAFEARRAY([RSurfaceForces](#)) **Forces**)

Forces array of surface forces.

Length of the array is AxisVMMModel.Surfaces.Count

Retrieves forces on all surface elements in an array according to the LoadCaseId (and LoadLevelOrTimeStep) property. Returns the total number of records in the array or an error code ([errDatabaseNotReady](#) or see [EForcesError](#)).

long **AllSurfaceForcesByLoadCombinationId** ([i/o] SAFEARRAY([RSurfaceForces](#)) **Forces**)

Forces array of surface forces.

Length of the array is AxisVMMModel.Surfaces.Count

Retrieves forces on all surface elements in an array according to the LoadCombinationId (and LoadLevelOrTimeStep) property. Returns the total number of records in the array or an error code ([errDatabaseNotReady](#) or see [EForcesError](#)).

long **AllEnvelopeSurfaceForces** ([i/o] SAFEARRAY([RSurfaceForces](#)) **Forces**)

Forces array of surface forces.

Length of the array is AxisVMMModel.Surfaces.Count

Retrieves envelope forces on all surface elements. Envelope is identified by SurfaceForceComponent, MinMaxType and EnvelopeUID properties. Forces array contains the result of the load case or combination in which Component is maximal or minimal. Load case or combination in which Component has its minimum or maximum can be read using the respective single location reader function.

Returns the total number of records in the array or an error code ([errDatabaseNotReady](#) or see [EForcesError](#)).

long **AllCriticalSurfaceForces** ([i/o] SAFEARRAY([RSurfaceForces](#)) **Forces**)

Forces array of surface forces.

Length of the array is AxisVMMModel.Surfaces.Count

Retrieves critical forces on all line elements. Critical combination is identified by Component and MinMaxType properties (e.g. Nxy max). Forces array contains the result of the load case or combination in which Component is maximal or minimal. Load case or combination in which Component has its minimum or maximum can be read using the respective single location reader function.

Returns the total number of records in the array or an error code ([errDatabaseNotReady](#) or see [EForcesError](#)).

Multiple BLOCK reader functions

long **SurfaceForcesForResultBlocks** ([in] long **Surfaceld**, [i/o] SAFEARRAY([RResultBlockInfo](#))

ResultBlockInfo,

[i/o] SAFEARRAY([RSurfaceForces](#)) **Forces**)

Surfaceld surface index ($0 < Surfaceld \leq \text{AxisVMMModel.Surfaces.Count}$)

ResultBlockInfo array of description records for result blocks

Forces force results for all result blocks

Retrieves forces on the given surface element in all result blocks consecutively. The number of result blocks depends on the AnalysisType property.

Returns the common length of ResultBlockInfo and Forces arrays or an error code ([errDatabaseNotReady](#) or see [EForcesError](#)).

long **SurfaceForceValuesForResultBlocks** ([in] long **Surfaceld**, [in] [ESurfaceVertexType](#)
SurfaceVertexType, [in] long **SurfaceVertexId**, [i/o] SAFEARRAY([RResultBlockInfo](#))
ResultBlockInfo,
[i/o] SAFEARRAY([RSurfaceForceValues](#)) **Forces**)

Surfaceld surface index ($0 < \text{Surfaceld} \leq \text{AxisVMModel.Surfaces.Count}$)

SurfaceVertexType vertex type

SurfaceVertexId vertex identifier

ResultBlockInfo array of description records for result blocks

Forces force results for all result blocks

Retrieves forces at a given point of a given surface element in all result blocks consecutively. The number of result blocks depends on the *AnalysisType* property.

Returns the common length of *ResultBlockInfo* and *Forces* arrays or an error code ([errDatabaseNotReady](#) or see [EForcesError](#)).

Surface Support Forces

Single LOCATION reader functions

long **SurfaceSupportForceByLoadCaseld** ([in] long **SurfaceSupportId**,
[in] [ESurfaceVertexType](#) SurfaceVertexType, [in] long **SurfaceVertexId**,
[i/o] [RSurfaceSupportForceValues](#) **Force**, [out] BSTR **Combination**)

SurfaceSupportId surface support index
($0 < \text{SurfaceSupportId} \leq \text{AxisVMSurfaceSupports.Count}$)

SurfaceVertexType vertex type

SurfaceVertexId vertex identifier

Force force results

Combination name of the load case

Retrieves surface support forces at a point of a surface according to the (and *LoadLevelOrTimeStep*) property. Returns *SurfaceSupportId* or an error code ([errDatabaseNotReady](#) or see [EForcesError](#)).

long **SurfaceSupportForceByLoadCombinationId** ([in] long **SurfaceSupportId**,
[in] [ESurfaceVertexType](#) SurfaceVertexType, [in] long **SurfaceVertexId**,
[i/o] [RSurfaceSupportForceValues](#) **Force**, [out] BSTR **Combination**)

SurfaceSupportId surface support index
($0 < \text{SurfaceSupportId} \leq \text{AxisVMSurfaceSupports.Count}$)

SurfaceVertexType vertex type

SurfaceVertexId vertex identifier

Force force results

Combination name of the load combination

Retrieves surface support forces at a point of a surface according to the *LoadCombinationId* (and *LoadLevelOrTimeStep*) property. Returns *SurfaceSupportId* or an error code ([errDatabaseNotReady](#) or see [EForcesError](#)).

long **EnvelopeSurfaceSupportForce** ([in] long **SurfaceSupportId**,
[in] [ESurfaceVertexType](#) SurfaceVertexType, [in] long **SurfaceVertexId**,
[i/o] [RSurfaceSupportForceValues](#) **Force**, [out] BSTR **Combination**)

SurfaceSupportId surface support index ($0 < \text{SurfaceSupportId} \leq \text{AxisVMSurfaceSupports.Count}$)

SurfaceVertexType vertex type

SurfaceVertexId vertex identifier

Force force results

Combination load case or combination in which Component has its minimum or maximum

Retrieves envelope surface support forces at one point of a surface element. Envelope is identified by *SurfaceSupportForceComponent*, *MinMaxType* and *EnvelopeUID* properties. Force contains the result of the load case or combination in which Component is maximal or minimal. Returns *SurfaceSupportId* or an error code ([errDatabaseNotReady](#) or see [EForcesError](#)).

long	EnvelopeSurfaceSupportForce2 ([in] long SurfaceSupportId , [in] ESurfaceVertexType SurfaceVertexType , [in] long SurfaceVertexId , [i/o] RSurfaceSupportForceValues Force , [out] long LoadCaseOrCombinationId , [out] long LoadLevel)
	SurfaceSupportId surface support index ($0 < \text{SurfaceSupportId} \leq \text{AxisVMSurfaceSupports.Count}$)
	SurfaceVertexType vertex type
	SurfaceVertexId vertex identifier
	Force force results
	LoadCaseOrCombinationId load case or load combination index, if index is $> \text{IAxisVMLoadcases.Count}$ then Load combination index = $\text{LoadCaseOrCombinationId} - \text{IAxisVMLoadcases.Count}$
	LoadLevel load level
	<i>Retrieves envelope surface support forces at one point of a surface element. Envelope is identified by SurfaceSupportForceComponent, MinMaxType and EnvelopeUID properties. Force contains the result of the load case or combination in which Component is maximal or minimal. Returns SurfaceSupportId or an error code (errDatabaseNotReady or see EForcesError).</i>
long	CriticalSurfaceSupportForce ([in] long SurfaceSupportId , [in] ESurfaceVertexType SurfaceVertexType , [in] long SurfaceVertexId , [i/o] RSurfaceSupportForceValues Force , [out] BSTR Combination)
	SurfaceSupportId surface support index ($0 < \text{SurfaceSupportId} \leq \text{AxisVMSurfaceSupports.Count}$)
	SurfaceVertexType vertex type
	SurfaceVertexId vertex identifier
	Force force results
	Combination critical combination in which Component has its minimum or maximum
	<i>Retrieves critical surface support forces at one point of a surface element. Critical combination is identified by Component and MinMaxType properties (e.g. Rz max). Force contains the result of the critical combination in which Component is maximal or minimal. Returns SurfaceSupportId or an error code (errDatabaseNotReady or see EForcesError).</i>
long	CriticalSurfaceSupportForce2 ([in] long SurfaceSupportId , [in] ESurfaceVertexType SurfaceVertexType , [in] long SurfaceVertexId , [i/o] RSurfaceSupportForceValues Force , [out] ECombinationType CriticalCombinationType , [out] SAFEARRAY(double) Factors , [out] SAFEARRAY(long) LoadCaselds)
	CriticalCombinationType combination type corresponding to critical load combination
	Factors load factors of the critical load combination
	LoadCaselds load case indexes of the critical load combination
	<i>Similar to CriticalSurfaceSupportForce with more parameters described above.</i>

Single ELEMENT reader functions

long	SurfaceSupportForcesByLoadCaseld ([in] long SurfaceSupportId , [i/o] RSurfaceSupportForces Forces)
	SurfaceSupportId surface support index ($0 < \text{SurfaceSupportId} \leq \text{AxisVMSurfaceSupports.Count}$)
	Forces surface support forces on the element
	<i>Retrieves surface support forces in all available points of a surface element according to the LoadCaseld (and LoadLevelOrTimeStep) property. Returns SurfaceSupportId or an error code (errDatabaseNotReady or see EForcesError).</i>

long	SurfaceSupportForcesByLoadCombinationId ([in] long SurfaceSupportId , [i/o] RSurfaceSupportForces Forces) SurfaceSupportId surface support index ($0 < SurfaceSupportId \leq AxisVMSurfaceSupports.Count$) Forces surface support forces on the element <i>Retrieves surface support forces in all available points of a surface element according to the LoadCombinationId (and LoadLevelOrTimeStep) property. Returns SurfaceSupportId or an error code (errDatabaseNotReady or see EForcesError).</i>
long	EnvelopeSupportSurfaceForces ([in] long SurfaceSupportId , [i/o] RSurfaceSupportForces Forces) SurfaceSupportId surface support index ($0 < SurfaceSupportId \leq AxisVMSurfaceSupports.Count$) Forces surface support forces on the element <i>Retrieves envelope surface support forces in all available points of a surface element. Envelope is identified by SurfaceSupportForceComponent, MinMaxType and EnvelopeUID properties. At each point Forces contain the result of the load case or combination in which Component is maximal or minimal. Returns SurfaceSupportId or an error code (errDatabaseNotReady or see EForcesError).</i>
long	CriticalSurfaceSupportForces ([in] long SurfaceSupportId , [i/o] RSurfaceSupportForces Forces) SurfaceSupportId surface support index ($0 < SurfaceSupportId \leq AxisVMSurfaceSupports.Count$) Forces surface support forces on the element <i>Retrieves critical surface support forces in all available points of a surface element. Critical combination is identified by Component and MinMaxType properties (e.g. Rz max). At each point Forces contain the result of the critical combination in which Component is maximal or minimal. Returns SurfaceSupportId or an error code (errDatabaseNotReady or see EForcesError).</i>

Multiple element reader functions

long	AllSurfaceSupportForcesByLoadCasId ([i/o] SAFEARRAY(RSurfaceSupportForces Forces)) Forces array of surface support forces. <i>Length of the array is AxisVMMModel.SurfaceSupports.Count</i> <i>Retrieves surface support forces on all surface elements in an array according to the LoadCasId (and LoadLevelOrTimeStep) property. Returns the total number of records in the array or an error code (errDatabaseNotReady or see EForcesError).</i>
long	AllSurfaceSupportForcesByLoadCombinationId ([i/o] SAFEARRAY(RSurfaceSupportForces Forces)) Forces array of surface support forces. <i>Length of the array is AxisVMMModel.SurfaceSupports.Count</i> <i>Retrieves surface support forces on all surface elements in an array according to the LoadCombinationId (and LoadLevelOrTimeStep) property. Returns the total number of records in the array or an error code (errDatabaseNotReady or see EForcesError).</i>
long	AllEnvelopeSurfaceSupportForces ([i/o] SAFEARRAY(RSurfaceSupportForces Forces)) Forces array of surface support forces. <i>Length of the array is AxisVMMModel.SurfaceSupports.Count</i> <i>Retrieves envelope surface support forces on all surface elements. Envelope is identified by SurfaceSupportForceComponent, MinMaxType and EnvelopeUID properties. Forces array contains the result of the load case or combination in which Component is maximal or minimal. Load case or combination in which Component has its minimum or maximum can be read using the respective single location reader function.</i> <i>Returns the total number of records in the array or an error code (errDatabaseNotReady or see EForcesError).</i>

long **AllCriticalSurfaceSupportForces** ([*i/o*] SAFEARRAY([RSurfaceSupportForces](#)) **Forces**)

Forces array of surface support forces.

Length of the array is AxisVMModel.SurfaceSupports.Count

Retrieves critical surface support forces on all line elements. Critical combination is identified by Component and MinMaxType properties (e.g. Rz max). Forces array contains the result of the load case or combination in which Component is maximal or minimal. Load case or combination in which Component has its minimum or maximum can be read using the respective single location reader function.

Returns the total number of records in the array or an error code ([errDatabaseNotReady](#) or see [EForcesError](#)).

Multiple BLOCK reader functions

long **SurfaceSupportForcesForResultBlocks** ([*in*] long **SurfaceSupportId**,

[*i/o*] SAFEARRAY([RResultBlockInfo](#)) **ResultBlockInfo**,

[*i/o*] SAFEARRAY([RSurfaceSupportForces](#)) **Forces**)

SurfaceSupportId surface support index

(0 < SurfaceSupportId ≤ AxisVMSurfaceSupports.Count)

ResultBlockInfo array of description records for result blocks

Forces support force results for all result blocks

Retrieves surface support forces on the given surface element in all result blocks consecutively. The number of result blocks depends on the AnalysisType property.

Returns the common length of ResultBlockInfo and Forces arrays or an error code ([errDatabaseNotReady](#) or see [EForcesError](#)).

long **SurfaceSupportForceValuesForResultBlocks** ([*in*] long **SurfaceSupportId**,

[*in*] [ESurfaceVertexType](#) **SurfaceVertexType**, [*in*] long **SurfaceVertexId**,

[*i/o*] SAFEARRAY([RResultBlockInfo](#)) **ResultBlockInfo**,

[*i/o*] SAFEARRAY([RSurfaceSupportForces](#)) **Forces**)

SurfaceSupportId surface support index

(0 < SurfaceSupportId ≤ AxisVMSurfaceSupports.Count)

SurfaceVertexType vertex type

SurfaceVertexId vertex identifier

ResultBlockInfo array of description records for result blocks

Forces support force results for all result blocks

Retrieves surface support forces at a given point of a given surface element in all result blocks consecutively. The number of result blocks depends on the AnalysisType property.

Returns the common length of ResultBlockInfo and Forces arrays or an error code ([errDatabaseNotReady](#) or see [EForcesError](#)).

Edge Connection Forces

Single LOCATION reader functions

long **GetEdgeConnectionForcesByLoadCaseId** ([*in*] long **EdgeConnectionId**, [*in*] long **LoadCaseId**,

[*in*] long **LoadLevel**, [*in*] [EAnalysisType](#) **AnalysisType**, [*i/o*] [REdgeConnectionForces](#)* **Forces**,

[*out*] BSTR* **Combination**)

EdgeConnectionId edge connection index

(0 < EdgeConnectionId ≤ AxisVMEdgeConnections.Count)

LoadCaseId load case index (0 < LoadCaseId ≤ AxisVMLoadCases.Count)

LoadLevel load level (increment) index

AnalysisType type of [analysis](#)

Forces forces in edge connection

Combination name of the load case

If successful returns edge connection index, otherwise returns an error code

([errDatabaseNotReady](#), [feInvalidCombinationOfLoadCaseAndLoadLevel](#), [feInvalidAnalysisType](#), [feEdgeConnectionIndexOutOfBounds](#)).

long	GetEdgeConnectionForcesByLoadCombinationId ([in] long EdgeConnectionId , [in] long LoadCombinationId , [in] long LoadLevel , [in] EAnalysisType AnalysisType , [i/o] REdgeConnectionForces * Forces , [out] BSTR* Combination)
	EdgeConnectionId <i>edge connection index (0 < EdgeConnectionId ≤ AxisVMEConnections.Count)</i>
	LoadCombinationId <i>load combination index (0 < LoadCombinationId ≤ AxisVMLoadCombinations.Count)</i>
	LoadLevel <i>load level (increment) index</i>
	AnalysisType <i>type of analysis</i>
	Forces <i>forces in edge connection</i>
	Combination <i>name of the combination</i>
	<i>If successful returns edge connection index, otherwise returns an error code (errDatabaseNotReady, feInvalidCombinationOfLoadCombinationAndLoadLevel, feInvalidAnalysisType, feEdgeConnectionIndexOutOfBounds).</i>
long	GetEnvelopeEdgeConnectionForces ([in] long EdgeConnectionId , [in] long SectionId , [in] EMinMaxType MinMaxType , [in] EAnalysisType AnalysisType , [in] EEdgeConnectionForceComponent , [i/o] REdgeConnectionForces * Forces , [out] BSTR* Combination)
	EdgeConnectionId <i>edge connection index (0 < EdgeConnectionId ≤ AxisVMEConnections.Count)</i>
	SectionId <i>section index (1,2 or 3) along edge</i>
	MinMaxType <i>minimum or maximum value</i>
	AnalysisType <i>type of analysis</i>
	Component <i>edge connection force component</i>
	Forces <i>forces of edge connection</i>
	Combination <i>name of the combination</i>
	<i>Envelope is identified by EdgeConnectionForceComponent, MinMaxType and EnvelopeUID properties. If successful returns edge connection index, otherwise returns an error code (errDatabaseNotReady, feInvalidAnalysisType, feEdgeConnectionIndexOutOfBounds, feSectionIndexOutOfBounds).</i>
long	GetEnvelopeEdgeConnectionForces2 ([in] long EdgeConnectionId , [in] long SectionId , [in] EMinMaxType MinMaxType , [in] EAnalysisType AnalysisType , [in] EEdgeConnectionForceComponent , [i/o] REdgeConnectionForces * Forces , [out] long LoadCaseOrCombinationId , [out] long LoadLevel)
	EdgeConnectionId <i>edge connection index (0 < EdgeConnectionId ≤ AxisVMEConnections.Count)</i>
	SectionId <i>section index (1,2 or 3) along edge</i>
	MinMaxType <i>minimum or maximum value</i>
	AnalysisType <i>type of analysis</i>
	Component <i>edge connection force component</i>
	Forces <i>forces of edge connection</i>
	LoadCaseOrCombinationId <i>load case or load combination index, if index is > IAxisVMLoadcases.Count then Load combination index = LoadCaseOrCombinationId - IAxisVMLoadcases.Count</i>
	LoadLevel <i>load level</i>
	<i>Envelope is identified by EdgeConnectionForceComponent, MinMaxType and EnvelopeUID properties. If successful returns edge connection index, otherwise returns an error code (errDatabaseNotReady, feInvalidAnalysisType, feEdgeConnectionIndexOutOfBounds, feSectionIndexOutOfBounds).</i>

long **GetCriticalEdgeConnectionForces** ([in] long **EdgeConnectionId**, [in] long **SectionId**,
[in] [EMinMaxType](#) **MinMaxType**, [in] [ECombinationType](#) **CombinationType**, [in] [EAnalysisType](#)
[AnalysisType](#), [in] [EEdgeConnectionForce](#) **Component**, [i/o] [REdgeConnectionForces](#)* **Forces**,
[out] BSTR* **Combination**)

EdgeConnectionId edge connection index
($0 < EdgeConnectionId \leq AxisVMEConnections.Count$)
SectionId section index (1,2 or 3) along edge
MinMaxType minimum or maximum value
CombinationType combination type
AnalysisType type of analysis
Component edge connection force component
Forces forces of edge connection
Combination name of critical combination

If successful returns edge connection index, otherwise returns an error code ([errDatabaseNotReady](#),
[feInvalidAnalysisType](#), [feEdgeConnectionIndexOutOfBounds](#), [feSectionIndexOutOfBounds](#),
[feCombinationTypeNotValidForCurrentNationalDesignCode](#)).

long **GetCriticalEdgeConnectionForces2** ([in] long **EdgeConnectionId**, [in] long **SectionId**,
[in] [EMinMaxType](#) **MinMaxType**, [in] [ECombinationType](#) **CombinationType**, [in] [EAnalysisType](#)
[AnalysisType](#), [in] [EEdgeConnectionForce](#) **Component**, [i/o] [REdgeConnectionForces](#)* **Forces**,
[out] [ECCombinationType](#) **CriticalCombinationType**, [out] SAFEARRAY(double) **Factors**,
[out] SAFEARRAY(long) **LoadCaselds**)

EdgeConnectionId edge connection index
($0 < EdgeConnectionId \leq AxisVMEConnections.Count$)
SectionId section index (1,2 or 3) along edge
MinMaxType minimum or maximum value
CombinationType combination type
AnalysisType type of analysis
Component edge connection force component
Forces forces of edge connection
CriticalCombinationType combination type corresponding to critical result
Factors load factors of the critical load combination
LoadCaselds load case indexes of the critical load combination

If successful returns edge connection index, otherwise returns an error code ([errDatabaseNotReady](#),
[feInvalidAnalysisType](#), [feEdgeConnectionIndexOutOfBounds](#), [feSectionIndexOutOfBounds](#),
[feCombinationTypeNotValidForCurrentNationalDesignCode](#)).

long **EdgeConnectionForcesByLoadCaseld** ([in] long **EdgeConnectionId**,
[i/o] [REdgeConnectionForces](#)* **Forces**, [out] BSTR* **Combination**)

EdgeConnectionId edge connection index
($0 < EdgeConnectionId \leq AxisVMEConnections.Count$)
Forces forces in edge connection
Combination name of the load case

Envelope is identified by EdgeConnectionForceComponent, MinMaxType and EnvelopeUID properties.
If successful returns edge connection index, otherwise returns an error code ([errDatabaseNotReady](#),
[feInvalidCombinationOfLoadCaseAndLoadLevel](#), [feInvalidAnalysisType](#), [feEdgeConnectionIndexOutOfBounds](#)).

long	EdgeConnectionForcesByLoadCombinationId ([in] long EdgeConnectionId , [i/o] REdgeConnectionForces * Forces , [out] BSTR* Combination)
	<p style="margin-left: 20px;">EdgeConnectionId edge connection index $(0 < EdgeConnectionId \leq AxisVMEConnections.Count)$</p> <p style="margin-left: 20px;">Forces forces</p> <p style="margin-left: 20px;">Combination name of the combination</p> <p><i>Envelope is identified by EdgeConnectionForceComponent, MinMaxType and EnvelopeUID properties. If successful returns edge connection index, otherwise returns an error code (errDatabaseNotReady, feInvalidCombinationOfLoadCombinationAndLoadLevel, feInvalidAnalysisType, feEdgeConnectionIndexOutOfBounds).</i></p>
long	EnvelopeEdgeConnectionForces ([in] long EdgeConnectionId , [in] long SectionId , [i/o] REdgeConnectionForces * Forces , [out] BSTR* Combination)
	<p style="margin-left: 20px;">EdgeConnectionId edge connection index $(0 < EdgeConnectionId \leq AxisVMEConnections.Count)$</p> <p style="margin-left: 20px;">SectionId section index (1,2 or 3) along edge</p> <p style="margin-left: 20px;">Forces forces of edge connection</p> <p style="margin-left: 20px;">Combination name of the combination</p> <p><i>If successful returns edge connection index, otherwise returns an error code (errDatabaseNotReady, feInvalidAnalysisType, feEdgeConnectionIndexOutOfBounds, feSectionIndexOutOfBounds).</i></p>
long	EnvelopeEdgeConnectionForces2 ([in] long EdgeConnectionId , [in] long SectionId , [i/o] REdgeConnectionForces * Forces , [out] long LoadCaseOrCombinationId , [out] long LoadLevel)
	<p style="margin-left: 20px;">EdgeConnectionId edge connection index $(0 < EdgeConnectionId \leq AxisVMEConnections.Count)$</p> <p style="margin-left: 20px;">SectionId section index (1,2 or 3) along edge</p> <p style="margin-left: 20px;">Forces forces of edge connection</p> <p style="margin-left: 20px;">LoadCaseOrCombinationId load case or load combination index, if index is > IAxisVMLoadcases.count then Load combination index = LoadCaseOrCombinationId - IAxisVMLoadcases.count</p> <p style="margin-left: 20px;">LoadLevel load level</p> <p><i>Envelope is identified by EdgeConnectionForceComponent, MinMaxType and EnvelopeUID properties. If successful returns edge connection index, otherwise returns an error code (errDatabaseNotReady, feInvalidAnalysisType, feEdgeConnectionIndexOutOfBounds, feSectionIndexOutOfBounds).</i></p>
long	CriticalEdgeConnectionForces ([in] long EdgeConnectionId , [in] long SectionId , [i/o] REdgeConnectionForces * Forces , [out] BSTR* Combination)
	<p style="margin-left: 20px;">EdgeConnectionId edge connection index $(0 < EdgeConnectionId \leq AxisVMEConnections.Count)$</p> <p style="margin-left: 20px;">SectionId section index (1 for NN or 1,2 or 3 for LL)</p> <p style="margin-left: 20px;">Forces forces of edge connection</p> <p style="margin-left: 20px;">Combination name of critical combination</p> <p><i>If successful returns edge connection index, otherwise returns an error code (errDatabaseNotReady, feInvalidAnalysisType, feEdgeConnectionIndexOutOfBounds, feSectionIndexOutOfBounds, feCombinationTypeNotValidForCurrentNationalDesignCode).</i></p>
long	CriticalEdgeConnectionForces2 ([in] long EdgeConnectionId , [in] long SectionId , [i/o] REdgeConnectionForces * Forces , [out] ECombinationType CriticalCombinationType , [out] SAFEARRAY(double) Factors , [out] SAFEARRAY(long) LoadCaselds)
	<p style="margin-left: 20px;">CriticalCombinationType combination type corresponding to critical load combination</p> <p style="margin-left: 20px;">Factors load factors of the critical load combination</p> <p style="margin-left: 20px;">LoadCaselds load case indexes of the critical load combination</p> <p><i>Similar to CriticalEdgeConnectionForces with more parameters described above.</i></p>

Multiple element reader functions

long	GetAllEdgeConnectionForcesByLoadCaseId ([in] long LoadCaseId, [in] long LoadLevel, [in] EAnalysisType AnalysisType, [out] SAFEARRAY(REdgeConnectionForces)* Forces)
	<p>LoadCaseId load case index ($0 < \text{LoadCaseId} \leq \text{AxisVMLoadCases.Count}$)</p> <p>LoadLevel load level or time step (increment) index</p> <p>AnalysisType type of Analysis</p> <p>Forces array with forces of all edge connections</p> <p>If successful returns number of edge connections, otherwise returns an error code (errDatabaseNotReady, feInvalidCombinationOfLoadCaseAndLoadLevel, feInvalidAnalysisType, feNoEdgeConnectionsInTheModel).</p>
long	GetAllEdgeConnectionForcesByLoadCombinationId ([in] long LoadCombinationId, [in] long LoadLevel, [in] EAnalysisType AnalysisType, [out] SAFEARRAY(REdgeConnectionForces)* Forces)
	<p>LoadCombinationId load combination index ($0 < \text{LoadCombinationId} \leq \text{AxisVMLoadCombinations.Count}$)</p> <p>LoadLevel load level or time step (increment) index</p> <p>AnalysisType type of Analysis</p> <p>Forces array with forces of all edge connections</p> <p>If successful returns number of edge connections, otherwise returns an error code (errDatabaseNotReady, feInvalidCombinationOfLoadCombinationAndLoadLevel, feInvalidAnalysisType, feNoEdgeConnectionsInTheModel).</p>
long	GetAllEnvelopeEdgeConnectionForces ([in] EMinMaxType MinMaxType, [in] EAnalysisType AnalysisType, [in] EEdgeConnectionForce Component, [out] SAFEARRAY(REdgeConnectionForces)* Forces)
	<p>MinMaxType minimum or maximum value</p> <p>AnalysisType type of analysis</p> <p>Component edge connection force component</p> <p>Forces array with forces of all edge connections (each edge connection returns three REdgeConnectionForces records)</p> <p>Envelope is identified by EdgeConnectionForceComponent, MinMaxType and EnvelopeUID properties. If successful returns 3 x number of edge connections, otherwise returns an error code (errDatabaseNotReady, feInvalidAnalysisType, feNoEdgeConnectionsInTheModel).</p>
long	GetAllCriticalEdgeConnectionForces ([in] EMinMaxType MinMaxType, [in] ECombinationType CombinationType, [in] EAnalysisType AnalysisType, [in] EEdgeConnectionForce Component, [out] SAFEARRAY(REdgeConnectionForces)* Forces)
	<p>MinMaxType minimum or maximum value</p> <p>CombinationType combination type</p> <p>AnalysisType type of analysis</p> <p>Component edge connection force component</p> <p>Forces array with forces of all edge connections (each edge connection returns three REdgeConnectionForces records)</p> <p>If successful returns 3 x number of edge connections, otherwise returns an error code (errDatabaseNotReady, feInvalidAnalysisType, feNoEdgeConnectionsInTheModel, feCombinationTypeNotValidForCurrentNationalDesignCode).</p>
long	AllEdgeConnectionForcesByLoadCaseId ([out] SAFEARRAY(REdgeConnectionForces)* Forces)
	<p>Forces array with forces of all edge connections</p> <p>If successful returns number of edge connections, otherwise returns an error code (errDatabaseNotReady, feInvalidCombinationOfLoadCaseAndLoadLevel, feInvalidAnalysisType, feNoEdgeConnectionsInTheModel).</p>

long	AllEdgeConnectionForcesByLoadCombinationId ([out] SAFEARRAY(REdgeConnectionForces)* Forces) Forces array with forces of all edge connections <i>If successful returns number of edge connections, otherwise returns an error code (errDatabaseNotReady, feInvalidCombinationOfLoadCombinationAndLoadLevel, feInvalidAnalysisType, feNoEdgeConnectionsInTheModel).</i>
long	AllEnvelopeEdgeConnectionForces ([out] SAFEARRAY(REdgeConnectionForces)* Forces) Forces array with forces of all edge connections <i>Envelope is identified by EdgeConnectionForceComponent, MinMaxType and EnvelopeUID properties. If successful returns 3 x number of edge connections, otherwise returns an error code (errDatabaseNotReady, feInvalidAnalysisType, feNoEdgeConnectionsInTheModel).</i>
long	AllCriticalEdgeConnectionForces ([out] SAFEARRAY(REdgeConnectionForces)* Forces) Forces array with forces of all edge connections <i>If successful returns 3 x number of edge connections, otherwise returns an error code (errDatabaseNotReady, feInvalidAnalysisType, feNoEdgeConnectionsInTheModel, feCombinationTypeNotValidForCurrentNationalDesignCode).</i>

Multiple BLOCK reader functions

long	GetEdgeConnectionForcesForResultBlocks ([in] long EdgeConnectionId , [in] EAAnalysisType AnalysisType , [out] SAFEARRAY(RResultBlockInfo)* ResultBlockInfo , [out] SAFEARRAY(REdgeConnectionForces)* Forces) EdgeConnectionId edge connection index <i>(0 < EdgeConnectionId ≤ AxisVMEdgeConnections.Count)</i> AnalysisType type of analysis ResultBlockInfo array of result block info Forces array with forces of all edge connections <i>If successful returns number of result blocks (the number of result blocks depends on the AnalysisType parameter), otherwise returns an error code (errDatabaseNotReady, feInvalidAnalysisType, feNoResultBlocksInTheModel, feEdgeConnectionIndexOutOfBounds).</i>
long	EdgeConnectionForcesForResultBlocks ([in] long EdgeConnectionId , [out] SAFEARRAY(RResultBlockInfo)* ResultBlockInfo , [out] SAFEARRAY(REdgeConnectionForces)* Forces) EdgeConnectionId edge connection index <i>(0 < EdgeConnectionId ≤ AxisVMEdgeConnections.Count)</i> ResultBlockInfo array of result block info Forces array with forces of all edge connections <i>If successful returns number of result blocks (the number of result blocks depends on the AnalysisType parameter), otherwise returns an error code (errDatabaseNotReady, feInvalidAnalysisType, feNoResultBlocksInTheModel, feEdgeConnectionIndexOutOfBounds).</i>

Link Element Forces

Single LOCATION reader functions

long	GetLinkElementForcesByLoadCaseId ([in] long LinkElementId , [in] long LoadCaseId , [in] long LoadLevel , [in] EAnalysisType AnalysisType , [i/o] RLinkElementForces* Forces , [out] BSTR* Combination)
	<p>LinkElementId <i>link element index ($0 < \text{LinkElementId} \leq \text{AxisVMLinkElements.Count}$)</i></p> <p>LoadCaseId <i>load case index ($0 < \text{LoadCaseId} \leq \text{AxisVMLoadCases.Count}$)</i></p> <p>LoadLevel <i>load level or time step (increment) index</i></p> <p>AnalysisType <i>type of analysis</i></p> <p>Forces <i>forces in edge connection</i></p> <p>Combination <i>name of the load case</i></p> <p><i>If successful returns edge connection index, otherwise returns an error code (errDatabaseNotReady, feInvalidCombinationOfLoadCaseAndLoadLevel, feInvalidAnalysisType, feLinkElementIndexOutOfBounds).</i></p>
long	GetLinkElementForcesByLoadCombinationId ([in] long LinkElementId , [in] long LoadCombinationId , [in] long LoadLevel , [in] EAnalysisType AnalysisType , [i/o] RLinkElementForces* Forces , [out] BSTR* Combination)
	<p>LinkElementId <i>link element index ($0 < \text{LinkElementId} \leq \text{AxisVMLinkElements.Count}$)</i></p> <p>LoadCombinationId <i>load combination index ($0 < \text{LoadCombinationId} \leq \text{AxisVMLoadCombinations.Count}$)</i></p> <p>LoadLevel <i>load level or time step (increment) index</i></p> <p>AnalysisType <i>type of analysis</i></p> <p>Forces <i>forces in edge connection</i></p> <p>Combination <i>name of the combination</i></p> <p><i>If successful returns edge connection index, otherwise returns an error code (errDatabaseNotReady, feInvalidCombinationOfLoadCombinationAndLoadLevel, feInvalidAnalysisType, feLinkElementIndexOutOfBounds).</i></p>
long	GetEnvelopeLinkElementForces ([in] long LinkElementId , [in] long SectionId , [in] EMinMaxType MinMaxType , [in] EAnalysisType AnalysisType , [in] ELinkElementForceComponent , [i/o] RLinkElementForces* Forces , [out] BSTR* Combination)
	<p>LinkElementId <i>link element index ($0 < \text{LinkElementId} \leq \text{AxisVMLinkElements.Count}$)</i></p> <p>SectionId <i>section index (1 for NN or 1,2 or 3 for LL)</i></p> <p>MinMaxType <i>minimum or maximum value</i></p> <p>AnalysisType <i>type of analysis</i></p> <p>Component <i>link element force component</i></p> <p>Forces <i>forces of edge connection</i></p> <p>Combination <i>name of the combination</i></p> <p><i>Envelope is identified by LinkElementForceComponent, MinMaxType and EnvelopeUID properties. If successful returns edge connection index, otherwise returns an error code (errDatabaseNotReady, feInvalidAnalysisType, feLinkElementIndexOutOfBounds, feSectionIndexOutOfBounds (for LL elements)).</i></p>

long	GetCriticalLinkElementForces ([in] long LinkElementId , [in] long SectionId , [in] EMinMaxType MinMaxType , [in] ECombinationType CombinationType , [in] EAnalysisType AnalysisType , [in] ELinkElementForce Component , [i/o] RLinkElementForces* Forces , [out] BSTR* Combination)
	<p style="margin-left: 20px;">LinkElementId <i>link element index (0 < LinkElementId ≤ AxisVMLinkElements.Count)</i></p> <p style="margin-left: 20px;">SectionId <i>section index (1 for NN or 1,2 or 3 fol LL)</i></p> <p style="margin-left: 20px;">MinMaxType <i>minimum or maximum value</i></p> <p style="margin-left: 20px;">CombinationType <i>combination type</i></p> <p style="margin-left: 20px;">AnalysisType <i>type of analysis</i></p> <p style="margin-left: 20px;">Component <i>link element force component</i></p> <p style="margin-left: 20px;">Forces <i>forces of edge connection</i></p> <p style="margin-left: 20px;">Combination <i>name of critical combination</i></p> <p>If successful returns edge connection index, otherwise returns an error code (errDatabaseNotReady, feInvalidAnalysisType, feLinkElementIndexOutOfBounds, feSectionIndexOutOfBounds (for LL elements), feCombinationTypeNotValidForCurrentNationalDesignCode).</p>
long	LinkElementForcesByLoadCaseId ([in] long LinkElementId , [i/o] RLinkElementForces* Forces , [out] BSTR* Combination)
	<p style="margin-left: 20px;">LinkElementId <i>link element index (0 < LinkElementId ≤ AxisVMLinkElements.Count)</i></p> <p style="margin-left: 20px;">Forces <i>forces in edge connection</i></p> <p style="margin-left: 20px;">Combination <i>name of the load case</i></p> <p>If successful returns edge connection index, otherwise returns an error code (errDatabaseNotReady, feInvalidCombinationOfLoadCaseAndLoadLevel, feInvalidAnalysisType, feLinkElementIndexOutOfBounds).</p>
long	LinkElementForcesByLoadCombinationId ([in] long LinkElementId , [i/o] RLinkElementForces* Forces , [out] BSTR* Combination)
	<p style="margin-left: 20px;">LinkElementId <i>link element index (0 < LinkElementId ≤ AxisVMLinkElements.Count)</i></p> <p style="margin-left: 20px;">Forces <i>forces in edge connection</i></p> <p style="margin-left: 20px;">Combination <i>name of the combination</i></p> <p>If successful returns edge connection index, otherwise returns an error code (errDatabaseNotReady, feInvalidCombinationOfLoadCombinationAndLoadLevel, feInvalidAnalysisType, feLinkElementIndexOutOfBounds).</p>
long	EnvelopeLinkElementForces ([in] long LinkElementId , [in] long SectionId , [i/o] RLinkElementForces* Forces , [out] BSTR* Combination)
	<p style="margin-left: 20px;">LinkElementId <i>link element index (0 < LinkElementId ≤ AxisVMLinkElements.Count)</i></p> <p style="margin-left: 20px;">SectionId <i>section index (1 for NN or 1,2 or 3 fol LL)</i></p> <p style="margin-left: 20px;">Forces <i>forces of edge connection</i></p> <p style="margin-left: 20px;">Combination <i>name of the combination</i></p> <p>Envelope is identified by <i>LinkElementForceComponent</i>, <i>MinMaxType</i> and <i>EnvelopeUID</i> properties. If successful returns edge connection index, otherwise returns an error code (errDatabaseNotReady, feInvalidAnalysisType, feLinkElementIndexOutOfBounds, feSectionIndexOutOfBounds (for LL elements)).</p>

long	EnvelopeLinkElementForces2 ([in] long LinkElementId , [in] long SectionId , [i/o] RLinkElementForces * Forces , [out] long LoadCaseOrCombinationId , [out] long LoadLevel)
	LinkElementId <i>link element index ($0 < \text{LinkElementId} \leq \text{AxisVMLinkElements.Count}$)</i> SectionId <i>section index (1 for NN or 1,2 or 3 fol LL)</i> Forces <i>forces of edge connection</i> LoadCaseOrCombinationId <i>load case or load combination index, if index is > $\text{IAxisVMLoadcases.count}$ then Load combination index = $\text{LoadCaseOrCombinationId} - \text{IAxisVMLoadcases.count}$</i> LoadLevel <i>load level</i>
	<i>Envelope is identified by LinkElementForceComponent, MinMaxType and EnvelopeUID properties. If successful returns edge connection index, otherwise returns an error code (errDatabaseNotReady, feInvalidAnalysisType, feLinkElementIndexOutOfBounds, feSectionIndexOutOfBounds (for LL elements)).</i>
long	CriticalLinkElementForces ([in] long LinkElementId , [in] long SectionId , [i/o] RLinkElementForces * Forces , [out] BSTR* Combination)
	LinkElementId <i>link element index ($0 < \text{LinkElementId} \leq \text{AxisVMLinkElements.Count}$)</i> SectionId <i>section index (1 for NN or 1,2 or 3 fol LL)</i> Forces <i>forces of edge connection</i> Combination <i>name of critical combination</i>
	<i>If successful returns edge connection index, otherwise returns an error code (errDatabaseNotReady, feInvalidAnalysisType, feLinkElementIndexOutOfBounds, feSectionIndexOutOfBounds (for LL elements), feCombinationTypeNotValidForCurrentNationalDesignCode.).</i>
long	CriticalLinkElementForces2 ([in] long LinkElementId , [in] long SectionId , [i/o] RLinkElementForces * Forces , [out] ECombinationType CriticalCombinationType , [bout] SAFEARRAY(double) Factors , [out] SAFEARRAY(long) LoadCaselds)
	CriticalCombinationType <i>combination type corresponding to critical load combination</i> Factors <i>load factors of the critical load combination</i> LoadCaselds <i>load case indexes of the critical load combination</i> <i>Similar to CriticalLinkElementForces with more parameters described above.</i>

Multiple element reader functions

long	GetAllLinkElementForcesByLoadCaseld ([in] long LoadCaseld , [in] long LoadLevel , [in] EAnalysisType AnalysisType , [out] SAFEARRAY(RLinkElementForces)* Forces)
	LoadCaseld <i>load case index ($0 < \text{LoadCaseld} \leq \text{AxisVMLoadCases.Count}$)</i> LoadLevel <i>load level or time step (increment) index</i> AnalysisType <i>type of analysis</i> Forces <i>Array with forces from all link elements</i>
	<i>If successful returns number of link elements, otherwise returns an error code (errDatabaseNotReady, feInvalidCombinationOfLoadCaseAndLoadLevel, feInvalidAnalysisType).</i>

long	GetAllLinkElementForcesByLoadCombinationId ([in] long LoadCombinationId, [in] long LoadLevel, [in] EAnalysisType AnalysisType, [out] SAFEARRAY(RLinkElementForces)* Forces)
	<p>LoadCombinationId <i>load combination index ($0 < \text{LoadCombinationId} \leq \text{AxisVMLoadCombinations.Count}$)</i></p> <p>LoadLevel <i>load level or time step (increment) index</i></p> <p>AnalysisType <i>type of analysis</i></p> <p>Forces <i>Array with forces from all link elements</i></p> <p><i>If successful returns number of link elements, otherwise returns an error code (errDatabaseNotReady, felInvalidCombinationOfLoadCombinationAndLoadLevel, felInvalidAnalysisType).</i></p>
long	GetAllEnvelopeLinkElementForces ([in] EMinMaxType MinMaxType, [in] EAnalysisType AnalysisType, [in] ELinkElementForce Component, [out] SAFEARRAY(RLinkElementForces)* Forces)
	<p>MinMaxType <i>minimum or maximum value</i></p> <p>AnalysisType <i>type of analysis</i></p> <p>Component <i>link element force component</i></p> <p>Forces <i>Array with forces from all link elements</i></p> <p><i>Envelope is identified by LinkElementForceComponent, MinMaxType and EnvelopeUID properties. If successful returns 3 x number of link elements for LL and number of link elements for NN, otherwise returns an error code (errDatabaseNotReady, felInvalidAnalysisType, feSectionIndexOutOfBounds(for LL elements))</i></p>
long	GetAllCriticalLinkElementForces ([in] EMinMaxType MinMaxType, [in] ECombinationType CombinationType, [in] EAnalysisType AnalysisType, [in] ELinkElementForce Component, [out] SAFEARRAY(RLinkElementForces)* Forces)
	<p>MinMaxType <i>minimum or maximum value</i></p> <p>CombinationType <i>combination type</i></p> <p>AnalysisType <i>type of analysis</i></p> <p>Component <i>link element force component</i></p> <p>Forces <i>Array with forces from all link elements</i></p> <p><i>If successful returns 3 x number of link elements for LL and number of link elements for NN, otherwise returns an error code (errDatabaseNotReady, felInvalidAnalysisType, feCombinationTypeNotValidForCurrentNationalDesignCode).</i></p>
long	AllLinkElementForcesByLoadCaseId ([out] SAFEARRAY(RLinkElementForces)* Forces)
	<p>Forces <i>Array with forces from all link elements</i></p> <p><i>If successful returns number of link elements, otherwise returns an error code (errDatabaseNotReady, felInvalidCombinationOfLoadCaseAndLoadLevel, felInvalidAnalysisType).</i></p>
long	AllLinkElementForcesByLoadCombinationId ([out] SAFEARRAY(RLinkElementForces)* Forces)
	<p>Forces <i>Array with forces from all link elements</i></p> <p><i>If successful returns number of link elements, otherwise returns an error code (errDatabaseNotReady, felInvalidCombinationOfLoadCombinationAndLoadLevel, felInvalidAnalysisType).</i></p>
long	AllEnvelopeLinkElementForces ([out] SAFEARRAY(RLinkElementForces)* Forces)
	<p>Forces <i>Array with forces from all link elements</i></p> <p><i>Envelope is identified by LinkElementForceComponent, MinMaxType and EnvelopeUID properties. If successful returns 3 x number of link elements for LL and number of link elements for NN, otherwise returns an error code (errDatabaseNotReady, felInvalidAnalysisType, feSectionIndexOutOfBounds(for LL elements))</i></p>

long **AllCriticalLinkElementForces** ([out] SAFEARRAY([RLinkElementForces](#))* **Forces**)
 Forces Array with forces from all link elements
 If successful returns 3 x number of link elements for LL and number of link elements for NN, otherwise returns an error code ([errDatabaseNotReady](#), [feInvalidAnalysisType](#), [feCombinationTypeNotValidForCurrentNationalDesignCode](#)).

Multiple BLOCK reader functions

long **GetLinkElementForcesForResultBlocks** ([in] long **LinkElementId**, [in] [EAnalysisType](#) **AnalysisType**, [out] SAFEARRAY([RResultBlockInfo](#))* **ResultBlockInfo**, [out] SAFEARRAY([RLinkElementForces](#))* **Forces**)
 LinkElementId link element index
 ($0 < \text{LinkElementId} \leq \text{AxisVMLinkElements.Count}$)
 AnalysisType type of [analysis](#)
 ResultBlockInfo array of result block info
 Forces Array with forces from all link elements
 If successful returns number of result blocks (the number of result blocks depends on the AnalysisType parameter), otherwise returns an error code ([errDatabaseNotReady](#), [feInvalidAnalysisType](#), [feNoResultBlocksInTheModel](#), [feLinkElementIndexOutOfBounds](#)).

long **LinkElementForcesForResultBlocks** ([in] long **LinkElementId**, [out] SAFEARRAY([RResultBlockInfo](#))* **ResultBlockInfo**, [out] SAFEARRAY([RLinkElementForces](#))* **Forces**)
 LinkElementId link element index
 ($0 < \text{LinkElementId} \leq \text{AxisVMLinkElements.Count}$)
 ResultBlockInfo array of result block info
 Forces Array with forces from all link elements
 If successful returns number of result blocks (the number of result blocks depends on the AnalysisType parameter), otherwise returns an error code ([errDatabaseNotReady](#), [feInvalidAnalysisType](#), [feNoResultBlocksInTheModel](#), [feLinkElementIndexOutOfBounds](#)).

Member Forces

long **GetMemberForcesByLoadCasId** ([in] long **MemberID**, [in] long **LoadCasId**, [in] long **LoadLevelOrTimeStep**, [in] [EAnalysisType](#) **AnalysisType**, [out] SAFEARRAY([RLineForceValues](#)) **Forces**, [out] SAFEARRAY(double) **PosX**)
 MemberID member index ($0 < \text{MemberId} \leq \text{AxisVMMModel.Members.Count}$)
 LoadCasId load case index
 LoadLevelOrTimeStep load level or time step, according to the type of analysis. See [LoadLevelOrModeShapeOrTimeStep](#) parameter.
 AnalysisType analysis type
 Forces array of member forces for the member
 PosX array of cross-section positions in m according to the local x direction
 Retrieves forces for each cross-section of a given element according to the LoadCasId and LoadLevelOrTimeStep. Returns the total number of cross-sections or an error code ([errDatabaseNotReady](#) or see [EForcesError](#)).

long	GetMemberStressesByLoadCombinationId ([in] long MemberID, [in] long LoadCombinationId, [in] long LoadLevelOrTimeStep, [in] EAnalysisType AnalysisType, [out] SAFEARRAY(RLineForceValues) Forces, [out] SAFEARRAY(double) PosX)
	<p>MemberID member index ($0 < MemberId \leq AxisVMMModel.Members.Count$)</p> <p>LoadCombinationId load combination index</p> <p>LoadLevelOrTimeStep load level or time step, according to the type of analysis. See LoadLevelOrModeShapeOrTimeStep parameter.</p> <p>AnalysisType analysis type</p> <p>Forces array of member forces for the member</p> <p>PosX array of cross-section positions in m according to the local x direction</p> <p>Retrieves forces for each cross-section of a given element according to the LoadCombinationId and LoadLevelOrTimeStep. Returns the total number of cross-sections or an error code (errDatabaseNotReady or see EForcesError).</p>
long	GetEnvelopeMemberForces ([in] long MemberID, [in] long EnvelopeUID, [in] EMinMaxType MinMaxType, [in] EAnalysisType AnalysisType, [in] ELineForce Component, [out] SAFEARRAY(RLineForceValues) Forces, [out] SAFEARRAY(double) PosX, [out] SAFEARRAY(long) LoadCaseOrCombinationIds, [out] SAFEARRAY(long) LoadLevels)
	<p>MemberID member index ($0 < MemberId \leq AxisVMMModel.Members.Count$)</p> <p>EnvelopeUID unique envelope index</p> <p>MinMaxType minimum or maximum value</p> <p>AnalysisType analysis type</p> <p>Component force component</p> <p>Forces array of member forces for the member</p> <p>PosX array of cross-section positions in m according to the local x direction</p> <p>LoadCaseOrCombinationIds array of load case or load combination indexes, if index is $> \text{IAxisVMLoadcases}.Count$ then Load combination index = LoadCaseOrCombinationId - IAxisVMLoadcases.Count</p> <p>LoadLevels array of load levels</p> <p>Retrieves envelope forces for each cross-section of a member. Envelope is identified by MinMaxType, Component and EnvelopeUI. Forces array contains the result of the load case or combination in which Component is maximal or minimal. Returns the total number of cross-sections or an error code (errDatabaseNotReady or see EForcesError).</p>
long	GetCriticalMemberForces ([in] long MemberID, [in] EMinMaxType MinMaxType, [in] ECombinationType CombinationType, [in] EAnalysisType AnalysisType, [in] ELineForce Component, [out] SAFEARRAY(RLineForceValues) Forces, [out] SAFEARRAY(double) PosX)
	<p>MemberID member index ($0 < MemberId \leq AxisVMMModel.Members.Count$)</p> <p>MinMaxType Minimum or maximum value</p> <p>CombinationType combination type</p> <p>AnalysisType analysis type</p> <p>Component force component</p> <p>Forces array of member forces for the member</p> <p>PosX array of cross-section positions in m according to the local x direction</p> <p>Retrieves critical forces in each cross-section of a member. Critical combination is identified by Component and MinMaxType properties (e.g. IsSmin). Forces array contains the result of the critical combination in which Component is maximal or minimal. Returns the total number of cross-sections or an error code (errDatabaseNotReady or see EForcesError).</p>

Save results to MetaFile functions

long **SaveCriticalMemberForcesToMetaFile** ([in] BSTR **FileName**, [in] long **MemberId**,
[in] [ECombinationType](#) **CombinationType**, [in] [EAnalysisType](#) **AnalysisType**, [in] long **Width**,
[in] long **Height**, [in] [EBoolean](#) **EnvelopeOnly**, [in] double **Position**,
[in] [EWindowColourMode](#) **ColourMode**)

FileName	<i>name of the file with extension emf</i>
MemberId	<i>member index</i>
CombinationType	<i>combination type</i>
AnalysisType	<i>analysis type</i>
Width	<i>picture's size in pixel (minimal acceptable value is 640)</i>
Height	<i>picture's size in pixel (minimal acceptable value is 580)</i>
EnvelopeOnly	<i>only Envelope</i>
Position	<i>position of the investigated cross section along the member</i>
ColourMode	<i>colour mode</i>

It creates a metafile from the member's critical forces. It returns MemberId or an error code ([EGeneralError](#) or see [EForcesError](#)).

long **SaveEnvelopeMemberForcesToMetaFile** ([in] BSTR **FileName**, [in] long **MemberId**,
[in] long **EnvelopeUID**, [in] [EAnalysisType](#) **AnalysisType**, [in] long **Width**, [in] long **Height**,
[in] [EBoolean](#) **EnvelopeOnly**, [in] double **Position**, [in] [EWindowColourMode](#) **ColourMode**)

FileName	<i>name of the file with extension emf</i>
MemberId	<i>member index</i>
EnvelopeUID	<i>unique envelope index</i>
AnalysisType	<i>analysis type</i>
Width	<i>picture's size in pixel (minimal acceptable value is 640)</i>
Height	<i>picture's size in pixel (minimal acceptable value is 580)</i>
EnvelopeOnly	<i>only Envelope</i>
Position	<i>position of the investigated cross section along the member</i>
ColourMode	<i>colour mode</i>

It creates a metafile from the member's forces envelope. It returns MemberId or an error code ([EGeneralError](#) or see [EForcesError](#)).

long **SaveMemberForcesToMetaFileByLoadCaseId** ([in] BSTR **FileName**, [in] long **MemberId**,
[in] long **LoadCaseId**, [in] long **LoadLevelOrTimeStep**, [in] [EAnalysisType](#) **AnalysisType**,
[in] long **Width**, [in] long **Height**, [in] double **Position**, [in] [EWindowColourMode](#) **ColourMode**)

FileName	<i>name of the file with extension emf</i>
MemberId	<i>member index</i>
LoadCaseId	<i>load case index</i>
LoadLevelOrTimeStep	<i>for load level ($0 < LoadLevelOrTimeStep \leq LoadLevelCount$)</i> <i>for timestep ($0 < LoadLevelOrTimeStep \leq TimeStepCount$)</i>
AnalysisType	<i>analysis type</i>
Width	<i>picture's size in pixel (minimal acceptable value is 640)</i>
Height	<i>picture's size in pixel (minimal acceptable value is 580)</i>
Position	<i>position of the investigated cross section along the member</i>
ColourMode	<i>colour mode</i>

It saves the member's forces by LoadCaseId into a metafile. It returns MemberId or an error code ([EGeneralError](#) or see [EForcesError](#)).

long **SaveMemberForcesToMetaFileByLoadCombinationID** ([in] BSTR **FileName**, [in] long **MemberId**, [in] long **LoadCombinationId**, [in] long **LoadLevelOrTimeStep**, [in] **EAnalysisType** **AnalysisType**, [in] long **Width**, [in] long **Height**, [in] double **Position**, [in] **EWindowColourMode** **ColourMode**)

FileName	<i>name of the file with extension emf</i>
MemberId	<i>member index</i>
LoadCombinationId	<i>load combination index</i>
LoadLevelOrTimeStep	<i>for load level ($0 < \text{LoadLevelOrTimeStep} \leq \text{LoadLevelCount}$)</i> <i>for timestep ($0 < \text{LoadLevelOrTimeStep} \leq \text{TimestepCount}$)</i>
AnalysisType	<i>analysis type</i>
Width	<i>picture's size in pixel (minimal acceptable value is 640)</i>
Height	<i>picture's size in pixel (minimal acceptable value is 580)</i>
Position	<i>position of the investigated cross section along the member</i>
ColourMode	<i>colour mode</i>

It saves the member's forces by LoadCombinationId into a metafile. It returns MemberId or an error code ([EGeneralError](#) or see [EForcesError](#)).

long **SetUserCreep** ([in] **EBoolean** **Creep**)

Creep *If lbTrue concrete creep will be considered in results of nonlinear analysis if national design code allows it. More [here...](#)*

Enable or disable consideration of concrete creep in nonlinear analysis results.. If successful returns 1, otherwise an error code ([errDatabaseNotReady](#), [errCreepNotSupported](#)).

Properties

EAnalysisType	AnalysisType • Get or set the type of analysis (linear/nonlinear/vibration/buckling)
ECombinationType	CombinationType • Get or set the type of combination for critical results
EEdgeConnectionForce	EdgeConnectionForceComponent • Get or set the component for envelope or critical results.
long	EnvelopeUID • Get or set the unique index of the envelope used in functions for reading envelope results
long	LoadCaseId • Get or set load case index ($0 < \text{LoadCaseId} \leq \text{AxisVMModel.LoadCases.Count}$)
long	LoadCombinationId • Get or set load combination index ($0 < \text{LoadCombinationId} \leq \text{AxisVMModel.LoadCombinations.Count}$)
long	LoadLevelOrTimeStep • Get or set the load level or time step, according to the type of analysis. See LoadLevelOrModeShapeOrTimeStep parameter.
ELineForce	LineForceComponent • Get or set the line force component for envelope or critical results
ELineSupportForce	LineSupportForceComponent • Get or set the line support force component for envelope or critical results
ELinkElementForce	LinkElementForceComponent • Get or set the link element force component for envelope or critical results
EMinMaxType	MinMaxType • Get or set whether results containing minimum or maximum values of the component (LineForceComponent, LineSupportForceComponent, etc.) should be read (for envelope or critical values)
ESeismicComponentSumType	SeismicComponentSumType • Get or set the type of summation of seismic internal forces
ENodalSupportForce	NodalSupportForceComponent • Get or set the nodal support force component for envelope or critical results
ESpringForce	SpringForceComponent • Get or set the spring force component for envelope or critical results
ESurfaceForce	SurfaceForceComponent • Get or set the surface force component for envelope or critical results

<u>ESurfaceSupportForce</u>	SurfaceSupportForceComponent • Get or set the surface support force component for envelope or critical results
<u>EBoolean</u>	UserCreep Returns <i>IbTrue</i> if nonlinear analysis results consider concrete creep. More here...
<u>EVirtualBeamForce</u>	VirtualBeamForceComponent • Get or set the virtual beam force component for envelope or critical results

IAxisVMReinforcementCheck

Interface for reading reinforcement design forces

Error codes

```
enum EReinforcementCheckError = {
    rceCOMError = -100001
    rceLoadCaseIdIndexOutOfBounds = -100002
    rceLoadCombinationIdIndexOutOfBounds = -100003
    rceInvalidAnalysisType = -100004
    rceCombinationTypeNotValidForCurrentNationalDesignCode
        = -100005

    rceInvalidCombinationOfLoadCaseAndLoadLevel = -100006
    rceInvalidCombinationOfLoadCombinationAndLoadLevel = -
        100007}
```

COM server error
Invalid LoadCaseID while reading results
Invalid CombinationID while reading results
Invalid type of results for used analysis
The CombinationType cannot be used for the selected national code. Some design codes allow only certain ECombinationTypes (see [here](#)) or results not available for CombinationType
Invalid combination LoadCaseID and load level
Invalid combination CombinationID and load level

Records / structures

RReinforcementCheckValues = (

double	rmxdmin_ULS	Minimum design moment for ULS in x direction [kNm/m]
double	rmxdmax_ULS	Maximum design moment for ULS in x direction [kNm/m]
double	rmydmin_ULS	Minimum design moment for ULS in y direction [kNm/m]
double	rmydmax_ULS	Maximum design moment for ULS in y direction [kNm/m]
double	rmxminin_ULS	Minimum ultimate(residual) moment for ULS in x direction [kNm/m]
double	rmxmax_ULS	Maximum ultimate(residual) moment for ULS in x direction [kNm/m]
double	rmyumin_ULS	Minimum ultimate(residual) moment for ULS in y direction [kNm/m]
double	rmyumax_ULS	Maximum ultimate(residual) moment for ULS in y direction [kNm/m]
double	rmxdmin_SLS	Minimum design moment for SLS in x direction [kNm/m]
double	rmxdmax_SLS	Maximum design moment for SLS in x direction [kNm/m]
double	rmydmin_SLS	Minimum design moment for SLS in y direction [kNm/m]
double	rmydmax_SLS	Maximum design moment for SLS in y direction [kNm/m]
double	rmxminin_SLS	Minimum ultimate(residual) moment for SLS in x direction [kNm/m]
double	rmxmax_SLS	Maximum ultimate(residual) moment for SLS in x direction [kNm/m]
double	rmyumin_SLS	Minimum ultimate(residual) moment for SLS in y direction [kNm/m]
double	rmyumax_SLS	Maximum ultimate(residual) moment for SLS in y direction [kNm/m]
)	

RReinforcementCheck = (

long	ContourPointCount	Number of contour points
long	ContourPoint1Id	index of contour point 1
long	ContourPoint2Id	index of contour point 2
long	ContourPoint3Id	index of contour point 3
long	ContourPoint4Id	index of contour point 4
long	ContourLine1Id	index of contour line midpoint 1
long	ContourLine2Id	index of contour line midpoint 2
long	ContourLine3Id	index of contour line midpoint 3
long	ContourLine4Id	index of contour line midpoint 4
RReinforcementCheckValues	rcvCenterPoint	Reinforcement check at centre point
RReinforcementCheckValues	rcvContourPoint1	Reinforcement check at contour point 1
RReinforcementCheckValues	rcvContourPoint2	Reinforcement check at contour point 2
RReinforcementCheckValues	rcvContourPoint3	Reinforcement check at contour point 3
RReinforcementCheckValues	rcvContourPoint4	Reinforcement check at contour point 4
RReinforcementCheckValues	rcvContourLineMidPoint1	Reinforcement check at contour line midpoint 1
RReinforcementCheckValues	rcvContourLineMidPoint2	Reinforcement check at contour line midpoint 2
RReinforcementCheckValues	rcvContourLineMidPoint3	Reinforcement check at contour line midpoint 3
RReinforcementCheckValues	rcvContourLineMidPoint4	Reinforcement check at contour line midpoint 4
)	

Functions

long	GetReinforcementChecksByLoadCaselId ([in] long SurfacelD , [in] long LoadCaselD , [in] long LoadLevel , [in] EAnalysisType AnalysisType , [i/o] RReinforcementCheck * ReinforcementCheck , [out] BSTR* Combination)
	<p style="margin-left: 20px;">SurfacelD <i>index of the surface element</i></p> <p style="margin-left: 20px;">LoadCaselD <i>load case index (0 < LoadCaselD ≤ AxisVMLoadCases.Count)</i></p> <p style="margin-left: 20px;">LoadLevel <i>load level (increment) index</i></p> <p style="margin-left: 20px;">AnalysisType <i>Type of Analysis</i></p> <p style="margin-left: 20px;">ReinforcementCheck <i>Reinforcement checks of the surface element</i></p> <p style="margin-left: 20px;">Combination <i>String with name of load case.</i></p> <p><i>If successful returns SurfacelD, otherwise an error code (errDatabaseNotReady, errNotSupportedByNationalDesignCode, errIndexOutOfBoundsException, rcelLoadCaseIndexOutOfBoundsException, rcelInvalidCombinationOfLoadCaseAndLoadLevel, rcelInvalidAnalysisType).</i></p>
long	GetReinforcementChecksByLoadCombinationId ([in] long SurfacelD , [in] long LoadCombinationId , [in] long LoadLevel , [in] EAnalysisType AnalysisType , [i/o] RReinforcementCheck * ReinforcementCheck , [out] BSTR* Combination)
	<p style="margin-left: 20px;">SurfacelD <i>index of the surface element</i></p> <p style="margin-left: 20px;">LoadCombinationId <i>load combination index (0 < LoadCombinationId ≤ AxisVMLoadCombinations.Count)</i></p> <p style="margin-left: 20px;">LoadLevel <i>load level (increment) index</i></p> <p style="margin-left: 20px;">AnalysisType <i>Type of Analysis</i></p> <p style="margin-left: 20px;">ReinforcementCheck <i>Reinforcement checks of the surface element</i></p> <p style="margin-left: 20px;">Combination <i>String with name of combination</i></p> <p><i>If successful returns SurfacelD, otherwise an error code (errDatabaseNotReady, errNotSupportedByNationalDesignCode, errIndexOutOfBoundsException, rcelLoadCombinationIndexOutOfBoundsException, rcelInvalidCombinationOfLoadCombinationAndLoadLevel, rcelInvalidAnalysisType).</i></p>
long	GetEnvelopeReinforcementChecks ([in] long SurfacelD , [in] EMinMaxType MinMaxType , [in] EAnalysisType AnalysisType , [i/o] RReinforcementCheck * ReinforcementCheck)
	<p style="margin-left: 20px;">SurfacelD <i>index of the surface element</i></p> <p style="margin-left: 20px;">MinMaxType <i>Minimum or maximum value</i></p> <p style="margin-left: 20px;">AnalysisType <i>Type of Analysis</i></p> <p style="margin-left: 20px;">ReinforcementCheck <i>Reinforcement checks of the surface element</i></p> <p><i>Envelope is identified by MinMaxType and EnvelopeUID properties. If successful returns SurfacelD, otherwise an error code (errDatabaseNotReady, errNotSupportedByNationalDesignCode, errIndexOutOfBoundsException, rcelInvalidAnalysisType).</i></p>
long	GetCriticalReinforcementChecks ([in] long SurfacelD , [in] EMinMaxType MinMaxType , [in] ECombinationType CombinationType , [in] EAnalysisType AnalysisType , [i/o] RReinforcementCheck * ReinforcementCheck)
	<p style="margin-left: 20px;">SurfacelD <i>index of the surface element</i></p> <p style="margin-left: 20px;">MinMaxType <i>Minimum or maximum value</i></p> <p style="margin-left: 20px;">CombinationType <i>Combination Type</i></p> <p style="margin-left: 20px;">AnalysisType <i>Type of Analysis</i></p> <p style="margin-left: 20px;">ReinforcementCheck <i>Reinforcement checks of the surface element</i></p> <p><i>If successful returns SurfacelD, otherwise an error code (errDatabaseNotReady, errNotSupportedByNationalDesignCode, errIndexOutOfBoundsException, rcelInvalidAnalysisType).</i></p>

long	GetAllReinforcementChecksByLoadCaseId ([in] long LoadCaseId , [in] long LoadLevel , [in] EAnalysisType AnalysisType , [out] SAFEARRAY(RReinforcementCheck) [*] ReinforcementChecks , [out] SAFEARRAY (BSTR) [*] Combinations)
	<p>LoadCaseId <i>load case index ($0 < LoadCaseId \leq \text{AxisVMLoadCases.Count}$)</i></p> <p>LoadLevel <i>load level (increment) index</i></p> <p>AnalysisType <i>Type of Analysis</i></p> <p>ReinforcementChecks <i>Array of reinforcement checks of each surface element</i></p> <p>Combinations <i>Array of string with name of load cases.</i></p> <p>If successful returns number of surface elements, otherwise an error code (errDatabaseNotReady, errNotSupportedByNationalDesignCode, rceLoadCaseIdIndexOutOfBounds, rceInvalidCombinationOfLoadCaseAndLoadLevel, rceInvalidAnalysisType).</p>
long	GetAllReinforcementChecksByLoadCombinationId ([in] long LoadCombinationId , [in] long LoadLevel , [in] EAnalysisType AnalysisType , [out] SAFEARRAY(RReinforcementCheck) [*] ReinforcementChecks , [out] SAFEARRAY (BSTR) [*] Combinations)
	<p>LoadCombinationId <i>load combination index ($0 < LoadCombinationId \leq \text{AxisVMLoadCombinations.Count}$)</i></p> <p>LoadLevel <i>load level (increment) index</i></p> <p>AnalysisType <i>Type of Analysis</i></p> <p>ReinforcementChecks <i>Array of reinforcement checks of each surface element</i></p> <p>Combinations <i>Array of strings with name of combinations.</i></p> <p>If successful returns number of surface elements, otherwise an error code (errDatabaseNotReady, errNotSupportedByNationalDesignCode, rceLoadCombinationIdIndexOutOfBounds, rceInvalidCombinationOfLoadCombinationAndLoadLevel, rceInvalidAnalysisType).</p>
long	GetAllEnvelopeReinforcementChecks ([in] EMinMaxType MinMaxType , [in] EAnalysisType AnalysisType , [out] SAFEARRAY(RReinforcementCheck) [*] ReinforcementChecks)
	<p>MinMaxType <i>Minimum or maximum value</i></p> <p>AnalysisType <i>Type of Analysis</i></p> <p>ReinforcementChecks <i>Array of reinforcement checks of each surface element</i> <i>Envelope is identified by MinMaxType and EnvelopeUID properties. If successful returns number of surface elements, otherwise an error code (errDatabaseNotReady, errNotSupportedByNationalDesignCode, rceInvalidAnalysisType).</i></p>
long	GetAllCriticalReinforcementChecks ([in] EMinMaxType MinMaxType , [in] ECombinationType CombinationType , [in] EAnalysisType AnalysisType , [out] SAFEARRAY(RReinforcementCheck) [*] ReinforcementChecks)
	<p>MinMaxType <i>Minimum or maximum value</i></p> <p>CombinationType <i>Combination Type</i></p> <p>AnalysisType <i>Type of Analysis</i></p> <p>ReinforcementChecks <i>Array of reinforcement checks of each surface element</i> <i>If successful returns number of surface elements, otherwise an error code (errDatabaseNotReady, errNotSupportedByNationalDesignCode, rceInvalidAnalysisType).</i></p>

long	ReinforcementChecksByLoadCaseId ([in] long SurfaceId, [i/o] <u>RReinforcementCheck</u> * ReinforcementCheck, [out] BSTR* Combination)
	<p>SurfaceId index of the surface element ReinforcementCheck Reinforcement checks of the surface element Combination String with name of load case.</p> <p>If successful returns SurfaceId, otherwise an error code (errDatabaseNotReady, errNotSupportedByNationalDesignCode, errIndexOutOfBoundsException, rceLoadCaseIdIndexOutOfBoundsException, rceInvalidCombinationOfLoadCaseAndLoadLevel, rceInvalidAnalysisType).</p>
long	ReinforcementChecksByLoadCombinationId ([in] long SurfaceId, [i/o] <u>RReinforcementCheck</u> * ReinforcementCheck, [out] BSTR* Combination)
	<p>SurfaceId index of the surface element ReinforcementCheck Reinforcement checks of the surface element Combination String with name of load combination.</p> <p>If successful returns SurfaceId, otherwise an error code (errDatabaseNotReady, errNotSupportedByNationalDesignCode, errIndexOutOfBoundsException, rceLoadCombinationIdIndexOutOfBoundsException, rceInvalidCombinationOfLoadCombinationAndLoadLevel, rceInvalidAnalysisType).</p>
long	EnvelopeReinforcementChecks ([in] long SurfaceId, [i/o] <u>RReinforcementCheck</u> * ReinforcementCheck)
	<p>SurfaceId index of the surface element ReinforcementCheck Reinforcement checks of the surface element <i>Envelope</i> is identified by <i>MinMaxType</i> and <i>EnvelopeUID</i> properties. If successful returns SurfaceId, otherwise an error code (errDatabaseNotReady, errNotSupportedByNationalDesignCode, errIndexOutOfBoundsException, rceInvalidAnalysisType).</p>
long	CriticalReinforcementChecks ([in] long SurfaceId, [i/o] <u>RReinforcementCheck</u> * ReinforcementCheck)
	<p>SurfaceId index of the surface element ReinforcementCheck Reinforcement checks of the surface element If successful returns SurfaceId, otherwise an error code (errDatabaseNotReady, errNotSupportedByNationalDesignCode, errIndexOutOfBoundsException, rceInvalidAnalysisType).</p>
long	AllReinforcementChecksByLoadCaseId ([out] SAFEARRAY(<u>RReinforcementCheck</u>)* ReinforcementChecks, [out] SAFEARRAY (BSTR)* Combinations)
	<p>ReinforcementChecks Array of reinforcement checks of each surface element Combinations Array of string with name of load cases.</p> <p>If successful returns number of surface elements, otherwise an error code (errDatabaseNotReady, errNotSupportedByNationalDesignCode, rceLoadCaseIdIndexOutOfBoundsException, rceInvalidCombinationOfLoadCaseAndLoadLevel, rceInvalidAnalysisType).</p>
long	AllReinforcementChecksByLoadCombinationId ([out] SAFEARRAY(<u>RReinforcementCheck</u>)* ReinforcementChecks, [out] SAFEARRAY (BSTR)* Combinations)
	<p>ReinforcementChecks Array of reinforcement checks of each surface element Combinations Array of string with name of combinations.</p> <p>If successful returns number of surface elements, otherwise an error code (errDatabaseNotReady, errNotSupportedByNationalDesignCode, rceLoadCombinationIdIndexOutOfBoundsException, rceInvalidCombinationOfLoadCombinationAndLoadLevel, rceInvalidAnalysisType).</p>

long	AllEnvelopeReinforcementChecks ([out] SAFEARRAY(RReinforcementCheck)* ReinforcementChecks)
	ReinforcementChecks <i>Array of reinforcement checks of each surface element Envelope is identified by MinMaxType and EnvelopeUID properties If successful returns number of surface elements, otherwise an error code (errDatabaseNotReady, errNotSupportedByNationalDesignCode, rcelInvalidAnalysisType).</i>
long	AllCriticalReinforcementChecks ([out] SAFEARRAY(RReinforcementCheck)* ReinforcementChecks)
	ReinforcementChecks <i>Array of reinforcement checks of each surface element If successful returns number of surface elements, otherwise an error code (errDatabaseNotReady, errNotSupportedByNationalDesignCode, rcelInvalidAnalysisType).</i>

Properties

EAnalysisType	AnalysisType • Get or set type of analysis
long	Count <i>Get number of surface elements in the model</i>
ECombinationType	CombinationType • Get or set the combination type
long	EnvelopeUID • Get or set the unique index of the envelope used in functions for reading envelope results
long	LoadCaseId • Get or set the load case index <i>(0 < LoadCaseId ≤ AxisVMLoadCases.Count)</i>
long	LoadCombinationId • Get or set load combination index <i>(0 < LoadCombinationId ≤ AxisVMLoadCombinations.Count)</i>
long	LoadLevel • Get or set load level (increment) index
EMaxMinType	MinMaxType • Get or set whether results containing minimum or maximum should be read (for envelope or critical values)

IAxisVMShearCapacity

Interface for shear capacity calculations.

If property returning this interface is null (nil) then the extension module RC3 is not available.

Error codes

enum	EShearCapacitiesError = {	
	sceCOMError = -100001	COM server error
	sceLoadCaseIdIndexOutOfBounds = -100002	Invalid LoadCaseID while reading results
	sceLoadCombinationIdIndexOutOfBounds = -100003	Invalid CombinationID while reading results
	sceInvalidAnalysisType = -100004	Invalid type of results for used analysis
	sceCombinationTypeNotValidForCurrentNationalDesignCode = -100005	The CombinationType cannot be used for the selected national code. Some design codes allow only certain ECombinationTypes (see here) or results not available for CombinationType
	sceInvalidCombinationOfLoadCaseAndLoadLevel = -100006	Invalid combination LoadCaseID and load level
	sceInvalidCombinationOfLoadCombinationAndLoadLevel = -100007	Invalid combination CombinationID and load level or this

Enumerated types

enum	EShearCapacity : = {	
	scVRdc = 0	<i>shear resistance</i>
	scVEdMinusVRdc = 1	<i>resultant shear force minus shear resistance</i>
	scVRdmax = 2	<i>maximum shear resistance</i>
	scVEDivVRdmax = 3	<i>resultant shear force divided by maximum shear resistance</i>
	scaVED = 4}	<i>angle of resultant shear force</i>
	<i>Shear capacity</i>	

Records / structures

	RShearCapacities = (
long	ContourPointCount	<i>number of contour points</i>
long	ContourPoint1Id	<i>index of contour point 1</i>
long	ContourPoint2Id	<i>index of contour point 2</i>
long	ContourPoint3Id	<i>index of contour point 3</i>
long	ContourPoint4Id	<i>index of contour point 4</i>
long	ContourLine1Id	<i>index of contour line midpoint 1</i>
long	ContourLine2Id	<i>index of contour line midpoint 2</i>
long	ContourLine3Id	<i>index of contour line midpoint 3</i>
long	ContourLine4Id	<i>index of contour line midpoint 4</i>
	scvCenterPoint	<i>shear capacity at centre point</i>
	scvContourPoint1	<i>shear capacity at contour point 1</i>
	scvContourPoint2	<i>shear capacity at contour point 2</i>
	scvContourPoint3	<i>shear capacity at contour point 3</i>
	scvContourPoint4	<i>shear capacity at contour point 4</i>
	scvContourLineMidPoint1	<i>shear capacity at contour line midpoint 1</i>
	scvContourLineMidPoint2	<i>shear capacity at contour line midpoint 2</i>
	scvContourLineMidPoint3	<i>shear capacity at contour line midpoint 3</i>
	scvContourLineMidPoint4	<i>shear capacity at contour line midpoint 4</i>
)	
	RShearCapacityValues = (
double	VRdc	<i>shear resistance [kN/m]</i>
double	VEdMinusVRdc	<i>resultant shear force minus shear resistance [kN/m]</i>
double	VRdmax	<i>maximum shear resistance [kN/m]</i>
double	VEDivVRdmax	<i>resultant shear force divided by maximum shear resistance [kN/m]</i>
double	aVED	<i>angle of resultant shear force [°]</i>
)	

Functions

long **GetShearCapacitiesByLoadCaseId** ([in] long **Surfaceld**, [in] long **LoadCaseId**,
[in] long **LoadLevel**, [in] **EAnalysisType** **AnalysisType**, [i/o] **RShearCapacities*** **ShearCapacities**,
[out] BSTR* **Combination**)

Surfaceld index of the surface element
LoadCaseId load case index ($0 < \text{LoadCaseId} \leq \text{AxisVMLoadCases.Count}$)
LoadLevel load level (increment) index
AnalysisType Type of Analysis
ShearCapacities Shear capacities of the surface element
Combination String with name of load case.

If successful returns Surfaceld, otherwise an error code ([errDatabaseNotReady](#),
[errNotSupportedByNationalDesignCode](#), [errIndexOutOfBounds](#),
[sceLoadCaseIdOutOfBounds](#), [sceInvalidAnalysisType](#),
[sceInvalidCombinationOfLoadCaseAndLoadLevel](#)).

long **GetShearCapacitiesByLoadCombinationId** ([in] long **Surfaceld**, [in] long **LoadCombinationId**,
[in] long **LoadLevel**, [in] **EAnalysisType** **AnalysisType**, [i/o] **RShearCapacities*** **ShearCapacities**,
[out] BSTR* **Combination**)

Surfaceld index of the surface element
LoadCombinationId load combination index ($0 < \text{LoadCombinationId} \leq \text{AxisVMLoadCombinations.Count}$)
LoadLevel load level (increment) index
AnalysisType Type of Analysis
ShearCapacities Shear capacities of the surface element
Combination String with name of combination.

If successful returns Surfaceld, otherwise an error code ([errDatabaseNotReady](#),
[errNotSupportedByNationalDesignCode](#), [errIndexOutOfBounds](#),
[sceLoadCombinationIdOutOfBounds](#), [sceInvalidAnalysisType](#),
[sceInvalidCombinationOfLoadCombinationAndLoadLevel](#)).

long **GetEnvelopeShearCapacities** ([in] long **Surfaceld**, [in] **EMinMaxType** **MinMaxType**,
[in] **EAnalysisType** **AnalysisType**, [in] **EShearCapacity** **Component**,
[i/o] **RShearCapacities*** **ShearCapacities**, [out] BSTR* **Combination**)

Surfaceld index of the surface element
MinMaxType Minimum or maximum value
AnalysisType Type of Analysis
Component Type of shear capacities
ShearCapacities Shear capacities of the surface element
Combination Combination contain a multiline strings (separated with CR+LF) where
the lines belong to: ContourPoint1, ContourLineMidPoint1,
ContourPoint2, ContourLineMidPoint2, ContourPoint3,
ContourLineMidPoint3, [ContourPoint4, ContourLineMidPoint4],
CenterPoint

Envelope is identified by MinMaxType, Component and EnvelopeUID properties. If successful
returns Surfaceld, otherwise an error code ([errDatabaseNotReady](#),
[errNotSupportedByNationalDesignCode](#), [errIndexOutOfBounds](#), [sceInvalidAnalysisType](#)).

long	GetEnvelopeShearCapacities2 ([in] long SurfaceId, [in] EMinMaxType MinMaxType, [in] EAnalysisType AnalysisType, [in] EShearCapacity Component, [i/o] RShearCapacities* ShearCapacities, [out] long LoadCaseOrCombinationId, [out] long LoadLevel)
	<p style="margin-left: 20px;">SurfaceId <i>index of the surface element</i> MinMaxType <i>Minimum or maximum value</i> AnalysisType <i>Type of Analysis</i> Component <i>Type of shear capacities</i> ShearCapacities <i>Shear capacities of the surface element</i> LoadCaseOrCombinationId <i>load case or load combination index of midpoint results, if index is > IAxisVMLoadcases.count then Load combination index = LoadCaseOrCombinationId - IAxisVMLoadcases.count</i> LoadLevel <i>load level</i></p>
	<p><i>Envelope is identified by MinMaxType, Component and EnvelopeUID properties. If successful returns SurfaceId, otherwise an error code (errDatabaseNotReady, errNotSupportedByNationalDesignCode, errIndexOutOfBoundsException, sceInvalidAnalysisType).</i></p>
long	GetCriticalShearCapacities ([in] long SurfaceId, [in] EMinMaxType MinMaxType, [in] ECombinationType CombinationType, [in] EAnalysisType AnalysisType, [in] EShearCapacity Component, [i/o] RShearCapacities* ShearCapacities, [out] BSTR* Combination)
	<p style="margin-left: 20px;">SurfaceId <i>index of the surface element</i> MinMaxType <i>Minimum or maximum value</i> CombinationType <i>Combination Type</i> AnalysisType <i>Type of Analysis</i> Component <i>Type of shear capacities</i> ShearCapacities <i>Shear capacities of the surface element</i> Combination <i>Combination contain a multiline strings (separated with CR+LF) where the lines belong to: ContourPoint1, ContourLineMidPoint1, ContourPoint2, ContourLineMidPoint2, ContourPoint3, ContourLineMidPoint3, [ContourPoint4, ContourLineMidPoint4], CenterPoint</i></p>
	<p><i>If successful returns SurfaceId, otherwise an error code (errDatabaseNotReady, errNotSupportedByNationalDesignCode, errIndexOutOfBoundsException, sceInvalidAnalysisType).</i></p>
long	GetCriticalShearCapacities2 ([in] long SurfaceId, [in] EMinMaxType MinMaxType, [in] ECombinationType CombinationType, [in] EAnalysisType AnalysisType, [in] EShearCapacity Component, [i/o] RShearCapacities* ShearCapacities, [out] ECombinationType CriticalCombinationType, [out] SAFEARRAY(double) Factors, [out] SAFEARRAY(long) LoadCaseIds)
	<p style="margin-left: 20px;">SurfaceId <i>index of the surface element</i> MinMaxType <i>Minimum or maximum value</i> CombinationType <i>Combination Type</i> AnalysisType <i>Type of Analysis</i> Component <i>Type of shear capacities</i> ShearCapacities <i>Shear capacities of the surface element</i> CriticalCombinationType <i>combination type corresponding to critical result</i> Factors <i>load factors of the critical load combination for midpoint results</i> LoadCaseIds <i>load case indexes of the critical load combination for midpoint results</i></p>
	<p><i>If successful returns SurfaceId, otherwise an error code (errDatabaseNotReady, errNotSupportedByNationalDesignCode, errIndexOutOfBoundsException, sceInvalidAnalysisType).</i></p>

long	GetAllShearCapacitiesByLoadCaseId ([in] long LoadCaseId , [in] long LoadLevel , [in] EAnalysisType AnalysisType , [out] SAFEARRAY(RShearCapacities)* ShearCapacities , [out] SAFEARRAY (BSTR)* Combinations)
	<p>LoadCaseId <i>load case index (0 < LoadCaseId ≤ <u>AxisVMLoadCases.Count</u>)</i></p> <p>LoadLevel <i>load level (increment) index</i></p> <p>AnalysisType <i>Type of <u>Analysis</u></i></p> <p>ShearCapacities <i>Shear capacities of the surface element</i></p> <p>Combinations <i>Array of strings with names of load cases.</i></p>
	<i>If successful returns number of surface elements, otherwise an error code (errDatabaseNotReady, errNotSupportedByNationalDesignCode, errIndexOutOfBoundsException, sceLoadCaseIdOutOfBoundsException, sceInvalidAnalysisType, sceInvalidCombinationOfLoadCaseAndLoadLevel).</i>
long	GetAllShearCapacitiesByLoadCombinationId ([in] long LoadCombinationId , [in] long LoadLevel , [in] EAnalysisType AnalysisType , [out] SAFEARRAY(RShearCapacities)* ShearCapacities , [out] SAFEARRAY (BSTR)* Combinations)
	<p>LoadCombinationId <i>load combination index (0 < LoadCombinationId ≤ <u>AxisVMLoadCombinations.Count</u>)</i></p> <p>LoadLevel <i>load level (increment) index</i></p> <p>AnalysisType <i>Type of <u>Analysis</u></i></p> <p>ShearCapacities <i>Shear capacities of the surface element</i></p> <p>Combinations <i>Array of strings with combination names.</i></p>
	<i>If successful returns number of surface elements, otherwise an error code (errDatabaseNotReady, errNotSupportedByNationalDesignCode, errIndexOutOfBoundsException, sceLoadCombinationIdOutOfBoundsException, sceInvalidAnalysisType, sceInvalidCombinationOfLoadCombinationAndLoadLevel).</i>
long	GetAllEnvelopeShearCapacities ([in] EMinMaxType MinMaxType , [in] EAnalysisType AnalysisType , [in] EShearCapacity Component , [out] SAFEARRAY(RShearCapacities)* ShearCapacities , [out] SAFEARRAY (BSTR)* Combinations)
	<p>MinMaxType <i>Minimum or maximum value</i></p> <p>AnalysisType <i>Type of <u>Analysis</u></i></p> <p>Component <i>Type of shear capacities</i></p> <p>ShearCapacities <i>Shear capacities of the surface element</i></p> <p>Combinations <i>String array of multiline strings (separated with CR+LF) where the lines belong to: ContourPoint1, ContourLineMidPoint1, ContourPoint2, ContourLineMidPoint2, ContourPoint3, ContourLineMidPoint3, [ContourPoint4, ContourLineMidPoint4], CenterPoint for all surface elements</i></p>
	<i>Envelope is identified by MinMaxType, Component and EnvelopeUID properties. If successful returns number of surface elements, otherwise an error code (errDatabaseNotReady, errNotSupportedByNationalDesignCode, errIndexOutOfBoundsException, sceInvalidAnalysisType).</i>

long	GetAllCriticalShearCapacities ([in] EMinMaxType MinMaxType , [in] ECombinationType CombinationType , [in] EAnalysisType AnalysisType , [in] EShearCapacity Component , [out] SAFEARRAY(RShearCapacities) [*] ShearCapacities , [out] SAFEARRAY (BSTR) [*] Combinations)
	<p>MinMaxType <i>Minimum or maximum value</i></p> <p>CombinationType <i>Combination Type</i></p> <p>AnalysisType <i>Type of Analysis</i></p> <p>Component <i>Type of shear capacities</i></p> <p>ShearCapacities <i>Shear capacities of the surface element</i></p> <p>Combinations <i>String array of multiline strings (separated with CR+LF) where the lines belong to: ContourPoint1, ContourLineMidPoint1, ContourPoint2, ContourLineMidPoint2, ContourPoint3, ContourLineMidPoint3, [ContourPoint4, ContourLineMidPoint4], CenterPoint for all surface elements</i></p>
	<i>If successful returns number of surface elements, otherwise an error code (errDatabaseNotReady, errNotSupportedByNationalDesignCode, errIndexOutOfBounds, sceInvalidAnalysisType).</i>
long	ShearCapacitiesByLoadCaseId ([in] long Surfaceld , [i/o] RShearCapacities) [*] ShearCapacities , [out] BSTR [*] Combination)
	<p>Surfaceld <i>index of the surface element</i></p> <p>ShearCapacities <i>shear capacities of the surface element</i></p> <p>Combination <i>string with name of load case.</i></p>
	<i>If successful returns Surfaceld, otherwise an error code (errDatabaseNotReady, errNotSupportedByNationalDesignCode, errIndexOutOfBounds, sceLoadCaseIdOutOfBounds, sceInvalidAnalysisType, sceInvalidCombinationOfLoadCaseAndLoadLevel).</i>
long	ShearCapacitiesByLoadCombinationId ([in] long Surfaceld , [i/o] RShearCapacities) [*] ShearCapacities , [out] BSTR [*] Combination)
	<p>Surfaceld <i>index of the surface element</i></p> <p>ShearCapacities <i>shear capacities of the surface element</i></p> <p>Combination <i>string with name of combination.</i></p>
	<i>If successful returns Surfaceld, otherwise an error code (errDatabaseNotReady, errNotSupportedByNationalDesignCode, errIndexOutOfBounds, sceLoadCombinationIdOutOfBounds, sceInvalidAnalysisType, sceInvalidCombinationOfLoadCombinationAndLoadLevel).</i>
long	EnvelopeShearCapacities ([in] long Surfaceld , [i/o] RShearCapacities) [*] ShearCapacities , [out] BSTR [*] Combination)
	<p>Surfaceld <i>index of the surface element</i></p> <p>ShearCapacities <i>shear capacities of the surface element</i></p> <p>Combination <i>combination contain a multiline strings (separated with CR+LF) where the lines belong to: ContourPoint1, ContourLineMidPoint1, ContourPoint2, ContourLineMidPoint2, ContourPoint3, ContourLineMidPoint3, [ContourPoint4, ContourLineMidPoint4], CenterPoint</i></p>
	<i>Envelope is identified by MinMaxType, Component and EnvelopeUID properties. If successful returns Surfaceld, otherwise an error code (errDatabaseNotReady, errNotSupportedByNationalDesignCode, errIndexOutOfBounds, sceInvalidAnalysisType).</i>

long	EnvelopeShearCapacities2 ([in] long Surfaceld , [i/o] RShearCapacities * ShearCapacities , [out] long LoadCaseOrCombinationId , [out] long LoadLevel)
	Surfaceld index of the surface element
	ShearCapacities shear capacities of the surface element
	LoadCaseOrCombinationId load case or load combination index of midpoint results, if index is > IAxisVMLoadcases .count then Load combination index = LoadCaseOrCombinationId - IAxisVMLoadcases .count
	LoadLevel load level
	<i>Envelope is identified by MinMaxType, Component and EnvelopeUID properties. If successful returns Surfaceld, otherwise an error code (errDatabaseNotReady, errNotSupportedByNationalDesignCode, errIndexOutOfBounds, sceInvalidAnalysisType).</i>
long	CriticalShearCapacities ([in] long Surfaceld , [i/o] RShearCapacities * ShearCapacities , [out] BSTR* Combination)
	Surfaceld index of the surface element
	ShearCapacities shear capacities of the surface element
	Combination cCombination contain a multiline strings (separated with CR+LF) where the lines belong to: ContourPoint1, ContourLineMidPoint1, ContourPoint2, ContourLineMidPoint2, ContourPoint3, ContourLineMidPoint3, [ContourPoint4, ContourLineMidPoint4], CenterPoint
	<i>If successful returns Surfaceld, otherwise an error code (errDatabaseNotReady, errNotSupportedByNationalDesignCode, errIndexOutOfBounds, sceInvalidAnalysisType).</i>
long	CriticalShearCapacities2 ([in] long Surfaceld , [i/o] RShearCapacities * ShearCapacities , [out] ECombinationType CriticalCombinationType , [out] SAFEARRAY(double) Factors , [out] SAFEARRAY(long) LoadCaselds)
	CriticalCombinationType combination type corresponding to critical load combination
	Factors load factors of the critical load combination of midpoint results
	LoadCaselds load case indexes of the critical load combination of midpoint results
	<i>Similar to CriticalShearCapacities with more parameters described above.</i>
long	AllShearCapacitiesByLoadCaseld ([out] SAFEARRAY(RShearCapacities)* ShearCapacities , [out] SAFEARRAY (BSTR)* Combinations)
	ShearCapacities shear capacities of the surface element
	Combinations array of strings with name of load cases.
	<i>If successful returns number of surface elements, otherwise an error code (errDatabaseNotReady, errNotSupportedByNationalDesignCode, errIndexOutOfBounds, sceLoadCaseldIndexOutOfBounds, sceInvalidAnalysisType, sceInvalidCombinationOfLoadCaseAndLoadLevel).</i>
long	AllShearCapacitiesByLoadCombinationId ([out] SAFEARRAY(RShearCapacities)* ShearCapacities , [out] SAFEARRAY (BSTR)* Combinations)
	ShearCapacities shear capacities of the surface element
	Combinations array of strings with combination names.
	<i>If successful returns number of surface elements, otherwise an error code (errDatabaseNotReady, errNotSupportedByNationalDesignCode, errIndexOutOfBounds, sceLoadCombinationIdIndexOutOfBounds, sceInvalidAnalysisType, sceInvalidCombinationOfLoadCombinationAndLoadLevel).</i>

long	AllEnvelopeShearCapacities ([out] SAFEARRAY(RShearCapacities)* ShearCapacities , [out] SAFEARRAY (BSTR)* Combinations)
	<p>ShearCapacities shear capacities of the surface element Combinations string array of multiline strings (separated with CR+LF) where the lines belong to: ContourPoint1, ContourLineMidPoint1, ContourPoint2, ContourLineMidPoint2, ContourPoint3, ContourLineMidPoint3, [ContourPoint4, ContourLineMidPoint4], CenterPoint for all surface elements</p> <p><i>Envelope is identified by MinMaxType, Component and EnvelopeUID properties. If successful returns number of surface elements, otherwise an error code (errDatabaseNotReady, errNotSupportedByNationalDesignCode, errIndexOutOfBoundsException, scelInvalidAnalysisType).</i></p>
<hr/>	
long	AllCriticalShearCapacities ([out] SAFEARRAY(RShearCapacities)* ShearCapacities , [out] SAFEARRAY (BSTR)* Combinations)
	<p>ShearCapacities shear capacities of the surface element Combinations string array of multiline strings (separated with CR+LF) where the lines belong to: ContourPoint1, ContourLineMidPoint1, ContourPoint2, ContourLineMidPoint2, ContourPoint3, ContourLineMidPoint3, [ContourPoint4, ContourLineMidPoint4], CenterPoint for all surface elements</p> <p><i>If successful returns number of surface elements, otherwise an error code (errDatabaseNotReady, errNotSupportedByNationalDesignCode, errIndexOutOfBoundsException, scelInvalidAnalysisType).</i></p>
<hr/>	
long	SetUserCreep ([in] ELongBoolean Creep)
	<p>Creep If <i>IbTrue</i> concrete creep will be considered in results of nonlinear analysis, if national design code allows it. More here...</p> <p><i>Enable or disable consideration of concrete creep in nonlinear analysis results.. If successful returns 1, otherwise an error code (errDatabaseNotReady, errCreepNotSupported).</i></p>

Properties

EAnalysisType	AnalysisType • Get or set type of analysis
EShearCapacity	Component • Get or set type of shear capacity component
long	Count <i>Get number of surface elements in the model, if 0 then results not calculated or invalid.</i>
ECombinationType	CombinationType • Get or set the combination type
long	EnvelopeUID • Unique index of the envelope used in functions for reading envelope results
long	LoadCaseId • Get or set load case index ($0 < \text{LoadCaseId} \leq \text{AxisVMLoadCases.Count}$)
long	LoadCombinationId • Get or set load combination index ($0 < \text{LoadCombinationId} \leq \text{AxisVMLoadCombinations.Count}$)
long	LoadLevel • Get or set load level (increment) index
EMinMaxType	MinMaxType • Get or set whether results should contain minimum or maximum values of the component
ELongBoolean	UserCreep <i>Returns <i>IbTrue</i> if nonlinear analysis results consider concrete creep. More here...</i>

IAxisVMStresses

Interface containing stress results of the model.

For further details of result objects see [Overview of AxisVM result objects](#).

If the model database is not available when calling functions or reading properties an [*errDatabaseNotReady*](#) error code is returned.

Error codes

enum EStressesError = {	
steLineIndexOutOfBounds = -100001	<i>line index is out of bounds</i>
steLoadCaseIndexOutOfBounds = -100002	<i>LoadCaseId is out of bounds</i>
steLoadCombinationIndexOutOfBounds = -100003	<i>LoadCombinationId is out of bounds</i>
steNotValidLineType = -100004	<i>IAxisVMLine.LineType is not compatible with the reader function</i>
steSectionIndexOutOfBounds = -100005	<i>section index is out of bounds</i>
steCombinationTypeNotValidForCurrentNationalDesignCode = -100006	<i>CombinationType is incompatible with the current design code</i>
steCOMError = -100007	<i>internal COM error when calling GetRecordInfoFromGUIDs, SafeArrayCreateEx, SafeArrayAccesssData</i>
steLineHasNoSections = -100008	<i>line element has no cross-sections (e.g. link element)</i>
steNoValidLinesInTheModel = -100009	<i>no valid line elements in the model</i>
steLineStressComponentNotValidForThisLineType = -100010	<i>LineType is not compatible with the line stress component</i>
steInvalidAnalysisType = -100011	<i>AnalysisType is incompatible with the function</i>
steInvalidCombinationOfLoadCaseAndLoadLevel = -100012	<i>results are not available for the given LoadCaseId and LoadLevelOrModeShape or <i>this</i></i>
steInvalidCombinationOfLoadCombinationAndLoadLevel = -100013	<i>results are not available for the given LoadCombinationId and LoadLevelOrModeShape or <i>this</i></i>
steNoResultBlocksInTheModel = -100014	<i>no result blocks in the model</i>
steNodeIndexOutOfBounds = -100015	<i>node index out of bounds</i>
steSurfaceIndexOutOfBounds = -100016	<i>surface index out of bounds</i>
steNoSurfacesInTheModel = -100017	<i>no surface elements in the model</i>
steMemberIndexOutOfBounds = -100018	<i>member index is out of bounds</i>
steReadXLAMSurfaceStresses = -100019	<i>read stress results for XLAM elements with dedicated XLAM functions</i>
steInvalidSurfaceVertexType = -100020	<i>Invalid Surface Vertex Type</i>
steReadSurfaceStresses = -100021	<i>XLAM stress reading functions can return this, read normal surface stress results with dedicated surface stress reading functions</i>
steNotXLAMpanel = -100022	<i>XLAM efficiency reading functions can return this, read efficiencies with dedicated functions</i>
steXLAMmoduleNotAvailable = -100023	<i>XLAM efficiency reading functions can return this, read efficiencies with dedicated functions</i>
steStressPointIDOutOfBounds = -100024 }	<i>stress point's index is out of bounds</i>

Enumerated types

enum ELineStress = {	
IsSmin = 0x00	<i>Smin cross-section minimum of the axial stress [kN/m²]</i>
IsSmax = 0x01	<i>Smax cross-section maximum of the axial stress [kN/m²]</i>
IsVmin = 0x02	<i>Vmin cross-section minimum of shear stress [kN/m²]</i>
IsVmax = 0x03	<i>Vmax cross-section maximum of shear stress [kN/m²]</i>
IsSomin = 0x04	<i>Somin cross-section minimum of VonMises stress [kN/m²]</i>
IfSomax = 0x05	<i>Somax cross-section maximum of VonMises stress [kN/m²]</i>
IfVymean = 0x06	<i>Vy mean shear stress in local y direction [kN/m²]</i>
IfVzmean = 0x07 }	<i>Vz mean shear stress in local z direction [kN/m²]</i>

Line stress component identifiers

enum **ESurfaceStress** = {
ssSxx = 0x00 Sxx axial stress [kN/m^2]
ssSyy = 0x01 Syy axial stress [kN/m^2]
ssSxy = 0x02 Sxy shear stress [kN/m^2]
ssSxz = 0x03 Sxz shear stress [kN/m^2]
ssSyz = 0x04 Syz shear stress [kN/m^2]
ssSVM = 0x05 VonMises stress [kN/m^2]
ssS1 = 0x06 1st principal stress [kN/m^2]
ssS2 = 0x07 2nd principal stress [kN/m^2]
ssAs = 0x08 } principal direction angle [°]

Surface stress component identifiers

enum **ESurfaceStressPosition** = {
sspTop = 0x00 top layer
sspMiddle = 0x01 middle layer
sspBottom = 0x02 } bottom layer

Surface layer identifiers. Top and bottom is defined by local z direction.

enum **EXLAMSurfaceStress** = {
xssSxx_m_T = 0x00 Sxx flexural stresses at top [kN/m^2]
xssSyy_m_T = 0x01 Syy flexural stresses at top [kN/m^2]
xssSxx_m_B = 0x02 Sxx flexural stresses at bottom [kN/m^2]
xssSyy_m_B = 0x03 Syy flexural stresses at bottom [kN/m^2]
xssSxx_n = 0x04 Sxx axial stresses at centre [kN/m^2]
xssSyy_n = 0x05 Syy axial stresses at centre [kN/m^2]
xssSxy_max = 0x06 Syz shear stress [kN/m^2]
xssSxz_max = 0x07 Sxz shear stress [kN/m^2]
xssSrx_max = 0x08 rolling shear stress $\tau_{rx,max}$ [kN/m^2]
xssSry_max = 0x09 } rolling shear stress $\tau_{ry,max}$ [kN/m^2]

XLAM surface stress component identifiers

enum **EXLAMSurfaceEfficiency** = {
xse_M_N_0 = 0x00 XLAM efficiency M-N in grain direction [-]
xse_M_N_90 = 0x01 XLAM efficiency M-N perpendicular to grain direction [-]
xse_V_T = 0x02 XLAM efficiency in shear and torsion [-]
xse_Vr_N = 0x03 XLAM efficiency in rolling shear and normal force [-]
xse_Max = 0x04 } XLAM efficiency Maximum [-]

XLAM surface efficiency component identifiers

Records / structures

RLineStressValues = (

```

ELineType    IsvLineType      line type
long          IsvPointCount   number of cross-section's stress calculation points (SCPs)
double        IsvS1          axial stress in SCP 1 [kN/m2]
double        IsvS2          axial stress in SCP 2 [kN/m2]
double        IsvS3          axial stress in SCP 3 [kN/m2]
double        IsvS4          axial stress in SCP 4 [kN/m2]
double        IsvS5          axial stress in SCP 5 [kN/m2]
double        IsvS6          axial stress in SCP 6 [kN/m2]
double        IsvS7          axial stress in SCP 7 [kN/m2]
double        IsvS8          axial stress in SCP 8 [kN/m2]
double        IsvS9          axial stress in SCP 9 [kN/m2]
double        IsvV1          shear stress in SCP 1 [kN/m2]
double        IsvV2          shear stress in SCP 2 [kN/m2]
double        IsvV3          shear stress in SCP 3 [kN/m2]
double        IsvV4          shear stress in SCP 4 [kN/m2]
double        IsvV5          shear stress in SCP 5 [kN/m2]
double        IsvV6          shear stress in SCP 6 [kN/m2]
double        IsvV7          shear stress in SCP 7 [kN/m2]
double        IsvV8          shear stress in SCP 8 [kN/m2]
double        IsvV9          shear stress in SCP 9 [kN/m2]
double        IsvSo1         VonMises stress in SCP 1 [kN/m2]
double        IsvSo2         VonMises stress in SCP 2 [kN/m2]
double        IsvSo3         VonMises stress in SCP 3 [kN/m2]
double        IsvSo4         VonMises stress in SCP 4 [kN/m2]
double        IsvSo5         VonMises stress in SCP 5 [kN/m2]
double        IsvSo6         VonMises stress in SCP 6 [kN/m2]
double        IsvSo7         VonMises stress in SCP 7 [kN/m2]
double        IsvSo8         VonMises stress in SCP 8 [kN/m2]
double        IsvSo9         VonMises stress in SCP 9 [kN/m2]
double        IsvSeff1       nonlinear effective stress in SCP 1 [kN/m2]
double        IsvSeff2       nonlinear effective stress in SCP 2 [kN/m2]
double        IsvSeff3       nonlinear effective stress in SCP 3 [kN/m2]
double        IsvSeff4       nonlinear effective stress in SCP 4 [kN/m2]
double        IsvSeff5       nonlinear effective stress in SCP 5 [kN/m2]
double        IsvSeff6       nonlinear effective stress in SCP 6 [kN/m2]
double        IsvSeff7       nonlinear effective stress in SCP 7 [kN/m2]
double        IsvSeff8       nonlinear effective stress in SCP 8 [kN/m2]
double        IsvSeff9       nonlinear effective stress in SCP 9 [kN/m2]
double        Isvfy1         nonlinear actual yield strength in SCP 1 [kN/m2]
double        Isvfy2         nonlinear actual yield strength in SCP 2 [kN/m2]
double        Isvfy3         nonlinear actual yield strength in SCP 3 [kN/m2]
double        Isvfy4         nonlinear actual yield strength in SCP 4 [kN/m2]
double        Isvfy5         nonlinear actual yield strength in SCP 5 [kN/m2]
double        Isvfy6         nonlinear actual yield strength in SCP 6 [kN/m2]
double        Isvfy7         nonlinear actual yield strength in SCP 7 [kN/m2]
double        Isvfy8         nonlinear actual yield strength in SCP 8 [kN/m2]
double        Isvfy9         nonlinear actual yield strength in SCP 9 [kN/m2]
double        IsvKih1        nonlinear utilization in SCP 1 [kN/m2]
double        IsvKih2        nonlinear utilization in SCP 2 [kN/m2]
double        IsvKih3        nonlinear utilization in SCP 3 [kN/m2]
double        IsvKih4        nonlinear utilization in SCP 4 [kN/m2]
double        IsvKih5        nonlinear utilization in SCP 5 [kN/m2]
double        IsvKih6        nonlinear utilization in SCP 6 [kN/m2]
double        IsvKih7        nonlinear utilization in SCP 7 [kN/m2]
double        IsvKih8        nonlinear utilization in SCP 8 [kN/m2]
double        IsvKih9        nonlinear utilization in SCP 9 [kN/m2]
double        IsvVymean     mean shear stress in SCPs [kN/m2]
double        IsvVzmean     mean shear stress in SCPs [kN/m2]
long          IsvSmin       IsvSmin = index of SCP with minimum axial stress
long          IsvSmax       IsvSmax = index of SCP with maximum axial stress
long          IsvVmin       IsvVmin - 9 = index of SCP with minimum shear stress
long          IsvVmax       IsvVmax - 9 = index of SCP with maximum shear stress
long          IsvSomin     IsvSomin - 18 = index of SCP with minimum VonMises stress
long          IsvSomax     IsvSomax - 18 = index of SCP with maximum VonMises stress
)

```

NOTE:

SCP - stress calculation point

RSurfaceStressValues = (

double	ssvSxx	Sxx axial stress [kN/m ²]
double	ssvSyy	Syy axial stress [kN/m ²]
double	ssvSxy	Sxy shear stress [kN/m ²]
double	ssvSxz	Sxz shear stress [kN/m ²]
double	ssvSyz	Syz shear stress [kN/m ²]
double	ssvSVM	SVM VonMises stress [kN/m ²]
double	ssvS1	1st principal stress [kN/m ²]
double	ssvS2	2st principal stress [kN/m ²]
double	ssvAs	aS principal direction angle [°]
)	

RSurfaceStressValuesTMB = (

<u>RSurfaceStressValues</u>	ssvTop	stress values at the top of the surface
<u>RSurfaceStressValues</u>	ssvMiddle	stress values in the middle layer of the surface
<u>RSurfaceStressValues</u>	ssvBottom	stress values at the bottom of the surface
)	

RSurfaceStresses = (

long	ContourPointCount	number of vertices of the surface polygon (3 or 4)
long	ContourPoint1Id	1st contour node index
long	ContourPoint2Id	2nd contour node index
long	ContourPoint3Id	3rd contour node index
long	ContourPoint4Id	4th contour node index
long	ContourLine1Id	1st contour line index
long	ContourLine2Id	2nd contour line index
long	ContourLine3Id	3rd contour line index
long	ContourLine4Id	4th contour line index
<u>RSurfaceStressValuesTMB</u>	ssvtmbCenterPoint	stresses at the surface centerpoint
<u>RSurfaceStressValuesTMB</u>	ssvtmbContourPoint1	stresses at the 1st contour node
<u>RSurfaceStressValuesTMB</u>	ssvtmbContourPoint2	stresses at the 2nd contour node
<u>RSurfaceStressValuesTMB</u>	ssvtmbContourPoint3	stresses at the 3rd contour node
<u>RSurfaceStressValuesTMB</u>	ssvtmbContourPoint4	stresses at the 4th contour node
<u>RSurfaceStressValuesTMB</u>	ssvtmbContourLineMidPoint1	stresses at the 1st contour line midpoint
<u>RSurfaceStressValuesTMB</u>	ssvtmbContourLineMidPoint2	stresses at the 2nd contour line midpoint
<u>RSurfaceStressValuesTMB</u>	ssvtmbContourLineMidPoint3	stresses at the 3rd contour line midpoint
<u>RSurfaceStressValuesTMB</u>	ssvtmbContourLineMidPoint4	stresses at the 4th contour line midpoint
)	

RXLAMSurfaceStressValues = (

double	xssvSxx_m_T	stress from moment in local xx direction at top[kN/m ²]
double	xssvSyy_m_T	stress from moment in local yy direction at top[kN/m ²]
double	xssvSxy_m_T	stress from moment in local xy direction at top[kN/m ²]
double	xssvSxx_m_B	stress from moment in local xx direction at bottom[kN/m ²]
double	xssvSyy_m_B	stress from moment in local yy direction at bottom [kN/m ²]
double	xssvSxy_m_B	stress from moment in local xy direction at bottom [kN/m ²]
double	xssvSxx_n	stress from axial force in local xx direction [kN/m ²]
double	xssvSyy_n	stress from axial force in local yy direction [kN/m ²]
double	xssvSxy_n	stress from axial force in local xy direction [kN/m ²]
double	xssvSxz_max	shear stress in local x direction [kN/m ²]
double	xssvSyz_max	shear stress in local y direction [kN/m ²]
double	xssvSrx_max	rolling shear stress $\tau_{rx,max}$ [kN/m ²]
double	xssvSry_max)	rolling shear stress $\tau_{ry,max}$ [kN/m ²]

	RXLAMSurfaceStresses = (
long	ContourPointCount	<i>number of vertices of the surface polygon (3 or 4)</i>
long	ContourPoint1Id	<i>1st contour node index</i>
long	ContourPoint2Id	<i>2nd contour node index</i>
long	ContourPoint3Id	<i>3rd contour node index</i>
long	ContourPoint4Id	<i>4th contour node index</i>
long	ContourLine1Id	<i>1st contour line index</i>
long	ContourLine2Id	<i>2nd contour line index</i>
long	ContourLine3Id	<i>3rd contour line index</i>
long	ContourLine4Id	<i>4th contour line index</i>
RXLAMSurfaceStressValues	xssvCenterPoint	<i>XLAM stresses at the surface centerpoint</i>
RXLAMSurfaceStressValues	xssvContourPoint1	<i>XLAM stresses at the 1st contour node</i>
RXLAMSurfaceStressValues	xssvContourPoint2	<i>XLAM stresses at the 2nd contour node</i>
RXLAMSurfaceStressValues	xssvContourPoint3	<i>XLAM stresses at the 3rd contour node</i>
RXLAMSurfaceStressValues	xssvContourPoint4	<i>XLAM stresses at the 4th contour node</i>
RXLAMSurfaceStressValues	xssvContourLineMidPoint1	<i>XLAM stresses at the 1st contour line midpoint</i>
RXLAMSurfaceStressValues	xssvContourLineMidPoint2	<i>XLAM stresses at the 2nd contour line midpoint</i>
RXLAMSurfaceStressValues	xssvContourLineMidPoint3	<i>XLAM stresses at the 3rd contour line midpoint</i>
RXLAMSurfaceStressValues	xssvContourLineMidPoint4	<i>XLAM stresses at the 4th contour line midpoint</i>
)		

RXLAMSurfaceEfficiencyValues = (

double	xsev_M_N_0	<i>M-N efficiency in grain direction[-]</i>
double	xsev_M_N_90	<i>M-N efficiency perpendicular to grain direction[-]</i>
double	xsev_V_T	<i>shear and torsion efficiency[-]</i>
double	xsev_Vr_N	<i>rolling shear and normal force efficiency[-]</i>
double	xsev_Max	<i>Maximum efficiency [-]</i>
)		

RXLAMSurfaceEfficiencies = (

long	ContourPointCount	<i>number of vertices of the surface polygon (3 or 4)</i>
long	ContourPoint1Id	<i>1st contour node index</i>
long	ContourPoint2Id	<i>2nd contour node index</i>
long	ContourPoint3Id	<i>3rd contour node index</i>
long	ContourPoint4Id	<i>4th contour node index</i>
long	ContourLine1Id	<i>1st contour line index</i>
long	ContourLine2Id	<i>2nd contour line index</i>
long	ContourLine3Id	<i>3rd contour line index</i>
long	ContourLine4Id	<i>4th contour line index</i>
RXLAMSurfaceEfficiencyValues	xsevCenterPoint	<i>XLAM efficiencies at the surface centerpoint</i>
RXLAMSurfaceEfficiencyValues	xsevContourPoint1	<i>XLAM efficiencies at the 1st contour node</i>
RXLAMSurfaceEfficiencyValues	xsevContourPoint2	<i>XLAM efficiencies at the 2nd contour node</i>
RXLAMSurfaceEfficiencyValues	xsevContourPoint3	<i>XLAM efficiencies at the 3rd contour node</i>
RXLAMSurfaceEfficiencyValues	xsevContourPoint4	<i>XLAM efficiencies at the 4th contour node</i>
RXLAMSurfaceEfficiencyValues	xsevContourLineMidPoint1	<i>XLAM efficiencies at the 1st contour line midpoint</i>
RXLAMSurfaceEfficiencyValues	xsevContourLineMidPoint2	<i>XLAM efficiencies at the 2nd contour line midpoint</i>
RXLAMSurfaceEfficiencyValues	xsevContourLineMidPoint3	<i>XLAM efficiencies at the 3rd contour line midpoint</i>
RXLAMSurfaceEfficiencyValues	xsevContourLineMidPoint4	<i>XLAM efficiencies at the 4th contour line midpoint</i>

Functions

Line Stresses

Line stresses can be read only for trusses, beams and ribs. For trusses the only field the [RLineStressValues](#) record retrieves is *lsvS1*.

If LinId refers to a line of other type

- 1) the number of cross-sections will be zero.
- 2) If a single location reader functions is called [deSectionIndexOutOfBounds](#) error code is returned.
- 3) If a single element reader function is called a [deLineHasNoSections](#) error code is returned. If a multiple reader is called the SectionCounts array will contain zero at these line indexes.

Single LOCATION reader functions

long **GetLineStressByLoadCaseId** ([in] long LinId, [in] long SectionId, [in] long LoadCaseId,
[in] long LoadLevelOrTimeStep, [in] EAnalysisType AnalysisType,
[i/o] RLineStressValues Stress, [out] double PosX, [out] BSTR Combination)

LinId line index ($0 < \text{LinId} \leq \text{AxisVMMModel.Lines.Count}$)

SectionId section index ($0 < \text{SectionId} \leq \text{AxisVMMModel.Lines[LinId].SectionCount}$)

LoadCaseId load case index

LoadLevelOrTimeStep for load level ($0 < \text{LoadLevelOrTimeStep} \leq \text{LoadLevelCount}$)

for timestep ($0 < \text{LoadLevelOrTimeStep} \leq \text{TimeStepCount}$)

AnalysisType analysis type

Stress stress results

PosX position of SectionId in m according to the local x direction

Combination name of the load case

Retrieves stresses at a section of a line element. Returns LinId or an error code
([errDatabaseNotReady](#) or see [EStressesError](#)).

long **GetLineStressByLoadCombinationId** ([in] long LinId, [in] long SectionId,
[in] long LoadCombinationId, [in] long LoadLevelOrTimeStep,
[in] EAnalysisType AnalysisType, [i/o] RLineStressValues Stress,
[out] double PosX, [out] BSTR Combination)

LinId line index ($0 < \text{LinId} \leq \text{AxisVMMModel.Lines.Count}$)

SectionId section index ($0 < \text{SectionId} \leq \text{AxisVMMModel.Lines[LinId].SectionCount}$)

LoadCombinationId load combination index

LoadLevelOrTimeStep for load level ($0 < \text{LoadLevelOrTimeStep} \leq \text{LoadLevelCount}$)

for timestep ($0 < \text{LoadLevelOrTimeStep} \leq \text{TimeStepCount}$)

AnalysisType analysis type

Stress stress results

PosX position of SectionId in m according to the local x direction

Combination name of the load case

Retrieves stresses at a section of a line element. Returns LinId or an error code
([errDatabaseNotReady](#) or see [EStressesError](#)).

long **GetEnvelopeLineStress** ([in] long LinId, [in] long SectionId, [in] EMinMaxType MinMaxType,
[in] EAnalysisType AnalysisType, [in] ELineStress Component, [i/o] RLineStressValues Stress,
[out] double PosX, [out] BSTR Combination)

LinId line index ($0 < \text{LinId} \leq \text{AxisVMMModel.Lines.Count}$)

SectionId section index ($0 < \text{SectionId} \leq \text{AxisVMMModel.Lines[LinId].SectionCount}$)

MinMaxType Minimum or maximum value

AnalysisType analysis type

Component Stress component

Stress force results

PosX position of SectionId in m according to the local x direction

Combination load case or combination in which Component has its minimum or
maximum depending on MinMaxType

Retrieves envelope stresses at a section of a line element. Envelope is identified by MinMaxType,
LineStressComponent and EnvelopeUID properties. Stress contains the result of the load case or
combination in which Component is maximal or minimal. Returns LinId or an error code
([errDatabaseNotReady](#) or see [EStressesError](#)).

long	GetEnvelopeLineStress2 ([in] long LinId , [in] long SectionId , [in] EMinMaxType MinMaxType , [in] EAnalysisType AnalysisType , [in] ELineStress Component , [i/o] RLineStressValues Stress , [out] double PosX , [out] long LoadCaseOrCombinationId , [out] long LoadLevel)
	LinId <i>line index (0 < LinId ≤ AxisVMModel.Lines.Count)</i>
	SectionId <i>section index (0 < SectionId ≤ AxisVMModel.Lines[LinId].SectionCount)</i>
	MinMaxType <i>Minimum or maximum value</i>
	AnalysisType <i>analysis type</i>
	Component <i>Stress component</i>
	Stress <i>force results</i>
	PosX <i>position of SectionId in m according to the local x direction</i>
	LoadCaseOrCombinationId <i>load case or load combination index, if index is > IAxisVMLoadcases.count then Load combination index = LoadCaseOrCombinationId – IAxisVMLoadcases.count</i>
	LoadLevel <i>load level</i>
	<i>Retrieves envelope stresses at a section of a line element. Envelope is identified by MinMaxType, LineStressComponent and EnvelopeUID properties. Stress contains the result of the load case or combination in which Component is maximal or minimal. Returns LinId or an error code (errDatabaseNotReady or see EStressesError).</i>
long	GetCriticalLineStress ([in] long LinId , [in] long SectionId , [in] EMinMaxType MinMaxType , [in] ECombinationType CombinationType , [in] EAnalysisType AnalysisType , [in] ELineStress Component , [i/o] RLineStressValues Stress , [out] double PosX , [out] BSTR Combination)
	LinId <i>line index (0 < LinId ≤ AxisVMModel.Lines.Count)</i>
	SectionId <i>section index (0 < SectionId ≤ AxisVMModel.Lines[LinId].SectionCount)</i>
	MinMaxType <i>Minimum or maximum value</i>
	CombinationType <i>combination type</i>
	AnalysisType <i>analysis type</i>
	Component <i>Stress component</i>
	Stress <i>stress results</i>
	PosX <i>position of SectionId in m according to the local x direction</i>
	Combination <i>critical combination in which Component has its minimum or maximum</i>
	<i>Retrieves critical stresses at a section of a line element. Critical combination is identified by Component and MinMaxType properties (e.g. Sxx max). Stress contains the result of the critical combination in which Component is maximal or minimal. Returns LinId or an error code (errDatabaseNotReady or see EStressesError).</i>
long	GetCriticalLineStress2 ([in] long LinId , [in] long SectionId , [in] EMinMaxType MinMaxType , [in] ECombinationType CombinationType , [in] EAnalysisType AnalysisType , [in] ELineStress Component , [i/o] RLineStressValues Stress , [out] double PosX , [out] ECombinationType CriticalCombinationType , [out] SAFEARRAY(double) Factors , [out] SAFEARRAY(long) LoadCasIds)
	LinId <i>line index (0 < LinId ≤ AxisVMModel.Lines.Count)</i>
	SectionId <i>section index (0 < SectionId ≤ AxisVMModel.Lines[LinId].SectionCount)</i>
	MinMaxType <i>Minimum or maximum value</i>
	CombinationType <i>combination type</i>
	AnalysisType <i>analysis type</i>
	Component <i>Stress component</i>
	Stress <i>stress results</i>
	PosX <i>position of SectionId in m according to the local x direction</i>
	CriticalCombinationType <i>combination type corresponding to critical stress</i>
	Factors <i>load factors of the critical load combination</i>
	LoadCasIds <i>load case indexes of the critical load combination</i>
	<i>Retrieves critical stresses at a section of a line element. Critical combination is identified by Component and MinMaxType properties (e.g. Sxx max). Stress contains the result of the critical combination in which Component is maximal or minimal. Returns LinId or an error code (errDatabaseNotReady or see EStressesError).</i>

long	LineStressByLoadCaseld ([in] long Lineld , [in] long SectionId , [i/o] RLineStressValues Stress , [out] double PosX , [out] BSTR Combination) <i>Similar to GetLineStressByLoadCaseld function but, some parameters are set in properties of this interface or IAxisVMResults.</i>
long	LineStressByLoadCombinationId ([in] long Lineld , [in] long SectionId , [i/o] RLineStressValues Stress , [out] double PosX , [out] BSTR Combination) <i>Similar to GetLineStressByLoadCombinationId function but, some parameters are set in properties of this interface or IAxisVMResults.</i>
long	EnvelopeLineStress ([in] long Lineld , [in] long SectionId , [i/o] RLineStressValues Stress , [out] double PosX , [out] BSTR Combination) <i>Similar to GetEnvelopeLineStress function but, some parameters are set in properties of this interface or IAxisVMResults.</i>
long	EnvelopeLineStress2 ([in] long Lineld , [in] long SectionId , [i/o] RLineStressValues Stress , [out] double PosX , [out] long LoadCaseOrCombinationId , [out] long LoadLevel) <i>Similar to GetEnvelopeLineStress2 function but, some parameters are set in properties of this interface or IAxisVMResults.</i>
long	CriticalLineStress ([in] long Lineld , [in] long SectionId , [i/o] RLineStressValues Stress , [out] double PosX , [out] BSTR Combination) <i>Similar to GetCriticalLineStress function but, some parameters are set in properties of this interface or IAxisVMResults.</i>
long	CriticalLineStress2 ([in] long Lineld , [in] long SectionId , [i/o] RLineStressValues Stress , [out] double PosX , [out] ECombinationType CriticalCombinationType , [out] SAFEARRAY(double) Factors , [out] SAFEARRAY(long) LoadCaselds) <i>Similar to GetCriticalLineStress2 function but, some parameters are set in properties of this interface or IAxisVMResults.</i>

Single ELEMENT reader functions

long	LineStressesByLoadCaseld ([in] long Lineld , [i/o] SAFEARRAY(RLineStressValues) Stresses , [out] SAFEARRAY(double) PosX) Lineld line index ($0 < \text{Lineld} \leq \text{AxisVMMModel.Lines.Count}$) Stresses array of line stresses for the line element. Length of the array is SectionCount[Lineld] PosX array of cross-section positions in m according to the local x direction Length of the array is SectionCount[Lineld] <i>Retrieves stresses for each cross-section of a given element according to the LoadCaseld (and LoadLevelOrTimeStep) property. Returns the total number of cross-sections or an error code (errDatabaseNotReady or see EStressesError).</i>
long	LineStressesByLoadCombinationId ([in] long Lineld , [i/o] SAFEARRAY(RLineStressValues) Stresses , [out] SAFEARRAY(double) PosX) Lineld line index ($0 < \text{Lineld} \leq \text{AxisVMMModel.Lines.Count}$) Stresses array of line stresses for the line element. Length of the array is SectionCount[Lineld] PosX array of cross-section positions in m according to the local x direction Length of the array is SectionCount[Lineld] <i>Retrieves stresses for each cross-section of a given element according to the LoadCombinationId (and LoadLevelOrTimeStep) property. Returns the total number of cross-sections or an error code (errDatabaseNotReady or see EStressesError).</i>

long	EnvelopeLineStresses ([in] long Lineld , [i/o] SAFEARRAY(RLineStressValues) Stresses , [out] SAFEARRAY(double) PosX)
	Lineld <i>line index ($0 < \text{Lineld} \leq \text{AxisVMModel.Lines.Count}$)</i> Stresses <i>array of envelope line stresses for the line element.</i> <i>Length of the array is SectionCount[Lineld]</i> PosX <i>array of cross-section positions in m according to the local x direction</i> <i>Length of the array is SectionCount[Lineld]</i>
	<i>Retrieves envelope stresses for each cross-section of a given element. Envelope is identified by MinMaxType, LineStressComponent and EnvelopeUID properties. Stresses array contains the result of the load case or combination in which Component is maximal or minimal. Load case or combination in which Component has its minimum or maximum can be read using the respective single element/single cross-section reader function.</i> <i>Returns the total number of cross-sections or an error code (errDatabaseNotReady or see EStressesError).</i>
long	CriticalLineStresses ([in] long Lineld , [i/o] SAFEARRAY(RLineStressValues) Stresses , [out] SAFEARRAY(double) PosX)
	Lineld <i>line index ($0 < \text{Lineld} \leq \text{AxisVMModel.Lines.Count}$)</i> Stresses <i>array of critical line stresses for the line element.</i> <i>Length of the array is SectionCount[Lineld]</i> PosX <i>array of cross-section positions in m according to the local x direction</i> <i>Length of the array is SectionCount[Lineld]</i>
	<i>Retrieves critical stresses in each cross-section of a given element. Critical combination is identified by Component and MinMaxType properties (e.g. Nx max). Stresses array contains the result of the critical combination in which Component is maximal or minimal. Critical combination in which Component has its minimum or maximum can be read using the respective single element/single cross-section reader function.</i> <i>Returns the total number of cross-sections or an error code (errDatabaseNotReady or see EStressesError).</i>

Multiple element reader functions

long	AllLineStressesByLoadCaseld ([out] SAFEARRAY(long) SectionCounts , [i/o] SAFEARRAY(RLineStressValues) Stresses , [out] SAFEARRAY(double) PosX)
	SectionCounts <i>array containing the section counts of line elements.</i> <i>Length of the array = AxisVMModel.Lines.Count</i> Stresses <i>array of line stresses.</i> <i>For each line element it contains results for SectionCount[i] sections consecutively.</i> <i>Length of the array is the sum of the SectionCounts array elements</i> PosX <i>array of cross-section positions in m according to the local x direction of each line element</i> <i>Length of the array is the sum of the SectionCounts array elements</i>
	<i>Retrieves stresses of all lines and cross-sections consecutively according to the LoadCaseld (and LoadLevelOrTimeStep) property. Returns the total number of records in the array or an error code (errDatabaseNotReady or see EStressesError).</i>

long	AllLineStressesByLoadCombinationId ([out] SAFEARRAY(long) SectionCounts , [i/o] SAFEARRAY(RLineStressValues) Stresses , [out] SAFEARRAY(double) PosX)
	<p>SectionCounts array containing the section counts of line elements. Length of the array = AxisVMMModel.Lines.Count</p> <p>Stresses array of line stresses. For each line element it contains results for SectionCount[i] sections consecutively. Length of the array is the sum of the SectionCounts array elements</p> <p>PosX array of cross-section positions in m according to the local x direction of each line element Length of the array is the sum of the SectionCounts array elements</p> <p>Retrieves stresses of all lines and cross-sections consecutively according to the LoadCombinationId (and LoadLevelOrTimeStep) property. Returns the total number of records in the array or an error code (errDatabaseNotReady or see EStressesError).</p>
long	AllEnvelopeLineStresses ([out] SAFEARRAY(long) SectionCounts , [i/o] SAFEARRAY(RLineStressValues) Stresses , [out] SAFEARRAY(double) PosX)
	<p>SectionCounts array containing the section counts of line elements. Length of the array = AxisVMMModel.Lines.Count</p> <p>Stresses array of envelope line stresses for all line elements. For each line element it contains results for SectionCount[i] sections consecutively. Length of the array is the sum of the SectionCounts array elements</p> <p>PosX array of cross-section positions in m according to the local x direction of each line element Length of the array is the sum of the SectionCounts array elements</p> <p>Retrieves envelope stresses of all line elements. Envelope is identified by MinMaxType, LineStressComponent and EnvelopeUID properties. Stresses array contains the result of the load case or combination in which Component is maximal or minimal. Load case or combination in which Component has its minimum or maximum can be read using the respective single element/single cross-section reader function.</p> <p>Returns the total number of cross-sections or an error code (errDatabaseNotReady or see EStressesError).</p>
long	AllCriticalLineStresses ([out] SAFEARRAY(long) SectionCounts , [i/o] SAFEARRAY(RLineStressValues) Stresses , [out] SAFEARRAY(double) PosX)
	<p>SectionCounts array containing the section counts of line elements. Length of the array = AxisVMMModel.Lines.Count</p> <p>Stresses array of critical line stresses for all line elements. For each line element it contains results for SectionCount[i] sections consecutively. Length of the array is the sum of the SectionCounts array elements</p> <p>PosX array of cross-section positions in m according to the local x direction of each line element Length of the array is the sum of the SectionCounts array elements</p> <p>Retrieves critical stresses of all line elements. Critical combination is identified by Component and MinMaxType properties (e.g. Nx max). Stresses array contains the result of the critical combination in which Component is maximal or minimal. Critical combination in which Component has its minimum or maximum can be read using the respective single element/single cross-section reader function.</p> <p>Returns the total number of cross-sections or an error code (errDatabaseNotReady or see EStressesError).</p>

Multiple BLOCK reader functions

long **LineStressesForResultBlocks** ([in] long **Lineld**, [in] long **SectionId**,
[i/o] SAFEARRAY([RResultBlockInfo](#)) **ResultBlockInfo**,
[i/o] SAFEARRAY([RLineStressValues](#)) **Stresses**, [out] SAFEARRAY(double) **PosX**)

ResultBlockInfo array of description records for result blocks
Stresses stresses results for all result blocks
PosX array of cross-section positions in m according to the local x direction of each line element

Retrieves stresses at the given line and cross-section in all result blocks consecutively. The number of result blocks depends on the **AnalysisType** property.

Returns the common length of **ResultBlockInfo**, **Stresses** and **PosX** arrays or an error code ([errDatabaseNotReady](#) or see [EStressesError](#)).

Surface Stresses

Single location reader functions retrieve stresses at a certain point of a surface identified by **SurfaceVertexType** (svtContourPoint, svtContourLineMidPoint or svtCenterPoint), **SurfaceVertexId** and **SurfaceStressPosition** (sspTop, sspMiddle, sspBottom).

SurfaceVertexType	Meaning of SurfaceVertexId
svtContourPoint	node index ($0 < \text{SurfaceVertexId} \leq \text{AxisVMNodes.Count}$)
svtContourLinePoint	line index ($0 < \text{SurfaceVertexId} \leq \text{AxisVMLines.Count}$)
svtCenterPoint	-

Single element reader functions retrieve stresses in all available points of a surface. If the surface element is triangular the *ContourPointCount* field of [RSurfaceForces](#) = 3, record fields *ContourPoint4Id*, *ContourLine4Id*, *sfvContourPoint4* and *sfvContourLineMidPoint4* contain no meaningful values. If the surface element is quadrilateral the *ContourPointCount* field of [RSurfaceForces](#) = 4 and each field has a meaningful value.

Single LOCATION reader functions

long **GetSurfaceStressByLoadCaseld** ([in] long **Surfaceld**,
[in] [ESurfaceVertexType](#) **SurfaceVertexType**, [in] long **SurfaceVertexId**,
[in] [ESurfaceStressPosition](#) **SurfaceStressPosition**, [in] long **LoadCaseld**,
[in] long **LoadLevelOrTimeStep**, [in] [EAnalysisType](#) **AnalysisType**,
[i/o] [RSurfaceStressValues](#) **Stress**, [out] BSTR **Combination**)

Surfaceld surface index ($0 < \text{Surfaceld} \leq \text{AxisVMSurfaces.Count}$)
SurfaceVertexType vertex type
SurfaceVertexId vertex identifier, see [here](#)
SurfaceStressPosition layer of the surface element (top, middle, bottom)
LoadCaseld load case index
LoadLevelOrTimeStep for load level ($0 < \text{LoadLevelOrTimeStep} \leq \text{LoadLevelCount}$)
for timestep ($0 < \text{LoadLevelOrTimeStep} \leq \text{TimeStepCount}$)
AnalysisType analysis type
Stress stress results
Combination name of the load case

Retrieves stresses at a point and in one layer of a surface. Returns **Surfaceld** or an error code ([errDatabaseNotReady](#) or see [EStressesError](#)).

long **GetSurfaceStressByLoadCombinationId** ([in] long **Surfaceld**,
[in] [ESurfaceVertexType](#) **SurfaceVertexType**, [in] long **SurfaceVertexId**,
[in] [ESurfaceStressPosition](#) **SurfaceStressPosition**, [in] long **LoadCombinationId**,
[in] long **LoadLevel**, [in] [EAnalysisType](#) **AnalysisType**, [i/o] [RSurfaceStressValues](#) **Stress**,
[out] BSTR **Combination**)

Surfaceld	<i>surface index ($0 < Surfaceld \leq \text{AxisVMSurfaces.Count}$)</i>
SurfaceVertexType	<i>vertex type</i>
SurfaceVertexId	<i>vertex identifier, see here</i>
SurfaceStressPosition	<i>layer of the surface element (top, middle, bottom)</i>
LoadCombinationId	<i>load combination index</i>
LoadLevel	<i>for load level ($0 < LoadLevelOrTimeStep \leq \text{LoadLevelCount}$)</i>
AnalysisType	<i>analysis type</i>
Stress	<i>stress results</i>
Combination	<i>name of the load combination</i>

Retrieves stresses at a point and in one layer of a surface according to the LoadCaseld (and LoadLevelOrTimeStep) property. Returns Surfaceld or an error code ([errDatabaseNotReady](#) or see [EStressesError](#)).

long **GetEnvelopeSurfaceStress** ([in] long **Surfaceld**, [in] [ESurfaceVertexType](#) **SurfaceVertexType**,
[in] long **SurfaceVertexId**, [in] [ESurfaceStressPosition](#) **SurfaceStressPosition**,
[in] [EMinMaxType](#) **MinMaxType**, [in] [EAnalysisType](#) **AnalysisType**,
[in] [ESurfaceStress](#) **Component**, [i/o] [RSurfaceStressValues](#) **Stress**,
[out] BSTR **Combination**)

Surfaceld	<i>surface index ($0 < Surfaceld \leq \text{AxisVMSurfaces.Count}$)</i>
SurfaceVertexType	<i>vertex type</i>
SurfaceVertexId	<i>vertex identifier, see here</i>
SurfaceStressPosition	<i>layer of the surface element (top, middle, bottom)</i>
MinMaxType	<i>Minimum or maximum value</i>
AnalysisType	<i>analysis type</i>
Component	<i>surface stress component</i>
Stress	<i>stress results</i>
Combination	<i>load case or combination in which Component has its minimum or maximum</i>

Retrieves envelope stresses at one point and in one layer of a surface element. Envelope is identified by MinMaxType, SurfaceStressComponent and EnvelopeUID properties. Stress contains the result of the load case or combination in which Component is maximal or minimal. Returns Surfaceld or an error code ([errDatabaseNotReady](#) or see [EStressesError](#)).

long **GetEnvelopeSurfaceStress2** ([in] long **Surfaceld**, [in] [ESurfaceVertexType](#) **SurfaceVertexType**,
[in] long **SurfaceVertexId**, [in] [ESurfaceStressPosition](#) **SurfaceStressPosition**,
[in] [EMinMaxType](#) **MinMaxType**, [in] [EAnalysisType](#) **AnalysisType**,
[in] [ESurfaceStress](#) **Component**, [i/o] [RSurfaceStressValues](#) **Stress**,
[out] long **LoadCaseOrCombinationId**, [out] long **LoadLevel**)

Surfaceld	<i>surface index ($0 < Surfaceld \leq AxisVMSurfaces.Count$)</i>
SurfaceVertexType	<i>vertex type</i>
SurfaceVertexId	<i>vertex identifier, see here</i>
SurfaceStressPosition	<i>layer of the surface element (top, middle, bottom)</i>
MinMaxType	<i>Minimum or maximum value</i>
AnalysisType	<i>analysis type</i>
Component	<i>surface stress component</i>
Stress	<i>stress results</i>
LoadCaseOrCombinationId	<i>load case or load combination index, if index is > IAxisVMLoadcases.Count then $Load\ combination\ index = LoadCaseOrCombinationId - IAxisVMLoadcases.Count$</i>
LoadLevel	<i>load case or combination in which Component has its minimum or maximum</i>

Retrieves envelope stresses at one point and in one layer of a surface element. Envelope is identified by MinMaxType, SurfaceStressComponent and EnvelopeUID properties. Stress contains the result of the load case or combination in which Component is maximal or minimal. Returns Surfaceld or an error code ([errDatabaseNotReady](#) or see [EStressesError](#)).

long **GetCriticalSurfaceStress** ([in] long **Surfaceld**, [in] [ESurfaceVertexType](#) **SurfaceVertexType**,
[in] long **SurfaceVertexId**, [in] [ESurfaceStressPosition](#) **SurfaceStressPosition**,
[in] [EMinMaxType](#) **MinMaxType**, [in] [ECombinationType](#) **CombinationType**,
[in] [EAnalysisType](#) **AnalysisType**, [in] [ESurfaceStress](#) **Component**,
[i/o] [RSurfaceForceValues](#) **Stress**, [out] BSTR **Combination**)

Surfaceld	<i>surface index ($0 < Surfaceld \leq AxisVMSurfaces.Count$)</i>
SurfaceVertexType	<i>vertex type</i>
SurfaceVertexId	<i>vertex identifier, see here</i>
SurfaceStressPosition	<i>layer of the surface element (top, middle, bottom)</i>
MinMaxType	<i>Minimum or maximum value</i>
CombinationType	<i>combination type</i>
AnalysisType	<i>analysis type</i>
Component	<i>surface stress component</i>
Stress	<i>stress results</i>
Combination	<i>critical combination in which Component has its minimum or maximum</i>

Retrieves critical stresses at one point and in one layer of a surface element. Critical combination is identified by Component and MinMaxType properties (e.g. Sxy max). Stress contains the result of the critical combination in which Component is maximal or minimal. Returns Surfaceld or an error code ([errDatabaseNotReady](#) or see [EStressesError](#)).

long	GetCriticalSurfaceStress2 ([in] long Surfaceld , [in] ESurfaceVertexType SurfaceVertexType , [in] long SurfaceVertexId , [in] ESurfaceStressPosition SurfaceStressPosition , [in] EMinMaxType MinMaxType , [in] ECombinationType CombinationType , [in] EAnalysisType AnalysisType , [in] ESurfaceStress Component , [i/o] RSurfaceForceValues Stress , [out] ECombinationType CriticalCombinationType , [out] SAFEARRAY(double) Factors , [out] SAFEARRAY(long) LoadCaselds)
	Surfaceld <i>surface index ($0 < \text{Surfaceld} \leq \text{AxisVMSurfaces.Count}$)</i>
	SurfaceVertexType <i>vertex type</i>
	SurfaceVertexId <i>vertex identifier, see here</i>
	SurfaceStressPosition <i>layer of the surface element (top, middle, bottom)</i>
	MinMaxType <i>Minimum or maximum value</i>
	CombinationType <i>combination type</i>
	AnalysisType <i>analysis type</i>
	Component <i>surface stress component</i>
	Stress <i>stress results</i>
	CriticalCombinationType <i>combination type corresponding to the critical load combination</i>
	Factors <i>load factors of the critical load combination</i>
	LoadCaselds <i>load case indexes of the critical load combination</i>
	<i>Retrieves critical stresses at one point and in one layer of a surface element. Critical combination is identified by Component and MinMaxType properties (e.g. Sxy max). Stress contains the result of the critical combination in which Component is maximal or minimal. Returns Surfaceld or an error code (errDatabaseNotReady or see EStressesError).</i>
long	SurfaceStressByLoadCaseld ([in] long Surfaceld , [in] ESurfaceVertexType SurfaceVertexType , [in] long SurfaceVertexId , [in] ESurfaceStressPosition SurfaceStressPosition , [i/o] RSurfaceStressValues Stress , [out] BSTR Combination)
	<i>Similar to GetSurfaceStressByLoadCaseld function but, some parameters are set in properties of this interface or IAxisVMResults.</i>
long	SurfaceStressByLoadCombinationId ([in] long Surfaceld , [in] ESurfaceVertexType SurfaceVertexType , [in] long SurfaceVertexId , [in] ESurfaceStressPosition SurfaceStressPosition , [i/o] RSurfaceStressValues Stress , [out] BSTR Combination)
	<i>Similar to GetSurfaceStressByLoadCombinationId function but, some parameters are set in properties of this interface or IAxisVMResults.</i>
long	EnvelopeSurfaceStress ([in] long Surfaceld , [in] ESurfaceVertexType SurfaceVertexType , [in] long SurfaceVertexId , [in] ESurfaceStressPosition SurfaceStressPosition , [i/o] RSurfaceStressValues Stress , [out] BSTR Combination)
	<i>Similar to GetEnvelopeSurfaceStress function but, some parameters are set in properties of this interface or IAxisVMResults.</i>
long	EnvelopeSurfaceStress2 ([in] long Surfaceld , [in] ESurfaceVertexType SurfaceVertexType , [in] long SurfaceVertexId , [in] ESurfaceStressPosition SurfaceStressPosition , [i/o] RSurfaceStressValues Stress , [out] long LoadCaseOrCombinationId , [out] long LoadLevel)
	<i>Similar to GetEnvelopeSurfaceStress2 function but, some parameters are set in properties of this interface or IAxisVMResults.</i>
long	CriticalSurfaceStress ([in] long Surfaceld , [in] ESurfaceVertexType SurfaceVertexType , [in] long SurfaceVertexId, [in] ESurfaceStressPosition SurfaceStressPosition , [i/o] RSurfaceForceValues Stress , [out] BSTR Combination)
	<i>Similar to GetCriticalSurfaceStress function but, some parameters are set in properties of this interface or IAxisVMResults.</i>

long **CriticalSurfaceStress2** ([in] long **Surfaceld**, [in] [ESurfaceVertexType](#) **SurfaceVertexType**,
[in] long **SurfaceVertexId**, [in] [ESurfaceStressPosition](#) **SurfaceStressPosition**,
[i/o] [RSurfaceForceValues](#) **Stress**,
[out] [ECombinationType](#) **CriticalCombinationType**, [out] SAFEARRAY(double) **Factors**,
[out] SAFEARRAY(long) **LoadCaselds**)

Similar to *GetCriticalSurfaceStress2* function but, some parameters are set in properties of this interface or [IAxisVMResults](#).

Single ELEMENT reader functions

long **SurfaceStressesByLoadCaseld** ([in] long **Surfaceld**, [i/o] [RSurfaceStresses](#) **Stresses**)
Surfaceld surface index ($0 < \text{Surfaceld} \leq \text{AxisVMMModel.Surfaces.Count}$)
Stresses surface stresses on the element

Retrieves stresses in all available points and layers (top, middle, bottom) of a surface element according to the *LoadCaseld* (and *LoadLevelOrTimeStep*) property. Returns *Surfaceld* or an error code ([errDatabaseNotReady](#) or see [EStressesError](#)).

long **SurfaceStressesByLoadCombinationId** ([in] long **Surfaceld**,
[i/o] [RSurfaceStresses](#) **Stresses**)
Surfaceld surface index ($0 < \text{Surfaceld} \leq \text{AxisVMMModel.Surfaces.Count}$)
Stresses surface stresses on the element

Retrieves stresses in all available points and layers (top, middle, bottom) of a surface element according to the *LoadCombinationId* (and *LoadLevelOrTimeStep*) property. Returns *Surfaceld* or an error code ([errDatabaseNotReady](#) or see [EStressesError](#)).

long **EnvelopeSurfaceStresses** ([in] long **Surfaceld**, [i/o] [RSurfaceStresses](#) **Stresses**)
Surfaceld surface index ($0 < \text{Surfaceld} \leq \text{AxisVMMModel.Surfaces.Count}$)
Stresses surface stresses on the element

Retrieves envelope stresses in all available points and layers (top, middle, bottom) of a surface element. Envelope is identified by *MinMaxType*, *SurfaceStressComponent* and *EnvelopeUID* properties. At each point *Stresses* contain the result of the load case or combination in which Component is maximal or minimal. Returns *Surfaceld* or an error code ([errDatabaseNotReady](#) or see [EStressesError](#)).

long **CriticalSurfaceStresses** ([in] long **Surfaceld**, [i/o] [RSurfaceStresses](#) **Stresses**)
Surfaceld surface index ($0 < \text{Surfaceld} \leq \text{AxisVMMModel.Surfaces.Count}$)
Stresses surface stresses on the element

Retrieves critical stresses in all available points and layers (top, middle, bottom) of a surface element. Critical combination is identified by *Component* and *MinMaxType* properties (e.g. Sxy max). At each point *Stresses* contain the result of the critical combination in which Component is maximal or minimal. Returns *Surfaceld* or an error code ([errDatabaseNotReady](#) or see [EStressesError](#)).

Multiple element reader functions

long **AllSurfaceStressesByLoadCaseld** ([i/o] SAFEARRAY([RSurfaceStresses](#)) **Stresses**)
Stresses array of surface forces.
Length of the array is *AxisVMMModel.Surfaces.Count*

Retrieves stresses in all available points and layers (top, middle, bottom) of all surface elements in an array according to the *LoadCaseld* (and *LoadLevelOrTimeStep*) property. Returns the total number of records in the array or an error code ([errDatabaseNotReady](#) or see [EStressesError](#)).

long **AllSurfaceStressesByLoadCombinationId** ([i/o] SAFEARRAY([RSurfaceStresses](#)) **Stresses**)

Stresses array of surface forces.

Length of the array is AxisVMModel.Surfaces.Count

Retrieves stresses in all available points and layers (top, middle, bottom) of all surface elements in an array according to the LoadCombinationId (and LoadLevelOrTimeStep) property. Returns the total number of records in the array or an error code ([errDatabaseNotReady](#) or see [EStressesError](#)).

long **AllEnvelopeSurfaceStresses** ([i/o] SAFEARRAY([RSurfaceStresses](#)) **Stresses**)

Stresses array of surface forces.

Length of the array is AxisVMModel.Surfaces.Count

Retrieves envelope stresses in all available points and layers (top, middle, bottom) of all surface elements. Envelope is identified by MinMaxType, SurfaceStressComponent and EnvelopeUID properties. Stresses array contains the result of the load case or combination in which Component is maximal or minimal. Load case or combination in which Component has its minimum or maximum can be read using the respective single location reader function.

Returns the total number of records in the array or an error code ([errDatabaseNotReady](#) or see [EStressesError](#)).

long **AllCriticalSurfaceStresses** ([i/o] SAFEARRAY([RSurfaceStresses](#)) **Stresses**)

Stresses array of surface forces.

Length of the array is AxisVMModel.Surfaces.Count

Retrieves critical stresses in all available points and layers (top, middle, bottom) of all surface elements. Critical combination is identified by Component and MinMaxType properties (e.g. Sxy max). Stresses array contains the result of the load case or combination in which Component is maximal or minimal. Load case or combination in which Component has its minimum or maximum can be read using the respective single location reader function.

Returns the total number of records in the array or an error code ([errDatabaseNotReady](#) or see [EStressesError](#)).

Multiple BLOCK reader functions

long **SurfaceStressesForResultBlocks** ([in] long **Surfaceld**,
[i/o] SAFEARRAY([RResultBlockInfo](#)) **ResultBlockInfo**,
[i/o] SAFEARRAY([RSurfaceStresses](#)) **Stresses**)

Surfaceld surface index ($0 < \text{Surfaceld} \leq \text{AxisVMModel.Surfaces.Count}$)

ResultBlockInfo array of description records for result blocks

Stresses stress results for all result blocks

Retrieves stresses on the given surface element in all result blocks consecutively. The number of result blocks depends on the AnalysisType property.

Returns the common length of ResultBlockInfo and Stresses arrays or an error code ([errDatabaseNotReady](#) or see [EStressesError](#)).

long **SurfaceStressValuesForResultBlocks** ([in] long **Surfaceld**,

[in] [ESurfaceVertexType](#) **SurfaceVertexType**, [in] long **SurfaceVertexId**,

[in] [ESurfaceStressPosition](#) **SurfaceStressPosition**,

[i/o] SAFEARRAY([RResultBlockInfo](#)) **ResultBlockInfo**,

[i/o] SAFEARRAY([RSurfaceStressValues](#)) **Stresses**)

Surfaceld surface index ($0 < \text{Surfaceld} \leq \text{AxisVMModel.Surfaces.Count}$)

SurfaceVertexType vertex type

SurfaceVertexId vertex identifier, see [here](#)

SurfaceStressPosition layer of the surface element (top, middle, bottom)

ResultBlockInfo array of description records for result blocks

Stresses stress results for all result blocks

Retrieves stresses at one point and in one layer of a surface element in all result blocks consecutively. The number of result blocks depends on the AnalysisType property.

Returns the common length of ResultBlockInfo and Stresses arrays or an error code ([errDatabaseNotReady](#) or see [EStressesError](#)).

Member Stresses

long **GetMemberStressesByLoadCaseId** ([in] long **MemberID**, [in] long **LoadCaseId**,
[in] long **LoadLevelOrTimeStep**, [in] [EAnalysisType](#) **AnalysisType**,
[out] SAFEARRAY([RLineStressValues](#)) **Stresses**, [out] SAFEARRAY(double) **PosX**)

MemberID member index ($0 < MemberId \leq AxisVMMModel.Members.Count$)

LoadCaseId load case index

LoadLevelOrTimeStep load level or time step, according to the type of analysis. See
[LoadLevelOrModeShapeOrTimeStep](#) parameter.

AnalysisType analysis type

Stresses array of member stresses for the line element.

PosX array of cross-section positions in m according to the local x direction

Retrieves stresses for each cross-section of a given element according to the LoadCaseId (and LoadLevelOrTimeStep) property. Returns the total number of cross-sections or an error code ([errDatabaseNotReady](#) or see [EStressesError](#)).

long **GetMemberStressesByLoadCombinationId** ([in] long **MemberID**, [in] long **LoadCombinationId**,
[in] long **LoadLevelOrTimeStep**, [in] [EAnalysisType](#) **AnalysisType**,
[out] SAFEARRAY([RLineStressValues](#)) **Stresses**, [out] SAFEARRAY(double) **PosX**)

MemberID member index ($0 < MemberId \leq AxisVMMModel.Members.Count$)

LoadCombinationId load combination index

LoadLevelOrTimeStep load level or time step, according to the type of analysis. See
[LoadLevelOrModeShapeOrTimeStep](#) parameter.

AnalysisType analysis type

Stresses array of member stresses for the line element.

PosX array of cross-section positions in m according to the local x direction

Retrieves stresses for each cross-section of a given element according to the LoadCombinationId (and LoadLevelOrTimeStep) property. Returns the total number of cross-sections or an error code ([errDatabaseNotReady](#) or see [EStressesError](#)).

long **GetEnvelopeMemberStresses** ([in] long **MemberID**, [in] long **EnvelopeUID**,
[in] [EMinMaxType](#) **MinMaxType**, [in] [EAnalysisType](#) **AnalysisType**, [in] [ELineStress](#) **Component**,
[out] SAFEARRAY([RLineStressValues](#)) **Stresses**, [out] SAFEARRAY(double) **PosX**)

MemberID member index ($0 < MemberId \leq AxisVMMModel.Members.Count$)

EnvelopeUID unique envelope index

MinMaxType Minimum or maximum value

AnalysisType analysis type

Component Stress component

Stresses array of member stresses for the line element.

PosX array of cross-section positions in m according to the local x direction

Retrieves envelope stresses for each cross-section of a member. Envelope is identified by MinMaxType, LineStressComponent and EnvelopeUID properties. Stresses array contains the result of the load case or combination in which Component is maximal or minimal. Load case or combination in which Component has its minimum or maximum can be read using the respective single element/single cross-section reader function.

Returns the total number of cross-sections or an error code ([errDatabaseNotReady](#) or see [EStressesError](#)).

long	GetCriticalMemberStresses ([in] long MemberID , [in] EMinMaxType MinMaxType , [in] ECombinationType CombinationType , [in] EAnalysisType AnalysisType , [in] ELineStress Component , [out] SAFEARRAY(RLineStressValues) Stresses , [out] SAFEARRAY(double) PosX)
	MemberID <i>member index ($0 < MemberId \leq AxisVMMModel.Members.Count$)</i>
	MinMaxType <i>Minimum or maximum value</i>
	CombinationType <i>combination type</i>
	AnalysisType <i>analysis type</i>
	Component <i>Stress component</i>
	Stresses <i>array of member stresses for the line element.</i>
	PosX <i>array of cross-section positions in m according to the local x direction</i>
	<i>Retrieves critical stresses in each cross-section of a given element. Critical combination is identified by Component and MinMaxType properties (e.g. IsSmin). Stresses array contains the result of the critical combination in which Component is maximal or minimal. Critical combination in which Component has its minimum or maximum can be read using the respective single element/single cross-section reader function.</i>
	<i>Returns the total number of cross-sections or an error code (errDatabaseNotReady or see EStressesError).</i>

XLAM panel stresses

long	GetXLAMSurfaceStressByLoadCaseId ([in] long Surfaceld , [in] ESurfaceVertexType SurfaceVertexType , [in] long SurfaceVertexId , [in] long LoadCaseId , [in] long LoadLevelOrTimeStep , [in] EAnalysisType AnalysisType , [i/o] RXLAMSurfaceStressValues Stress , [out] BSTR Combination)
	Surfaceld <i>surface index ($0 < Surfaceld \leq AxisVMSurfaces.Count$)</i>
	SurfaceVertexType <i>vertex type</i>
	SurfaceVertexId <i>vertex identifier, see here</i>
	LoadCaseId <i>load case index</i>
	LoadLevelOrTimeStep <i>for load level ($0 < LoadLevelOrTimeStep \leq LoadLevelCount$)</i> <i>for timestep ($0 < LoadLevelOrTimeStep \leq TimeStepCount$)</i>
	AnalysisType <i>analysis type</i>
	Stress <i>XLAM surface stress results</i>
	Combination <i>name of the load case</i>
	<i>Retrieves stresses at a point of a surface. Returns Surfaceld or an error code (EGeneralError or see EStressesError).</i>
long	GetXLAMSurfaceStressByLoadCombinationId ([in] long Surfaceld , [in] ESurfaceVertexType SurfaceVertexType , [in] long SurfaceVertexId , [in] long LoadCombinationId , [in] long LoadLevel , [in] EAnalysisType AnalysisType , [i/o] RXLAMSurfaceStressValues Stress , [out] BSTR Combination)
	Surfaceld <i>surface index ($0 < Surfaceld \leq AxisVMSurfaces.Count$)</i>
	SurfaceVertexType <i>vertex type</i>
	SurfaceVertexId <i>vertex identifier, see here</i>
	LoadCombinationId <i>load combination index</i>
	LoadLevel <i>for load level ($0 < LoadLevelOrTimeStep \leq LoadLevelCount$)</i>
	AnalysisType <i>analysis type</i>
	Stress <i>XLAM surface stress results</i>
	Combination <i>name of the load case</i>
	<i>Retrieves XLAM stresses at a point of a surface. Returns Surfaceld or an error code (EGeneralError or see EStressesError).</i>

long	GetEnvelopeXLAMSurfaceStress ([in] long Surfaceld , [in] long EnvelopeUID , [in] ESurfaceVertexType SurfaceVertexType , [in] long SurfaceVertexId , [in] EMinMaxType MinMaxType , [in] EAnalysisType AnalysisType , [in] EXLAMSurfaceStress Component , [i/o] RXLAMSurfaceStressValues Stress , [out] long LoadCaseOrCombinationId , [out] long LoadLevel)
	Surfaceld surface index ($0 < \text{Surfaceld} \leq \text{AxisVMSurfaces.Count}$) EnvelopeUID unique envelope index SurfaceVertexType vertex type SurfaceVertexId vertex identifier, see here MinMaxType Minimum or maximum value AnalysisType analysis type Component surface stress component Stress XLAM surface stress results LoadCaseOrCombinationId load case or load combination index of surface midpoint, if index is > IAxisVMLoadcases .count then Load combination index = LoadCaseOrCombinationId - IAxisVMLoadcases .count LoadLevel load level
	<i>Retrieves envelope stresses at one point of a surface element. Envelope is identified by MinMaxType, Component and EnvelopeUID properties. XLAM stress contains the result of the load case or combination in which Component is maximal or minimal. Returns Surfaceld or an error code (EGeneralError or see EStressesError).</i>
long	GetCriticalXLAMSurfaceStress ([in] long Surfaceld , [in] ESurfaceVertexType SurfaceVertexType , [in] long SurfaceVertexId , [in] EMinMaxType MinMaxType , [in] ECombinationType CombinationType , [in] EAnalysisType AnalysisType , [in] EXLAMSurfaceStress Component , [i/o] RXLAMSurfaceStressValues Stress , [out] ECombinationType CriticalCombinationType , [out] SAFEARRAY(double) Factors , [out] SAFEARRAY(long) LoadCaselds)
	Surfaceld surface index ($0 < \text{Surfaceld} \leq \text{AxisVMSurfaces.Count}$) SurfaceVertexType vertex type SurfaceVertexId vertex identifier, see here MinMaxType Minimum or maximum value CombinationType combination type AnalysisType analysis type Component surface stress component Stress XLAM surface stress results CriticalCombinationType combination type corresponding to the critical load combination Factors load factors of the critical load combination LoadCaselds load case indexes of the critical load combination
	<i>Retrieves critical XLAM stresses at one point of a surface element. Critical combination is identified by Component and MinMaxType properties (e.g. XSSSxx_m_T). Stress contains the result of the critical combination in which Component is maximal or minimal. Returns Surfaceld or an error code (EGeneralError or see EStressesError).</i>

long	GetXLAMSurfaceStressValuesForResultBlocks ([in] long Surfaceld , [in] ESurfaceVertexType SurfaceVertexType , [in] long SurfaceVertexId , [in] EAnalysisType AnalysisType , [out] SAFEARRAY(RResultBlockInfo) ResultBlockInfo , [out] SAFEARRAY(RXLAMSurfaceStressValues) Stresses)
	<p style="margin-left: 20px;">Surfaceld surface index ($0 < \text{Surfaceld} \leq \text{AxisVMSurfaces.Count}$)</p> <p style="margin-left: 20px;">SurfaceVertexType vertex type</p> <p style="margin-left: 20px;">SurfaceVertexId vertex identifier, see here</p> <p style="margin-left: 20px;">AnalysisType analysis type</p> <p style="margin-left: 20px;">ResultBlockInfo array of description records for result blocks</p> <p style="margin-left: 20px;">Stresses XLAM surface stress results</p>
	<p><i>Retrieves XLAM stresses at one point of a surface element in all result blocks consecutively. The number of result blocks depends on the AnalysisType property.</i></p> <p><i>Returns the common length of ResultBlockInfo and Stresses arrays or an error code (EGeneralError or see EStressesError).</i></p>
long	GetXLAMSurfaceStressesByLoadCaseld ([in] long Surfaceld , [in] long LoadCaseld , [in] long LoadLevelOrTimeStep , [in] EAnalysisType AnalysisType , [i/o] RXLAMSurfaceStresses Stresses , [out] BSTR Combination)
	<p style="margin-left: 20px;">Surfaceld surface index ($0 < \text{Surfaceld} \leq \text{AxisVMSurfaces.Count}$)</p> <p style="margin-left: 20px;">LoadCaseld load case index</p> <p style="margin-left: 20px;">LoadLevelOrTimeStep for load level ($0 < \text{LoadLevelOrTimeStep} \leq \text{LoadLevelCount}$) for timestep ($0 < \text{LoadLevelOrTimeStep} \leq \text{TimeStepCount}$)</p> <p style="margin-left: 20px;">AnalysisType analysis type</p> <p style="margin-left: 20px;">Stresses XLAM surface stress results</p> <p style="margin-left: 20px;">Combination name of the load case</p>
	<p><i>Retrieves stresses of a surface. Returns Surfaceld or an error code (EGeneralError or see EStressesError).</i></p>
long	GetXLAMSurfaceStressesByLoadCombinationId ([in] long Surfaceld , [in] long LoadCombinationId , [in] long LoadLevel , [in] EAnalysisType AnalysisType , [i/o] RXLAMSurfaceStresses Stresses , [out] BSTR Combination)
	<p style="margin-left: 20px;">Surfaceld surface index ($0 < \text{Surfaceld} \leq \text{AxisVMSurfaces.Count}$)</p> <p style="margin-left: 20px;">LoadCombinationId load combination index</p> <p style="margin-left: 20px;">LoadLevel for load level ($0 < \text{LoadLevelOrTimeStep} \leq \text{LoadLevelCount}$)</p> <p style="margin-left: 20px;">AnalysisType analysis type</p> <p style="margin-left: 20px;">Stresses XLAM surface stress results</p> <p style="margin-left: 20px;">Combination name of the load case</p>
	<p><i>Retrieves stresses of a surface. Returns Surfaceld or an error code (EGeneralError or see EStressesError).</i></p>
long	GetEnvelopeXLAMSurfaceStresses ([in] long Surfaceld , [in] long EnvelopeUID , [in] EMinMaxType MinMaxType , [in] EAnalysisType AnalysisType , [in] EXLAMSurfaceStress Component , [i/o] RXLAMSurfaceStresses Stresses)
	<p style="margin-left: 20px;">Surfaceld surface index ($0 < \text{Surfaceld} \leq \text{AxisVMSurfaces.Count}$)</p> <p style="margin-left: 20px;">EnvelopeUID unique envelope index</p> <p style="margin-left: 20px;">MinMaxType Minimum or maximum value</p> <p style="margin-left: 20px;">AnalysisType analysis type</p> <p style="margin-left: 20px;">Component surface stress component</p> <p style="margin-left: 20px;">Stresses XLAM surface stress results</p>
	<p><i>Retrieves envelope stresses. Envelope is identified by MinMaxType, Component and EnvelopeUID properties. XLAM stress contains the result of the load case or combination in which Component is maximal or minimal. Returns Surfaceld or an error code (EGeneralError or see EStressesError).</i></p>

long **GetCriticalXLAMSurfaceStresses** ([in] long **Surfaceld**, [in] [EMinMaxType](#) **MinMaxType**,
[in] [ECombinationType](#) **CombinationType**, [in] [EAnalysisType](#) **AnalysisType**,
[in] [EXLAMSurfaceStress](#) **Component**,[i/o] [RXLAMSurfaceStresses](#) **Stresses**)

Surfaceld surface index ($0 < \text{Surfaceld} \leq \text{AxisVMSurfaces.Count}$)
CombinationType combination type
AnalysisType analysis type
Component surface stress component
Stresses XLAM surface stress results

Retrieves critical XLAM stresses at one point of a surface element. Critical combination is identified by Component and MinMaxType properties (e.g. `xssSxx_m_T`). Stress contains the result of the critical combination in which Component is maximal or minimal. Returns Surfaceld or an error code ([EGeneralError](#) or see [EStressesError](#)).

long **GetXLAMSurfaceStressesForResultBlocks** ([in] long **Surfaceld**,
[in] [EAnalysisType](#) **AnalysisType**, [out] SAFEARRAY([RResultBlockInfo](#)) **ResultBlockInfo**,
[out] SAFEARRAY([RXLAMSurfaceStresses](#)) **Stresses**)

Surfaceld surface index ($0 < \text{Surfaceld} \leq \text{AxisVMSurfaces.Count}$)
AnalysisType analysis type
ResultBlockInfo array of description records for result blocks

Stresses XLAM surface stress results

Retrieves XLAM stresses of a surface element in all result blocks consecutively. The number of result blocks depends on the AnalysisType property. Returns the common length of ResultBlockInfo and Stresses arrays or an error code ([EGeneralError](#) or see [EStressesError](#)).

XLAM panel efficiencies

long **GetXLAMSurfaceEfficiencyByLoadCaseld** ([in] long **Surfaceld**,
[in] [ESurfaceVertexType](#) **SurfaceVertexType**, [in] long **SurfaceVertexId**,
[in] long **LoadCaseld**, [in] long **LoadLevelOrTimeStep**, [in] [EAnalysisType](#) **AnalysisType**,
[i/o] [RXLAMSurfaceEfficiencyValues](#) **Efficiency**, [out] BSTR **Combination**)

Surfaceld surface index ($0 < \text{Surfaceld} \leq \text{AxisVMSurfaces.Count}$)
SurfaceVertexType vertex type
SurfaceVertexId vertex identifier, see [here](#)
LoadCaseld load case index
LoadLevelOrTimeStep for load level ($0 < \text{LoadLevelOrTimeStep} \leq \text{LoadLevelCount}$)
for timestep ($0 < \text{LoadLevelOrTimeStep} \leq \text{TimestepCount}$)
AnalysisType analysis type
Efficiency XLAM surface efficiency
Combination name of the load case

Retrieves XLAM efficiencies at a point of a surface. Returns Surfaceld or an error code ([EGeneralError](#) or see [EStressesError](#)).

long **GetXLAMSurfaceEfficiencyByLoadCombinationId** ([in] long **Surfaceld**,
[in] [ESurfaceVertexType](#) **SurfaceVertexType**, [in] long **SurfaceVertexId**,
[in] long **LoadCombinationId**, [in] long **LoadLevelOrTimeStep**, [in] [EAnalysisType](#) **AnalysisType**,
[i/o] [RXLAMSurfaceEfficiencyValues](#) **Efficiency**, [out] BSTR **Combination**)

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

SurfaceVertexType vertex type

SurfaceVertexId vertex identifier, see [here](#)

LoadCombinationId load combination index

LoadLevelOrTimeStep for load level ($0 < \text{LoadLevelOrTimeStep} \leq \text{LoadLevelCount}$)

for timestep ($0 < \text{LoadLevelOrTimeStep} \leq \text{TimestepCount}$)

AnalysisType analysis type

Efficiency XLAM surface efficiency

Combination name of the load case

Retrieves XLAM efficiencies at a point of a surface. Returns Surfaceld or an error code ([EGeneralError](#) or see [EStressesError](#)).

long **GetEnvelopeXLAMSurfaceEfficiency** ([in] long **Surfaceld**, [in] [ESurfaceVertexType](#) **SurfaceVertexType**, [in] long **SurfaceVertexId**, [in] [EMinMaxType](#) **MinMaxType**,
[in] [EAnalysisType](#) **AnalysisType**, [in] [EXLAMSurfaceEfficiency](#) **Component**,
[i/o] [RXLAMSurfaceEfficiencyValues](#) **Efficiency**, [out] long **LoadCaseOrCombinationId**,
[out] long **LoadLevel**)

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

SurfaceVertexType vertex type

SurfaceVertexId vertex identifier, see [here](#)

MinMaxType Minimum or maximum value

AnalysisType analysis type

Component surface efficiency component

Efficiency XLAM surface efficiency

LoadCaseOrCombinationId load case or load combination index of surface midpoint, if index is > [IAxisVMLoadcases](#).count then Load combination index = **LoadCaseOrCombinationId** - [IAxisVMLoadcases](#).count

LoadLevel load level

Retrieves envelope XLAM efficiencies at one point of a surface element. Envelope is identified by **MinMaxType**, **Component** and **EnvelopeUID** properties. XLAM efficiencies contains the result of the load case or combination in which Component is maximal or minimal. Returns Surfaceld or an error code ([EGeneralError](#) or see [EStressesError](#)).

long	GetCriticalXLAMSurfaceEfficiency ([in] long Surfaceld , [in] ESurfaceVertexType SurfaceVertexType , [in] long SurfaceVertexId , [in] EMinMaxType MinMaxType , [in] ECombinationType CombinationType , [in] EAnalysisType AnalysisType , [in] EXLAMSurfaceEfficiency Component , [i/o] RXLAMSurfaceEfficiencyValues Efficiency , [out] ECombinationType CriticalCombinationType , [out] SAFEARRAY(double) Factors , [out] SAFEARRAY(long) LoadCaselds)
	Surfaceld <i>surface index ($0 < \text{Surfaceld} \leq \text{AxisVMSurfaces.Count}$)</i>
	SurfaceVertexType <i>vertex type</i>
	SurfaceVertexId <i>vertex identifier, see here</i>
	MinMaxType <i>Minimum or maximum value</i>
	CombinationType <i>combination type</i>
	AnalysisType <i>analysis type</i>
	Component <i>surface efficiency component</i>
	Efficiency <i>XLAM surface efficiency</i>
	CriticalCombinationType <i>combination type corresponding to the critical load combination</i>
	Factors <i>load factors of the critical load combination</i>
	LoadCaselds <i>load case indexes of the critical load combination</i>
	<i>Retrieves critical XLAM efficiencies at one point of a surface element. Critical combination is identified by Component and efficiency properties (e.g. xse_M_N). Returns Surfaceld or an error code (EGeneralError or see EStressesError).</i>
long	GetCriticalXLAMSurfaceStress ([in] long Surfaceld , [in] ESurfaceVertexType SurfaceVertexType , [in] long SurfaceVertexId , [in] EMinMaxType MinMaxType , [in] ECombinationType CombinationType , [in] EAnalysisType AnalysisType , [in] EXLAMSurfaceEfficiency Component , [i/o] RXLAMSurfaceEfficiencyValues Efficiency , [out] ECombinationType CriticalCombinationType , [out] SAFEARRAY(double) Factors , [out] SAFEARRAY(long) LoadCaselds)
	Surfaceld <i>surface index ($0 < \text{Surfaceld} \leq \text{AxisVMSurfaces.Count}$)</i>
	SurfaceVertexType <i>vertex type</i>
	SurfaceVertexId <i>vertex identifier, see here</i>
	MinMaxType <i>Minimum or maximum value</i>
	CombinationType <i>combination type</i>
	AnalysisType <i>analysis type</i>
	Component <i>surface efficiency component</i>
	Efficiency <i>XLAM surface efficiency</i>
	CriticalCombinationType <i>combination type corresponding to the critical load combination</i>
	Factors <i>load factors of the critical load combination</i>
	LoadCaselds <i>load case indexes of the critical load combination</i>
	<i>Retrieves critical XLAM stresses at one point of a surface element. Critical combination is identified by Component and efficiency properties (e.g. xse_M_N). Stress contains the result of the critical combination in which Component is maximal or minimal. Returns Surfaceld or an error code (EGeneralError or see EStressesError).</i>

long	GetXLAMSurfaceEfficienciesByLoadCaseId ([in] long Surfaceld , [in] long LoadCaseId , [in] long LoadLevelOrTimeStep , [in] EAnalysisType AnalysisType , [i/o] RXLAMSurfaceEfficiencies Efficiencies , [out] BSTR Combination)
	<p style="margin-left: 20px;">Surfaceld surface index ($0 < \text{Surfaceld} \leq \text{AxisVMSurfaces.Count}$)</p> <p style="margin-left: 20px;">LoadCaseId load case index</p> <p style="margin-left: 20px;">LoadLevelOrTimeStep for load level ($0 < \text{LoadLevelOrTimeStep} \leq \text{LoadLevelCount}$) for timestep ($0 < \text{LoadLevelOrTimeStep} \leq \text{TimestepCount}$)</p> <p style="margin-left: 20px;">AnalysisType analysis type</p> <p style="margin-left: 20px;">Efficiencies XLAM surface efficiency results</p> <p style="margin-left: 20px;">Combination name of the load case</p>
	<p>Retrieves efficiencies of an XLAM surface. Returns Surfaceld or an error code (EGeneralError or see EStressesError).</p>
long	GetXLAMSurfaceEfficienciesByLoadCombinationId ([in] long Surfaceld , [in] long LoadCombinationId , [in] long LoadLevelOrTimeStep , [in] EAnalysisType AnalysisType , [i/o] RXLAMSurfaceEfficiencies Efficiencies)
	<p style="margin-left: 20px;">Surfaceld surface index ($0 < \text{Surfaceld} \leq \text{AxisVMSurfaces.Count}$)</p> <p style="margin-left: 20px;">LoadCombinationId load combination index</p> <p style="margin-left: 20px;">LoadLevelOrTimeStep for load level ($0 < \text{LoadLevelOrTimeStep} \leq \text{LoadLevelCount}$) for timestep ($0 < \text{LoadLevelOrTimeStep} \leq \text{TimestepCount}$)</p> <p style="margin-left: 20px;">AnalysisType analysis type</p> <p style="margin-left: 20px;">Efficiencies XLAM surface efficiency results</p>
	<p>Retrieves efficiencies of an XLAM surface. Returns Surfaceld or an error code (EGeneralError or see EStressesError).</p>
long	GetXLAMSurfaceStressesByLoadCombinationId ([in] long Surfaceld , [in] long LoadCombinationId , [in] long LoadLevel , [in] EAnalysisType AnalysisType , [i/o] RXLAMSurfaceEfficiencies Efficiencies , [out] BSTR Combination)
	<p style="margin-left: 20px;">Surfaceld surface index ($0 < \text{Surfaceld} \leq \text{AxisVMSurfaces.Count}$)</p> <p style="margin-left: 20px;">LoadCombinationId load combination index</p> <p style="margin-left: 20px;">LoadLevel for load level ($0 < \text{LoadLevelOrTimeStep} \leq \text{LoadLevelCount}$)</p> <p style="margin-left: 20px;">AnalysisType analysis type</p> <p style="margin-left: 20px;">Efficiencies XLAM surface efficiency results</p> <p style="margin-left: 20px;">Combination name of the load case</p>
	<p>Retrieves efficiencies of an XLAM surface. Returns Surfaceld or an error code (EGeneralError or see EStressesError).</p>
long	GetEnvelopeXLAMSurfaceEfficiencies ([in] long Surfaceld , [in] EMinMaxType MinMaxType , [in] EAnalysisType AnalysisType , [in] EXLAMSurfaceEfficiency Component , [i/o] RXLAMSurfaceStresses Efficiencies)
	<p style="margin-left: 20px;">Surfaceld surface index ($0 < \text{Surfaceld} \leq \text{AxisVMSurfaces.Count}$)</p> <p style="margin-left: 20px;">MinMaxType Minimum or maximum value</p> <p style="margin-left: 20px;">AnalysisType analysis type</p> <p style="margin-left: 20px;">Component surface efficiency component</p> <p style="margin-left: 20px;">Efficiencies XLAM surface efficiency results</p>
	<p>Retrieves envelope stresses. Envelope is identified by MinMaxType, Component and EnvelopeUID properties. XLAM stress contains the result of the load case or combination in which Component is maximal or minimal. Returns Surfaceld or an error code (EGeneralError or see EStressesError).</p>

long **GetCriticalXLAMSurfaceEfficiencies** ([in] long **Surfacing**, [in] [EMinMaxType](#) **MinMaxType**,
[in] [ECombinationType](#) **CombinationType**, [in] [EAnalysisType](#) **AnalysisType**,
[in] [EXLAMSurfaceEfficiency](#) **Component**, [i/o] [RXLAMSurfaceStresses](#) **Efficiencies**)

Surfacing surface index ($0 < \text{Surfacing} \leq \text{AxisVMSurfaces.Count}$)
MinMaxType Minimum or maximum value
CombinationType combination type
AnalysisType analysis type
Component surface efficiency component
Efficiencies XLAM surface efficiency

Retrieves critical XLAM efficiencies at one point of a surface element. Returns Surfacing or an error code ([EGeneralError](#) or see [EStressesError](#)).

Save results to MetaFile functions

long **SaveCriticalMemberStressesToMetaFile** ([in] BSTR **FileName**, [in] long **MemberId**,
[in] [ECombinationType](#) **CombinationType**, [in] [EAnalysisType](#) **AnalysisType**, [in] long **Width**,
[in] long **Height**, [in] [EBoolean](#) **EnvelopeOnly**, [in] double **Position**,
[in] [EWindowColourMode](#) **ColourMode**)

FileName name of the file with extension emf
MemberId member index
CombinationType combination type
AnalysisType analysis type
Width picture's size in pixel (minimal acceptable value is 640)
Height picture's size in pixel (minimal acceptable value is 580)
EnvelopeOnly only Envelope
Position position of the investigated cross section along the member
ColourMode colour mode

It creates a metafile from the member's critical stresses. It returns MemberId or an error code ([EGeneralError](#) or see [EStressesError](#)).

long **SaveEnvelopeMemberStressesToMetaFile** ([in] BSTR **FileName**, [in] long **MemberId**, [in] long **EnvelopeUID**,
[in] [EAnalysisType](#) **AnalysisType**, [in] long **Width**, [in] long **Height**, [in]
[EBoolean](#) **EnvelopeOnly**, [in] double **Position**, [in] [EWindowColourMode](#) **ColourMode**)

FileName name of the file with extension emf
MemberId member index
EnvelopeUID unique envelope index
AnalysisType analysis type
Width picture's size in pixel (minimal acceptable value is 640)
Height picture's size in pixel (minimal acceptable value is 580)
EnvelopeOnly only Envelope
Position position of the investigated cross section along the member
ColourMode colour mode

It creates a metafile from the member's stress envelope. It returns MemberId or an error code ([EGeneralError](#) or see [EStressesError](#)).

long **SaveMemberStressesToMetaFileByLoadCaseId** ([in] BSTR **FileName**, [in] long **MemberId**, [in] long **StressPointId**, [in] long **LoadCaseId**, [in] long **LoadLevelOrTimeStep**, [in] [EAnalysisType](#) **AnalysisType**, [in] long **Width**, [in] long **Height**, [in] double **Position**, [in] [EWindowColourMode](#) **ColourMode**)

FileName	<i>name of the file with extension emf</i>
MemberId	<i>member index</i>
StressPointId	<i>stress point index</i>
LoadCaseId	<i>load case index</i>
LoadLevelOrTimeStep	<i>for load level ($0 < LoadLevelOrTimeStep \leq LoadLevelCount$) for timestep ($0 < LoadLevelOrTimeStep \leq TimeStepCount$)</i>
AnalysisType	<i>analysis type</i>
Width	<i>picture's size in pixel (minimal acceptable value is 640)</i>
Height	<i>picture's size in pixel (minimal acceptable value is 580)</i>
Position	<i>position of the investigated cross section along the member</i>
ColourMode	<i>colour mode</i>

It saves the member's stresses by LoadCaseId into a metafile. It returns MemberId or an error code ([EGeneralError](#) or see [EStressesError](#)).

long **SaveMemberStressesToMetaFileByLoadCombinationId** ([in] BSTR **FileName**, [in] long **MemberId**, [in] long **StressPointId**, [in] long **LoadCombinationId**, [in] long **LoadLevelOrTimeStep**, [in] [EAnalysisType](#) **AnalysisType**, [in] long **Width**, [in] long **Height**, [in] double **Position**, [in] [EWindowColourMode](#) **ColourMode**)

FileName	<i>name of the file with extension emf</i>
MemberId	<i>member index</i>
StressPointId	<i>stress point index</i>
LoadCombinationId	<i>load combination index</i>
LoadLevelOrTimeStep	<i>for load level ($0 < LoadLevelOrTimeStep \leq LoadLevelCount$) for timestep ($0 < LoadLevelOrTimeStep \leq TimeStepCount$)</i>
AnalysisType	<i>analysis type</i>
Width	<i>picture's size in pixel (minimal acceptable value is 640)</i>
Height	<i>picture's size in pixel (minimal acceptable value is 580)</i>
Position	<i>position of the investigated cross section along the member</i>
ColourMode	<i>colour mode</i>

It saves the member's stresses by LoadCombinationId into a metafile. It returns MemberId or an error code ([EGeneralError](#) or see [EStressesError](#)).

long **SetUserCreep** ([in] [ELongBoolean](#) **Creep**)

Creep If *IbTrue* concrete creep will be considered in results of nonlinear analysis, if national design code allows it. More [here...](#)

Enable or disable consideration of concrete creep in nonlinear analysis results. If successful returns 1, otherwise an error code ([errDatabaseNotReady](#), [errCreepNotSupported](#)).

Properties

EAnalysisType	AnalysisType • Get or set the type of analysis (linear/nonlinear/vibration/buckling)
ECombinationType	CombinationType • Get or set the type of combination
long	EnvelopeUID • Get or set the unique index of the envelope used in functions for reading envelope results
long	LoadCaseId • Get or set the load case index ($0 < LoadCaseId \leq AxisVMMModel.LoadCases.Count$)
long	LoadCombinationId • Get or set the load combination index ($0 < LoadCombinationId \leq AxisVMMModel.LoadCombinations.Count$)
long	LoadLevelOrTimeStep • Get or set the load level or time step, according to the type of analysis. See LoadLevelOrModeShapeOrTimeStep parameter.
ELineStress	LineStressComponent • Get or set line stress component (for envelope or critical values)

[ESurfaceStress](#) **SurfaceStressComponent** • Get or set surface stress component (for envelope or critical values)

[EMinMaxType](#) **MinMaxType** • Get or set whether results containing minimum or maximum values of the component (LineStressComponent, SurfaceStressComponent) should be read (for envelope or critical values)

[XLAMSurfaceStress](#) **XLAMSurfaceStressComponent** • Get or set XLAM panel surface stress component (for envelope or critical values)

[ELongBoolean](#) **UserCreep**

Returns *IbTrue* if nonlinear analysis results consider concrete creep. More [here...](#)

IAxisVMSteelDesignResults

Interface for reading steel design results

Error codes

```
enum ESteelDesignResultsError = {
    sdreCOMError = -100001
    sdreLoadCaseIdIndexOutOfBounds = -100002
    sdreLoadCombinationIdIndexOutOfBounds = -100003
    sdreInvalidAnalysisType = -100004
    sdreCombinationTypeNotValidForCurrentNationalDesignCode = -100005}
```

COM server error
Invalid LoadCaseID while reading results
Invalid CombinationID while reading results
Invalid type of results for used analysis
The CombinationType cannot be used for the selected national code. Some design codes allow only certain ECombinationTypes (see [here](#)) or results not available for CombinationType

Records / structures

RSteelDesignResult = (

double PosX Position of the checked section [m]
double DesignValue Design value for the check (unit depends on type of check)
double LimitValue Limit value for the check (unit depends on type of check)
)

Steel design result for each check on a section according to the design code.

Type of steel design results in array

(for national code SIA 263:2003)

Array item No.	design value	limit value
1.	Interaction of normal force, bending moments and shear forces (N-M-V) [-] SIA 263:2013: 5.1.7, 5.2.7, 5.3.7 In an elastic calculation, if the shear stress is higher than 50% of the shear resistance, the Von Misses stress results are calculated (SIA 263: 4.3.5.4). See AxisVM manual for more details..	1,0
2.	Interaction of normal force, bending moments and stability (N-M-Stab) [-] SIA 263:2013: 5.1.9-10, 5.2.6, 5.3.5	1,0
3.	Capacity ratio of shear, y-y axis [-] SIA 263:2013: 5.1.4, 5.2.4, 5.3.4	1,0
4.	Capacity ratio of shear, z-z axis [-] SIA 263:2013: 5.1.4, 5.2.4, 5.3.4	1,0
5.	NcEd - design normal force [kN]	NplRd - design plastic resistance to normal forces of the gross cross-section for class 1,2 or 3 cross-sections [kN]
6.	NcEd - design normal force [kN]	NeffRd - design effective resistance to normal forces for class 4 cross-sections [kN]
7.	VyEd - design shear force, y-y axis [kN]	VplyRd - design resistance to shear, y-y axis [kN]
8.	VzEd - design shear force, z-z axis [kN]	VzRd - design resistance to shear, z-z axis [kN] including the effect of shear buckling, contribution from the web [kN] (calculated only for sections: I, IN, C and box) SIA 263:2013: 4.5.4
9.	MyEd - design bending moment, y-y axis [kNm]	MyRd - design resistance to bending moment, y-y axis [kNm]

10.	MzEd - design bending moment, z-z axis [kNm]	MzRd - design resistance to bending moment, z-z axis [kNm]
11.	MyEd - design bending moment, y-y axis [kNm]	MzRd - design resistance to bending moment, z-z axis [kNm]
12.	MzEd - design bending moment, z-z axis [kNm]	MeffyRd - design effective resistance to bending moment for class 4 cross-sections, y-y axis [kNm]
13.	NcEd - design normal force [kN]	NKRd - design flexural or torsional or torsional-flexural buckling resistance of a compression member [kN]
14.	0	LambdaMax - maximal relative slenderness for flexural or torsional or torsional-flexural buckling [-]
15.	0	KhiN - reduction factor for flexural buckling [-]
16.	0	class index for flexural buckling curve
17.	MyEd - design bending moment, y-y axis [kNm]	MDRd - design buckling resistance moment [kNm]
18.	MyEd - design bending moment, y-y axis [kNm]	Mcr- elastic critical moment for the calculation of lateral torsional buckling [kNm]
19.	0	KhiD - reduction factor for lateral torsional buckling [-]
20.	0	class index for lateral torsional buckling curve
21.	0	section class
22.	0	ky - Buckling factor about the local y axis[-]
23.	0	kz - Buckling factor about the local z axis[-]
24.	deprecated	
25.	0	Aeff - effective area of a cross-section when subjected only to uniform compression [m ²]
26.	0	eNy - shift of the centroid of the effective area (Aeff) relative to the center of gravity of the gross cross-section, when subjected only to uniform compression y-y axis [m]
27.	0	Wy,eff(+),min - effective section modulus (corresponding to the fibre with the maximum elastic stress) of the cross-section when subjected only to positive bending moment, y-y axis [m ³]
28.	0	Wy,eff(-),min - effective section modulus (corresponding to the fibre with the maximum elastic stress) of the cross-section when subjected only to negative bending moment, y-y axis [m ³]
29.	0	Wz,eff(+),min - effective section modulus (corresponding to the fibre with the maximum elastic stress) of the cross-section when subjected only to positive bending moment, z-z axis [m ³]
30.	0	Wz,eff(-),min - effective section modulus (corresponding to the fibre with the maximum elastic stress) of the cross-section when subjected only to negative bending moment, z-z axis [m ³]
31.	0	C1 - equivalent uniform moment factor for the calculation of the critical moment based on Lopez formula [-]
32-36.	deprecated	

37.	Utilization in ultimate limit state (ULS) [-]	1,0
38.	Utilization in serviceability limit state (SLS) [-]	1,0

(for national code Eurocode 3)

Array item No.	design value	limit value
1.	Interaction of normal force, bending moments and shear forces (N-M-V) [-] EN 1993-1-1:2005: 6.2.1, 6.2.8, 6.2.9.3	1,0
2.	Interaction of normal force, bending moments and buckling (N-M-Buckl) [-] EN 1993-1-1:2005: 6.3.3	1,0
3.	Interaction of normal force, bending and lateral torsional buckling (N-M-LTBuckl) [-] EN 1993-1-1:2005: 6.3.3	1,0
4.	Capacity ratio of shear, y-y axis [-] EN 1993-1-1:2005: 6.2.6 (1) (6.17)	1,0
5.	Capacity ratio of shear including shear web buckling, z-z axis [-] EN 1993-1-1:2005: 6.2.6 (1) (6.17); EN 1993-1-5:2006: 5.1-5.3	1,0
6.	Interaction of normal force, bending and shear web buckling (Vw-N-M) [-] EN 1993-1-1:2005: 6.2.9; EN 1993-1-5:2006: 7.1	1,0
7.	NcEd - design normal force [kN]	NplRd - design plastic resistance to normal forces of the gross cross-section for class 1,2 or 3 cross-sections [kN] - EN 1993-1-1:2005: 6.2.4 (2) (6.10)
8.	NcEd - design normal force [kN]	NeffRd - design effective resistance to normal forces for class 4 cross-sections [kN] - EN 1993-1-1:2005: 6.2.4 (2) (6.11)
9.	VyEd - design shear force, y-y axis [kN]	VplyRd - design resistance to shear, y-y axis [kN] - EN 1993-1-1:2005: 6.2.6 (2) (6.18)
10.	VzEd - design shear force, z-z axis [kN]	VplzRd - design resistance to shear, z-z axis [kN] - EN 1993-1-1:2005: 6.2.6 (2) (6.18)
11.	VzEd - design shear force, z-z axis [kN]	VbwRd - design resistance to shear buckling, contribution from the web [kN] (calculated only for sections: I, IN, and box; otherwise zero) - EN 1993-1-5:2006: 5.2 (1) (5.2)
12.	MyEd - design bending moment, y-y axis [kNm]	MelyRd - design elastic resistance to bending moment, y-y axis [kNm] - EN 1993-1-1:2005: 6.2.5 (2) (6.14)
13.	MzEd - design bending moment, z-z axis [kNm]	MelzRd - design elastic resistance to bending moment, z-z axis [kNm] - EN 1993-1-1:2005: 6.2.5 (2) (6.14)
14.	MyEd - design bending moment, y-y axis [kNm]	MplyRd - design plastic resistance to bending moment, y-y axis [kNm] - EN 1993-1-1:2005: 6.2.5 (2) (6.13)
15.	MzEd - design bending moment, z-z axis [kNm]	MplzRd - design plastic resistance to bending moment, z-z axis [kNm] - EN 1993-1-1:2005: 6.2.5 (2) (6.13)
16.	MyEd - design bending moment, y-y axis [kNm]	MeffyRd - design effective resistance to bending moment for class 4 cross-sections, y-y axis [kNm] - EN 1993-1-1:2005: 6.2.5 (2) (6.15)

17.	MzEd - design bending moment, z-z axis [kNm]	MeffzRd - design effective resistance to bending moment for class 4 cross-sections, z-z axis [kNm] - EN 1993-1-1:2005: 6.2.5 (2) (6.15)
18.	NcEd - design normal force [kN]	NbRd - design flexural or torsional or torsional-flexural buckling resistance of a compression member [kN] - EN 1993-1-1:2005: 6.3.1.1 (3)
19.	MyEd - design bending moment, y-y axis [kNm]	MbRd - design buckling resistance moment [kNm] - EN 1993-1-1:2005: 6.3.2.1 (3) (6.55)
20.	0	section class [-]
21.	MyEd - design bending moment, y-y axis [kNm]	Mcr - elastic critical moment for the calculation of lateral torsional buckling [kNm]
22.	0	KhiN - reduction factor for flexural buckling [-]
23.	0	KhiLT - reduction factor for lateral torsional buckling [-]
24.	0	class index for flexural buckling curve [-]
25.	0	class index for lateral torsional buckling curve [-]
26.	0	ky - Buckling factor about the local y axis [-]
27.	0	kz - Buckling factor about the local z axis [-]
28.	0	LambdaMax - maximal relative slenderness for flexural or torsional or torsional-flexural buckling [-]
29.	0	Aeff - effective area of a cross-section when subjected only to uniform compression [m ²]
30.	0	eNy - shift of the centroid of the effective area (Aeff) relative to the center of gravity of the gross cross-section, when subjected only to uniform compression y-y axis [m]
31.	0	Wy,eff(+),min - effective section modulus (corresponding to the fibre with the maximum elastic stress) of the cross-section when subjected only to positive bending moment, y-y axis [m ³]
32.	0	Wy,eff(-),min - effective section modulus (corresponding to the fibre with the maximum elastic stress) of the cross-section when subjected only to negative bending moment, y-y axis [m ³]
33.	0	Wz,eff(+),min - effective section modulus (corresponding to the fibre with the maximum elastic stress) of the cross-section when subjected only to positive bending moment, z-z axis [m ³]
34.	0	Wz,eff(-),min - effective section modulus (corresponding to the fibre with the maximum elastic stress) of the cross-section when subjected only to negative bending moment, z-z axis [m ³]
35.	0	C1 - equivalent uniform moment factor for the calculation of the critical moment based on Lopez formula [-]
36-43.	deprecated	
44.	Utilization in ultimate limit state (ULS) [-]	1,0
45.	Utilization in serviceability limit state (SLS) [-]	1,0

Functions

long	GetEfficiencyAndCombination ([in] long SteelDesignMemberId , [in] EResultType ResultType , [out] double Efficiency , [out] BSTR Combination)
	SteelDesignMemberId index of steel design member ResultType type of result Efficiency maximum efficiency Combination name of load case or combination corresponding to max. efficiency
	Get efficiency and combination depending on result type and set interface properties.
	Returns SteelDesignMemberId if successful, otherwise returns an error code (errDatabaseNotReady , errIndexOutOfBounds , sdreLoadCaseIdIndexOutOfBounds , sdreInvalidAnalysisType , sdreLoadCombinationIdIndexOutOfBounds).
long	GetEfficiencyAndCombinationByLoadCaseld ([in] long SteelDesignMemberId , [in] long LoadCaseld , [in] long LoadLevel , [out] double Efficiency , [out] BSTR Combination)
	SteelDesignMemberId index of steel design member LoadCaseld load case index ($0 < \text{LoadCaseld} \leq \text{AxisVMLoadCases.Count}$) LoadLevel load level (increment) index Efficiency maximum efficiency Combination name of load case or combination corresponding to max. efficiency
	Get efficiency and combination depending on load case. Returns SteelDesignMemberId if successful, otherwise returns an error code (errDatabaseNotReady , errIndexOutOfBounds , sdreInvalidAnalysisType , sdreLoadCaseIdIndexOutOfBounds).
long	GetEfficiencyAndCombinationByLoadCombinationId ([in] long SteelDesignMemberId , [in] long LoadCombinationId , [in] long LoadLevel , [out] double Efficiency , [out] BSTR Combination)
	SteelDesignMemberId index of steel design member LoadCombinationId load combination index ($0 < \text{LoadCaseld} \leq \text{AxisVMLoadCombinations.Count}$) LoadLevel load level (increment) index Efficiency maximum efficiency Combination name of load case or combination corresponding to max. efficiency
	Get efficiency and combination depending on load combination. Returns SteelDesignMemberId if successful, otherwise returns an error code (errDatabaseNotReady , errIndexOutOfBounds , sdreInvalidAnalysisType , sdreLoadCombinationIdIndexOutOfBounds).
long	GetEnvelopeEfficiencyAndCombination ([in] long SteelDesignMemberId , [in] long EnvelopeUID , [out] double Efficiency , [out] BSTR Combination)
	SteelDesignMemberId index of steel design member EnvelopeUID unique index of the envelope Efficiency maximum efficiency Combination name of load case or combination corresponding to max. efficiency
	Get efficiency and combination depending on envelope. Returns SteelDesignMemberId if successful, otherwise returns an error code (errDatabaseNotReady , errIndexOutOfBounds , sdreInvalidAnalysisType).

long	GetCriticalEfficiencyAndCombination ([in] long SteelDesignMemberId , [in] ECombinationType CombinationType , [out] double Efficiency , [out] BSTR Combination)
	<p>SteelDesignMemberId <i>index of steel design member</i></p> <p>CombinationType <i>combination type</i></p> <p>Efficiency <i>maximum efficiency</i></p> <p>Combination <i>name of load case or combination corresponding to max. efficiency</i></p> <p>Get efficiency and combination depending on critical combination type. Returns SteelDesignMemberId if successful, otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBounds, sdreInvalidAnalysisType, errCriticalCombinationNotAllowed, sdreCombinationTypeNotValidForCurrentNationalDesignCode).</p>
long	GetSteelDesignResultsByLoadCaseld ([in] long SteelDesignMemberId , [in] long LoadCaseld , [in] long LoadLevel , [in] EAnalysisType AnalysisType , [out] long * ResultsPerSection , [out] SAFEARRAY(RSteelDesignResult)* SteelDesignResults , [out] BSTR* Combination)
	<p>SteelDesignMemberId <i>index of steel design member</i></p> <p>LoadCaseld <i>load case index ($0 < LoadCaseld \leq \text{AxisVMLoadCases}.\text{Count}$)</i></p> <p>LoadLevel <i>load level (increment) index</i></p> <p>AnalysisType <i>Type of Analysis</i></p> <p>ResultsPerSection <i>Number of results per section</i></p> <p>SteelDesignResults <i>The form of the one dimensional array with ResultsPerSection*n items: [1..ResultsPerSection],[ResultsPerSection+1..2*ResultsPerSection],...,[$(n-1)*ResultsPerSection..n*ResultsPerSection$], where between [] are the results of one section. See here</i></p> <p><i>PosX is relative distance from start node of the line</i></p> <p>Combination <i>Name of load case</i></p> <p>Returns ResultsPerSection*n where n is number of sections, otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBounds, sdreLoadCaseldIndexOutOfBounds, sdreInvalidAnalysisType, sdreCOMError).</p>
long	GetSteelDesignResultsByLoadCaseld_Abs ([in] long SteelDesignMemberId , [in] long LoadCaseld , [in] long LoadLevel , [in] EAnalysisType AnalysisType , [out] long * ResultsPerSection , [out] SAFEARRAY(RSteelDesignResult)* SteelDesignResults , [out] BSTR* Combination)
	<i>Same as GetSteelDesignResultsByLoadCaseld but PosX is absolute distance from start node of the steel design member</i>
long	GetSteelDesignResultsByLoadCombinationId ([in] long SteelDesignMemberId , [in] long LoadCombinationId , [in] long LoadLevel , [in] EAnalysisType AnalysisType , [out] long * ResultsPerSection , [out] SAFEARRAY(RSteelDesignResult)* SteelDesignResults , [out] BSTR* Combination)
	<p>SteelDesignMemberId <i>index of steel design member</i></p> <p>LoadCombinationId <i>load combination index ($0 < LoadCombinationId \leq \text{AxisVMLoadCombinations}.\text{Count}$)</i></p> <p>LoadLevel <i>load level (increment) index</i></p> <p>AnalysisType <i>Type of Analysis</i></p> <p>ResultsPerSection <i>Number of results per section</i></p> <p>SteelDesignResults <i>The form of the one dimensional array with ResultsPerSection*n items: [1..ResultsPerSection],[ResultsPerSection+1..2*ResultsPerSection],...,[$(n-1)*ResultsPerSection..n*ResultsPerSection$], where between [] are the results of one section. See here</i></p> <p><i>PosX is relative distance from start node of the line</i></p> <p>Combination <i>Name of combination</i></p> <p>Returns ResultsPerSection*n where n is number of sections, otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBounds, sdreLoadCombinationIdIndexOutOfBounds, sdreInvalidAnalysisType, sdreCOMError).</p>

long	GetSteelDesignResultsByLoadCombinationId_Abs ([in] long SteelDesignMemberId , [in] long LoadCombinationId , [in] long LoadLevel , [in] EAnalysisType AnalysisType , [out] long * ResultsPerSection , [out] SAFEARRAY(RSteelDesignResult)* SteelDesignResults , [out] BSTR* Combination)
	<i>Same as GetSteelDesignResultsByLoadCombinationId, but PosX is absolute distance from start node of the steel design member</i>
long	GetEnvelopeSteelDesignResults ([in] long SteelDesignMemberId , [in] EAnalysisType AnalysisType , [out] long* ResultsPerSection , [out] SAFEARRAY(RSteelDesignResult)* SteelDesignResults , [out] BSTR* Combination)
	<p>SteelDesignMemberId index of steel design member</p> <p>AnalysisType Type of Analysis</p> <p>ResultsPerSection Number of results per section</p> <p>SteelDesignResults The form of the one dimensional array with ResultsPerSection*n items: [1..ResultsPerSection],[ResultsPerSection+1..2*ResultsPerSection],...,[(n-1)*ResultsPerSection..n*ResultsPerSection], where between [] are the results of one section. See here</p> <p>PosX is relative distance from start node of the line</p> <p>Combination Name of critical combination</p> <p><i>Envelope is identified by EnvelopeUID property. Returns ResultsPerSection*n where n is number of sections, otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBounds, sdreInvalidAnalysisType, sdreCOMError).</i></p>
long	GetEnvelopeSteelDesignResults_Abs ([in] long SteelDesignMemberId , [in] EAnalysisType AnalysisType , [out] long* ResultsPerSection , [out] SAFEARRAY(RSteelDesignResult)* SteelDesignResults , [out] BSTR* Combination)
	<i>Same as GetEnvelopeSteelDesignResults , but PosX is absolute distance from start node of the steel design member</i>
long	GetEnvelopeSteelDesignResults2 ([in] long SteelDesignMemberId , [in] long EnvelopeUID , [in] EAnalysisType AnalysisType , [out] long* ResultsPerSection , [out] SAFEARRAY(RSteelDesignResult)* SteelDesignResults , [out] BSTR* Combination)
	<p>SteelDesignMemberId index of steel design member</p> <p>EnvelopeUID unique index of the envelope used in functions for reading envelope results</p> <p>AnalysisType Type of Analysis</p> <p>ResultsPerSection Number of results per section</p> <p>SteelDesignResults The form of the one dimensional array with ResultsPerSection*n items: [1..ResultsPerSection],[ResultsPerSection+1..2*ResultsPerSection],...,[(n-1)*ResultsPerSection..n*ResultsPerSection], where between [] are the results of one section. See here</p> <p>PosX is relative distance from start node of the line</p> <p>Combination Name of critical combination</p> <p><i>Returns ResultsPerSection*n where n is number of sections, otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBounds, sdreInvalidAnalysisType, sdreCOMError).</i></p>

long	GetCriticalSteelDesignResults ([in] long SteelDesignMemberId , [in] ECombinationType CombinationType , [in] EAnalysisType AnalysisType , [out] long* ResultsPerSection , [out] SAFEARRAY(RSteelDesignResult)* SteelDesignResults , [out] BSTR* Combination)
	SteelDesignMemberId <i>index of steel design member</i> CombinationType <i>Combination Type</i> AnalysisType <i>Type of Analysis</i> ResultsPerSection <i>Number of results per section</i> SteelDesignResults <i>The form of the one dimensional array with ResultsPerSection*n items: [1..ResultsPerSection],[ResultsPerSection+1..2*ResultsPerSection],...,[(n-1)*ResultsPerSection..n*ResultsPerSection], where between [] are the results of one section. See here</i> <i>PosX is relative distance from start node of the line</i>
	Combination <i>Name of critical combination</i>
	<i>Returns ResultsPerSection*n where n is number of sections, otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBounds, sdreInvalidAnalysisType, sdreCOMError, sdreCombinationTypeNotValidForCurrentNationalDesignCode).</i>
long	GetCriticalSteelDesignResults_Abs ([in] long SteelDesignMemberId , [in] ECombinationType CombinationType , [in] EAnalysisType AnalysisType , [out] long* ResultsPerSection , [out] SAFEARRAY(RSteelDesignResult)* SteelDesignResults , [out] BSTR* Combination)
	<i>Same as GetCriticalSteelDesignResults, but PosX is absolute distance from start node of the steel design member</i>
long	GetAllSteelDesignResultsByLoadCaseld ([in] long LoadCaseld , [in] long LoadLevel , [in] EAnalysisType AnalysisType , [out] SAFEARRAY(long)* SectionCounts , [out] long * ResultsPerSection , [out] SAFEARRAY(RSteelDesignResult)* SteelDesignResults , [out] SAFEARRAY(BSTR)* Combinations)
	LoadCaseld <i>load case index (0 < LoadCaseld ≤ AxisVMLoadCases.Count)</i> LoadLevel <i>load level (increment) index</i> AnalysisType <i>Type of Analysis</i> SectionCounts <i>Array (long) with count of sections for each steel design member</i> ResultsPerSection <i>Number of results per section</i> SteelDesignResults <i>The form of the one dimensional array with N*ResultsPerSection*n items: - First SteelDesignMember results: [1..ResultsPerSection],[ResultsPerSection+1..2*ResultsPerSection],...,[(n-1)*ResultsPerSection..n*ResultsPerSection], where between [] are the results of one section. - Then continues the sameway with next steel design member Until the last steel design member. See here</i> <i>PosX is relative distance from start node of the line</i>
	Combinations <i>Array of loadcase names</i>
	<i>Returns N*ResultsPerSection*n where N is number of steel design members and n is number of sections. Otherwise returns an error code (errDatabaseNotReady, sdreLoadCaseldIndexOutOfBounds, sdreInvalidAnalysisType, sdreCOMError).</i>

long	GetAllSteelDesignResultsByLoadCombinationId ([in] long LoadCombinationId , [in] long LoadLevel , [in] EAnalysisType AnalysisType , [out] SAFEARRAY(long)* SectionCounts , [out] long* ResultsPerSection , [out] SAFEARRAY(RSteelDesignResult)* SteelDesignResults , [out] SAFEARRAY(BSTR)* Combinations)
	<p>LoadCombinationId <i>load combination index (0 < LoadCombinationId ≤ AxisVMLoadCombinations.Count)</i></p> <p>LoadLevel <i>load level (increment) index</i></p> <p>AnalysisType <i>Type of Analysis</i></p> <p>SectionCounts <i>Array (long) with count of sections for each steel design member</i></p> <p>ResultsPerSection <i>Number of results per section</i></p> <p>SteelDesignResults <i>The form of the one dimensional array with N*ResultsPerSection*n items:</i></p> <ul style="list-style-type: none"> - <i>First SteelDesignMember results: [1..ResultsPerSection],[ResultsPerSection+1..2*ResultsPerSection] ,...,[n-1]*ResultsPerSection..n*ResultsPerSection], where between [] are the results of one section.</i> - <i>Then continues the sameway with next steel design member</i> <p><i>Until the last steel design member. See here</i></p> <p><i>PosX is relative distance from start node of the line</i></p> <p>Combinations <i>Array of strings with combination names</i></p>
	<p><i>Returns N*ResultsPerSection*n where N is number of steel design members and n is number of sections. Otherwise returns an error code (errDatabaseNotReady, sdreLoadCaseIdIndexOutOfBounds, sdreInvalidAnalysisType, sdreCOMError).</i></p>
long	GetAllEnvelopeSteelDesignResults ([in] EAnalysisType AnalysisType , [out] SAFEARRAY(long)* SectionCounts , [out] long* ResultsPerSection , [out] SAFEARRAY(RSteelDesignResult)* SteelDesignResults , [out] SAFEARRAY(BSTR)* Combinations)
	<p>AnalysisType <i>Type of Analysis</i></p> <p>SectionCounts <i>Array (long) with count of sections for each steel design member</i></p> <p>ResultsPerSection <i>Number of results per section</i></p> <p>SteelDesignResults <i>The form of the one dimensional array with N*ResultsPerSection*n items:</i></p> <ul style="list-style-type: none"> - <i>First steel design member results: [1..ResultsPerSection],[ResultsPerSection+1..2*ResultsPerSection] ,...,[n-1]*ResultsPerSection..n*ResultsPerSection], where between [] are the results of one section.</i> - <i>Then continues the sameway with next steel design member</i> <p><i>Until the last steel design member. See here</i></p> <p><i>PosX is relative distance from start node of the line</i></p> <p>Combinations <i>Array of strings with names of critical combinations</i></p>
	<p><i>Envelope is identified by EnvelopeUID property. Returns N*ResultsPerSection*n where N is number of steel design members and n is number of sections. Otherwise returns an error code (errDatabaseNotReady, sdreLoadCaseIdIndexOutOfBounds, sdreInvalidAnalysisType, sdreCOMError).</i></p>

lon	GetAllCriticalSteelDesignResults ([in] ECombinationType CombinationType,
g	[in] EAnalysisType AnalysisType, [out] SAFEARRAY(long) * SectionCounts,
	[out] long * ResultsPerSection, [out] SAFEARRAY(RSteelDesignResult) * SteelDesignResults,
	[out] SAFEARRAY(BSTR)* Combinations)
	CombinationType Combination Type
	AnalysisType Type of Analysis
	SectionCounts Array (long) with count of sections for each steel design member
	ResultsPerSection Number of results per section
	SteelDesignResults The form of the one dimensional array with $N*ResultsPerSection*n$ items:
	- First steel design member results: [1..ResultsPerSection],[ResultsPerSection+1..2*ResultsPerSection],...,[$(n-1)*ResultsPerSection..n*ResultsPerSection$], where between [] are the results of one section.
	- Then continues the sameway with next steel design member Until the last steel design member. See here
	PosX is relative distance from start node of the line
	Combinations Array with names of combinations
	Returns $N*ResultsPerSection*n$ where N is number of steel design members and n is number of sections. Otherwise returns an error code (errDatabaseNotReady , sdreLoadCaseIdIndexOutOfBounds , sdreInvalidAnalysisType , sdreCOMError , sdreCombinationTypeNotValidForCurrentNationalDesignCode).
lon	SteelDesignResultsByLoadCaseId ([in] long SteelDesignMemberId, [out] long *
g	ResultsPerSection, [out] SAFEARRAY(RSteelDesignResult) * SteelDesignResults, [out] BSTR* Combination)
	SteelDesignMember index of steel design member
	Id
	ResultsPerSection Number of results per section
	SteelDesignResults The form of the one dimensional array with $ResultsPerSection*n$ items: [1..ResultsPerSection],[ResultsPerSection+1..2*ResultsPerSection],...,[$(n-1)*ResultsPerSection..n*ResultsPerSection$], where between [] are the results of one section. See here
	PosX is relative distance from start node of the line
	Combination Name of loadcase
	Returns $ResultsPerSection*n$ where n is number of sections, otherwise returns an error code (errDatabaseNotReady , errIndexOutOfBounds , sdreLoadCaseIdIndexOutOfBounds , sdreInvalidAnalysisType , sdreCOMError).
lon	SteelDesignResultsByLoadCombinationId ([in] long SteelDesignMemberId,
g	[out] long* ResultsPerSection, [out] SAFEARRAY(RSteelDesignResult)* SteelDesignResults,
	[out] BSTR* Combination)
	SteelDesignMember index of steel design member
	Id
	ResultsPerSection Number of results per section
	SteelDesignResults The form of the one dimensional array with $ResultsPerSection*n$ items: [1..ResultsPerSection],[ResultsPerSection+1..2*ResultsPerSection],...,[$(n-1)*ResultsPerSection..n*ResultsPerSection$], where between [] are the results of one section. See here
	PosX is relative distance from start node of the line
	Combination Name of combination
	Returns $ResultsPerSection*n$ where n is number of sections, otherwise returns an error code (errDatabaseNotReady , errIndexOutOfBounds , sdreLoadCombinationIdIndexOutOfBounds , sdreInvalidAnalysisType , sdreCOMError).

long	EnvelopeSteelDesignResults ([in] long SteelDesignMemberId [out] long* ResultsPerSection, [out] SAFEARRAY(RSteelDesignResult)* SteelDesignResults, [out] BSTR* Combination)
SteelDesignMemberId	index of steel design member
ResultsPerSection	Number of results per section
SteelDesignResults	The form of the one dimensional array with ResultsPerSection*n items: [1..ResultsPerSection],[ResultsPerSection+1..2*ResultsPerSection],...,[((n-1)*ResultsPerSection..n*ResultsPerSection], where between [] are the results of one section. See here
	PosX is relative distance from start node of the line
Combination	Name of combination
	Envelope is identified by EnvelopeUID property. Returns ResultsPerSection*n where n is number of sections, otherwise returns an error code (errDatabaseNotReady , errIndexOutOfBounds , sdreInvalidAnalysisType , sdreCOMError).
long	CriticalSteelDesignResults ([in] long SteelDesignMemberId, [out] long* ResultsPerSection, [out] SAFEARRAY(RSteelDesignResult)* SteelDesignResults, [out] BSTR* Combination)
SteelDesignMemberId	index of steel design member
ResultsPerSection	Number of results per section
SteelDesignResults	The form of the one dimensional array with ResultsPerSection*n items: [1..ResultsPerSection],[ResultsPerSection+1..2*ResultsPerSection],...,[((n-1)*ResultsPerSection..n*ResultsPerSection], where between [] are the results of one section. See here
	PosX is relative distance from start node of the line
Combination	Name of critical combination
	Returns ResultsPerSection*n where n is number of sections, otherwise returns an error code (errDatabaseNotReady , errIndexOutOfBounds , sdreInvalidAnalysisType , sdreCOMError , sdreCombinationTypeNotValidForCurrentNationalDesignCode).
long	AllSteelDesignResultsByLoadCaselD ([out] SAFEARRAY(long)* SectionCounts, [out] long * ResultsPerSection, [out] SAFEARRAY(RSteelDesignResult)* SteelDesignResults, [out] SAFEARRAY(BSTR)* Combinations)
SectionCounts	Array (long) with count of sections for each steel design member
ResultsPerSection	Number of results per section
SteelDesignResults	The form of the one dimensional array with N*ResultsPerSection*n items: <ul style="list-style-type: none"> - First SteelDesignMember results: [1..ResultsPerSection],[ResultsPerSection+1..2*ResultsPerSection],...,[((n-1)*ResultsPerSection..n*ResultsPerSection], where between [] are the results of one section. - Then continues the sameway with next steel design member Until the last steel design member. See here PosX is relative distance from start node of the line
Combinations	Array of strings with loadcase names
	Returns N*ResultsPerSection*n where N is number of steel design members and n is number of sections. Otherwise returns an error code (errDatabaseNotReady , sdreLoadCaselDIndexOutOfBounds , sdreInvalidAnalysisType , sdreCOMError).

long	AllSteelDesignResultsByLoadCombinationId ([out] SAFEARRAY(long)* SectionCounts , [out] long* ResultsPerSection , [out] SAFEARRAY(RSteelDesignResult)* SteelDesignResults , [out] SAFEARRAY(BSTR)* Combinations)
	<p>SectionCounts Array (long) with count of sections for each steel design member</p> <p>ResultsPerSection Number of results per section</p> <p>SteelDesignResults The form of the one dimensional array with $N * \text{ResultsPerSection} * n$ items:</p> <ul style="list-style-type: none"> - First SteelDesignMember results: [1..ResultsPerSection], [ResultsPerSection+1..2 * ResultsPerSection], ..., [(n-1) * ResultsPerSection..n * ResultsPerSection], where between [] are the results of one section. - Then continues the sameway with next steel design member <p>Until the last steel design member. See here</p> <p>PosX is relative distance from start node of the line</p> <p>Combinations Array of strings with combinations</p>
	<p>Returns $N * \text{ResultsPerSection} * n$ where N is number of steel design members and n is number of sections.</p> <p>Otherwise returns an error code (errDatabaseNotReady, sdreLoadCaseIdIndexOutOfBoundsException, sdreInvalidAnalysisType, sdreCOMError).</p>
long	AllEnvelopeSteelDesignResults ([out] SAFEARRAY(long)* SectionCounts , [out] long* ResultsPerSection , [out] SAFEARRAY(RSteelDesignResult)* SteelDesignResults , [out] SAFEARRAY(BSTR)* Combinations)
	<p>SectionCounts Array (long) with count of sections for each steel design member</p> <p>ResultsPerSection Number of results per section</p> <p>SteelDesignResults The form of the one dimensional array with $N * \text{ResultsPerSection} * n$ items:</p> <ul style="list-style-type: none"> - First SteelDesignMember results: [1..ResultsPerSection], [ResultsPerSection+1..2 * ResultsPerSection], ..., [(n-1) * ResultsPerSection..n * ResultsPerSection], where between [] are the results of one section. - Then continues the sameway with next steel design member <p>Until the last steel design member. See here</p> <p>PosX is relative distance from start node of the line</p> <p>Combinations Array with names of combinations</p>
	<p>Envelope is identified by EnvelopeUID property. Returns $N * \text{ResultsPerSection} * n$ where N is number of steel design members and n is number of sections. Otherwise returns an error code (errDatabaseNotReady, sdreLoadCaseIdIndexOutOfBoundsException, sdreInvalidAnalysisType, sdreCOMError).</p>
long	AllCriticalSteelDesignResults ([out] SAFEARRAY(long) * SectionCounts , [out] long * ResultsPerSection , [out] SAFEARRAY(RSteelDesignResult) * SteelDesignResults , [out] SAFEARRAY(BSTR)* Combinations)
	<p>SectionCounts Array (long) with count of sections for each steel design member</p> <p>ResultsPerSection Number of results per section</p> <p>SteelDesignResults The form of the one dimensional array with $N * \text{ResultsPerSection} * n$ items:</p> <ul style="list-style-type: none"> - First steel design member results: [1..ResultsPerSection], [ResultsPerSection+1..2 * ResultsPerSection], ..., [(n-1) * ResultsPerSection..n * ResultsPerSection], where between [] are the results of one section. - Then continues the sameway with next steel design member <p>Until the last steel design member. See here</p> <p>PosX is relative distance from start node of the line</p> <p>Combinations Array with names of critical combinations</p>
	<p>Returns $N * \text{ResultsPerSection} * n$ where N is number of steel design members and n is number of sections.</p> <p>Otherwise returns an error code (errDatabaseNotReady, sdreLoadCaseIdIndexOutOfBoundsException, sdreInvalidAnalysisType, sdreCOMError, sdreCombinationTypeNotValidForCurrentNationalDesignCode).</p>

Properties

<u>EAnalysisType</u>	EAnalysisType • Get or set type of analysis
double	Count <i>Get number of steel design members in the model, if 0 then results not calculated or invalid.</i>
<u>ECombinationType</u>	ECombinationType • Get or set combination type
long	EnvelopeUID • Get or set the unique index of the envelope used in functions for reading envelope results
long	LoadCaseId • Get or set load case index ($0 < \text{LoadCaseId} \leq \text{AxisVMLoadCases.Count}$)
long	LoadCombinationId • Get or set load combination index ($0 < \text{LoadCaseId} \leq \text{AxisVMLoadCombinations.Count}$)
long	LoadLevel • Get or set load level (increment) index

IAxisVMTimberDesignResults

Interface for reading timber design results

If property returning this interface is null (nil) then the extension module TD1 is not available.

Error codes

enum	ETimberDesignResultsError = {	
	tdreCOMError = -100001	COM server error
	tdreLoadCaseIdIndexOutOfBounds = -100002	Invalid LoadCaseID while reading results
	tdreLoadCombinationIdIndexOutOfBounds = -100003	Invalid CombinationID while reading results
	tdreInvalidAnalysisType = -100004	Invalid type of results for used analysis
	tdreCombinationTypeNotValidForCurrentNationalDesignCode = -100005}	The CombinationType cannot be used for the selected national code. Some design codes allow only certain ECombinationTypes (see here) or results not available for CombinationType

Records / structures

	RTimberDesignResult = (
double	PosX	Position of the checked section [m]
double	DesignValue	Design value for the check (unit depends on type of check)
double	LimitValue	Limit value for the check (unit depends on type of check)
)	
		Timber design result for each check on a section according to the design code.

Type of timber design results in array (for national code EC5)

Array item No.	design value	limit value
1.	Efficiency (Axial Force, Bending (EC5 6.3.2, 6.2.4))	1,0
2.	Efficiency (Compression, Bending, and Flexural Buckling (EC5 6.3.2))	1,0
3.	Efficiency (Axial Force, Bending, and Lateral-Torsional Buckling (EC5 6.3.3))	1,0
4.	Efficiency (Shear(y)+Shear (z)+Torsion (There is no EC5 formula, a linear failure criteria is adopted))	1,0
5.	Efficiency (Tension90 + Shear in the Apex zone (perpendicular to the x axis) (EC5 6.4.3))	1,0
6.	0	Relative Slenderness Ratio about y; EN 1995-1:2004 (6.21)
7.	0	Relative Slenderness Ratio about z; EN 1995-1:2004 (6.22)
8.	0	Relative Slenderness Ratio for bending; EN 1995-1:2004 (6.30)
9.	0	kc,y instability factor; EN 1995-1:2004 (6.25)
10.	0	kc,z instability factor; EN 1995-1:2004 (6.26)
11.	0	K,crit factor for Lateral Torsional buckling; EN 1995-1:2004 (6.34)
12.	0	K,mod modification factor (duration,moisture content); EN 1995-1:2004 Table 3.1)
13.	0	Szigma,t,90,d tensile stress perpendicular to the grain; EN 1995-1:2004 (6.54)
14-15.	deprecated	
16.	Utilization in serviceability limit state (SLS) [-]	1,0
17.	Utilization in ultimate limit state (ULS) [-]	1,0

Functions

long	AllTimberDesignResultsByLoadCaseId ([out] SAFEARRAY(long)* SectionCounts , [out] long * ResultsPerSection , [out] SAFEARRAY(RTimberDesignResult) * TimberDesignResults , [out] BSTR* Combination)	
	SectionCounts	Array (long) with count of sections for each timber design member
	ResultsPerSection	Number of results per section
	TimberDesignResults	The form of the one dimensional array with ResultsPerSection*n items: [1..ResultsPerSection],[ResultsPerSection+1..2*ResultsPerSection],..., [(n-1)*ResultsPerSection..n*ResultsPerSection], where between [] are the results of one section. See here
		PosX is relative distance from start node of the line
	Combination	Name of loadcase
	Returns ResultsPerSection*n where n is number of sections, otherwise returns an error code (errDatabaseNotReady , errIndexOutOfBounds , errNotSupportedByNationalDesignCode , tdreInvalidAnalysisType , tdreLoadCaseIdIndexOutOfBounds , tdreCOMError).	
long	AllTimberDesignResultsByLoadCombinationId ([out] SAFEARRAY(long)* SectionCounts , [out] long ResultsPerSection , [out] SAFEARRAY(RTimberDesignResult) * TimberDesignResult , [out] BSTR* Combination)	
	SectionCounts	Array (long) with count of sections for each TimberDesignMember
	ResultsPerSection	Number of results per section
	TimberDesignResult	The form of the one dimensional array with ResultsPerSection*n items: [1..ResultsPerSection],[ResultsPerSection+1..2*ResultsPerSection],..., [(n-1)*ResultsPerSection..n*ResultsPerSection], where between [] are the results of one section. See here
		PosX is relative distance from start node of the line
	Combination	Name of combination
	Returns ResultsPerSection*n where n is number of sections, otherwise returns an error code (errDatabaseNotReady , errIndexOutOfBounds , errNotSupportedByNationalDesignCode , tdreInvalidAnalysisType , tdreLoadCombinationIdIndexOutOfBounds , tdreCOMError).	
long	AllCriticalTimberDesignResults ([out] SAFEARRAY(long)* SectionCounts , [out] long* ResultsPerSection , [out] SAFEARRAY(RTimberDesignResult) * TimberDesignResult , [out] BSTR* Combination)	
	SectionCounts	Array (long) with count of sections for each timber design member
	ResultsPerSection	Number of results per section
	TimberDesignResult	The form of the one dimensional array with ResultsPerSection*n items: [1..ResultsPerSection],[ResultsPerSection+1..2*ResultsPerSection],..., [(n-1)*ResultsPerSection..n*ResultsPerSection], where between [] are the results of one section. See here
		PosX is relative distance from start node of the line
	Combination	Name of critical combination
	Returns ResultsPerSection*n where n is number of sections, otherwise returns an error code (errDatabaseNotReady , errIndexOutOfBounds , tdreInvalidAnalysisType , tdreCOMError , tdreCombinationTypeNotValidForCurrentNationalDesignCode).	

long **AllEnvelopeTimberDesignResults** ([out] SAFEARRAY(long)* **SectionCounts**,
[out] long* **ResultsPerSection**, [out] SAFEARRAY([RTimberDesignResult](#))* **TimberDesignResult**,
[out] BSTR* **Combination**)

SectionCounts Array (long) with count of sections for each timber design member

ResultsPerSection Number of results per section

TimberDesignResult The form of the one dimensional array with **ResultsPerSection***n items:
[1..ResultsPerSection],[ResultsPerSection+1..2*ResultsPerSection],...,
[(n-1)*ResultsPerSection..n*ResultsPerSection], where between [] are
the results of one section. See [here](#)

PosX is relative distance from start node of the line

Combination Name of critical combination

Envelope is identified by EnvelopeUID property. Returns ResultsPerSection*n where n is number of sections, otherwise returns an error code ([errDatabaseNotReady](#), [errIndexOutOfBoundsException](#), [tdreInvalidAnalysisType](#), [tdreCOMError](#)).

long **GetEfficiencyAndCombination** ([in] long **TimberDesignMemberId**, [in] [EResultType](#) **ResultType**,
[out] double **Efficiency**, [out] BSTR **Combination**)

TimberDesignMemberId index of timber design member

ResultType type of result

Efficiency maximum efficiency

Combination name of load case or combination corresponding to max. efficiency

Get efficiency and combination depending on result type and set interface properties.

Returns TimberDesignMemberId if successful, otherwise returns an error code
([errDatabaseNotReady](#), [errIndexOutOfBoundsException](#), [tdreLoadCaseIdOutOfBoundsException](#),
[tdreInvalidAnalysisType](#), [tdreLoadCombinationIdOutOfBoundsException](#)).

long **GetEfficiencyAndCombinationByLoadCaseId** ([in] long **TimberDesignMemberId**,
[in] long **LoadCaseId**, [in] long **LoadLevel**, [out] double **Efficiency**, [out] BSTR **Combination**)

TimberDesignMemberId index of timber design member

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

LoadLevel load level (increment) index

Efficiency maximum efficiency

Combination name of load case or combination corresponding to max. efficiency

Get efficiency and combination depending on load case. Returns TimberDesignMemberId if successful, otherwise returns an error code ([errDatabaseNotReady](#), [errIndexOutOfBoundsException](#), [tdreLoadCaseIdOutOfBoundsException](#), [sdreLoadCaseIdOutOfBoundsException](#)).

long **GetEfficiencyAndCombinationByLoadCombinationId** ([in] long **TimberDesignMemberId**,
[in] long **LoadCombinationId**, [in] long **LoadLevel**, [out] double **Efficiency**,
[out] BSTR **Combination**)

TimberDesignMemberId index of timber design member

LoadCombinationId load combination index (0 < LoadCombinationId ≤ [AxisVMLoadCombinations.Count](#))

LoadLevel load level (increment) index

Efficiency maximum efficiency

Combination name of load case or combination corresponding to max. efficiency

Get efficiency and combination depending on load combination. Returns TimberDesignMemberId if successful, otherwise returns an error code ([errDatabaseNotReady](#), [errIndexOutOfBoundsException](#), [tdreLoadCombinationIdOutOfBoundsException](#), [tdreInvalidAnalysisType](#)).

long	GetEnvelopeEfficiencyAndCombination ([in] long TimberDesignMemberId, [in] long EnvelopeUID, [out] double Efficiency, [out] BSTR Combination)	
	TimberDesignMemberId index of timber design member	
	EnvelopeUID unique index of the envelope	
	Efficiency maximum efficiency	
	Combination name of load case or combination corresponding to max. efficiency	
	Get efficiency and combination depending on envelope. Returns TimberDesignMemberId if successful, otherwise returns an error code (errDatabaseNotReady , errIndexOutOfBounds , tdreInvalidAnalysisType).	
long	GetCriticalEfficiencyAndCombination ([in] long TimberDesignMemberId, [in] ECombinationType CombinationType, [out] double Efficiency, [out] BSTR Combination)	
	TimberDesignMemberId index of timber design member	
	CombinationType combination type	
	Efficiency maximum efficiency	
	Combination name of load case or combination corresponding to max. efficiency	
	Get efficiency and combination depending on critical combination type. Returns TimberDesignMemberId if successful, otherwise returns an error code (errDatabaseNotReady , errIndexOutOfBounds , tdreInvalidAnalysisType , errCriticalCombinationNotAllowed , tdreCombinationTypeNotValidForCurrentNationalDesignCode).	
long	GetTimberDesignResultsByLoadCaseld ([in] long TimberDesignMemberId, [in] long LoadCaseld, [in] long LoadLevel, [in] EAnalysisType AnalysisType, [out] long ResultsPerSection, [out] SAFEARRAY(RTimberDesignResult)* TimberDesignResult, [out] BSTR* Combination)	
	TimberDesignMemberId index of timber design member	
	LoadCaseld load case index ($0 < \text{LoadCaseld} \leq \text{AxisVMLoadCases.Count}$)	
	LoadLevel load level (increment) index	
	AnalysisType Type of Analysis	
	ResultsPerSection Number of results per section	
	TimberDesignResult The form of the one dimensional array with ResultsPerSection*n items: [1..ResultsPerSection], [ResultsPerSection+1..2*ResultsPerSection], ..., [(n-1)*ResultsPerSection..n*ResultsPerSection], where between [] are the results of one section. See here	
	Combination PosX is relative distance from start node of the line	
	Combination Name of loadcase	
	Returns ResultsPerSection*n where n is number of sections, otherwise returns an error code (errDatabaseNotReady , errIndexOutOfBounds , errNotSupportedByNationalDesignCode , tdreInvalidAnalysisType , tdreLoadCaseldIndexOutOfBounds , tdreCOMError).	
long	GetTimberDesignResultsByLoadCaseld_Abs ([in] long TimberDesignMemberId, [in] long LoadCaseld, [in] long LoadLevel, [in] EAnalysisType AnalysisType, [out] long ResultsPerSection, [out] SAFEARRAY(RTimberDesignResult)* TimberDesignResult, [out] BSTR* Combination)	
	Same as GetTimberDesignResultsByLoadCaseld, but PosX is absolute distance from start node of the steel design member	

long	GetTimberDesignResultsByLoadCombinationId ([in] long TimberDesignMemberId, [in] long LoadCombinationId, [in] long LoadLevel, [in] EAnalysisType AnalysisType, [out] long ResultsPerSection, [out] SAFEARRAY(RTimberDesignResult)* TimberDesignResult, [out] BSTR* Combination)
	TimberDesignMemberId index of timber design member LoadCombinationId load combination index ($0 < \text{LoadCombinationId} \leq \text{AxisVMLoadCombinations.Count}$) LoadLevel load level (increment) index AnalysisType Type of Analysis ResultsPerSection Number of results per section TimberDesignResult The form of the one dimensional array with $\text{ResultsPerSection}^n$ items: $[1..\text{ResultsPerSection}], [\text{ResultsPerSection}+1..2*\text{ResultsPerSection}], \dots, [(n-1)*\text{ResultsPerSection}..n*\text{ResultsPerSection}]$, where between [] are the results of one section. See here PosX is relative distance from start node of the line Combination Name of combination
	<i>Returns ResultsPerSectionⁿ where n is number of sections, otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBounds, errNotSupportedByNationalDesignCode, tdreInvalidAnalysisType, tdreLoadCombinationIdOutOfBounds, tdreCOMError).</i>
long	GetTimberDesignResultsByLoadCombinationId_Abs ([in] long TimberDesignMemberId, [in] long LoadCombinationId, [in] long LoadLevel, [in] EAnalysisType AnalysisType, [out] long ResultsPerSection, [out] SAFEARRAY(RTimberDesignResult)* TimberDesignResult, [out] BSTR* Combination)
	<i>Same as GetTimberDesignResultsByLoadCombinationId, but PosX is absolute distance from start node of the steel design member</i>
long	GetEnvelopeTimberDesignResults ([in] long TimberDesignMemberId, [in] EAnalysisType AnalysisType, [out] long* ResultsPerSection, [out] SAFEARRAY(RTimberDesignResult)* TimberDesignResult, [out] BSTR* Combination)
	TimberDesignMemberId index of timber design member AnalysisType Type of Analysis ResultsPerSection Number of results per section TimberDesignResult The form of the one dimensional array with $\text{ResultsPerSection}^n$ items: $[1..\text{ResultsPerSection}], [\text{ResultsPerSection}+1..2*\text{ResultsPerSection}], \dots, [(n-1)*\text{ResultsPerSection}..n*\text{ResultsPerSection}]$, where between [] are the results of one section. See here PosX is relative distance from start node of the line Combination Name of critical combination
	<i>Envelope is identified by EnvelopeUID property. Returns ResultsPerSectionⁿ where n is number of sections, otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBounds, tdreInvalidAnalysisType, tdreCOMError).</i>
long	GetEnvelopeTimberDesignResults_Abs ([in] long TimberDesignMemberId, [in] EAnalysisType AnalysisType, [out] long* ResultsPerSection, [out] SAFEARRAY(RTimberDesignResult)* TimberDesignResult, [out] BSTR* Combination)
	<i>Same as GetEnvelopeTimberDesignResults, but PosX is absolute distance from start node of the steel design member.</i>

long	GetEnvelopeTimberDesignResults2 ([in] long TimberDesignMemberId , [in] long EnvelopeUID , [in] EAnalysisType AnalysisType , [out] long* ResultsPerSection , [out] SAFEARRAY(RTimberDesignResult)* TimberDesignResult , [out] BSTR* Combination)
	<p>TimberDesignMemberId <i>index of timber design member</i></p> <p>EnvelopeUID <i>unique index of the envelope used in functions for reading envelope results</i></p> <p>AnalysisType <i>Type of Analysis</i></p> <p>ResultsPerSection <i>Number of results per section</i></p> <p>TimberDesignResult <i>The form of the one dimensional array with ResultsPerSection*n items: [1..ResultsPerSection],[ResultsPerSection+1..2*ResultsPerSection],..., [(n-1)*ResultsPerSection..n*ResultsPerSection], where between [] are the results of one section. See here</i></p> <p><i>PosX is relative distance from start node of the line</i></p> <p>Combination <i>Name of critical combination</i></p>
	<p><i>Envelope is identified by EnvelopeUID property. Returns ResultsPerSection*n where n is number of sections, otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBounds, tdreInvalidAnalysisType, tdreCOMError).</i></p>
long	GetCriticalTimberDesignResults ([in] long TimberDesignMemberId , [in] ECombinationType CombinationType , [in] EAnalysisType AnalysisType , [out] long* ResultsPerSection , [out] SAFEARRAY(RTimberDesignResult)* TimberDesignResult , [out] BSTR* Combination)
	<p>TimberDesignMemberId <i>index of timber design member</i></p> <p>CombinationType <i>Combination Type</i></p> <p>AnalysisType <i>Type of Analysis</i></p> <p>ResultsPerSection <i>Number of results per section</i></p> <p>TimberDesignResult <i>The form of the one dimensional array with ResultsPerSection*n items: [1..ResultsPerSection],[ResultsPerSection+1..2*ResultsPerSection],..., [(n-1)*ResultsPerSection..n*ResultsPerSection], where between [] are the results of one section. See here</i></p> <p><i>PosX is relative distance from start node of the line</i></p> <p>Combination <i>Name of critical combination</i></p>
	<p><i>Returns ResultsPerSection*n where n is number of sections, otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBounds, tdreInvalidAnalysisType, tdreCOMError, tdreCombinationTypeNotValidForCurrentNationalDesignCode).</i></p>
long	GetCriticalTimberDesignResults_Abs ([in] long TimberDesignMemberId , [in] ECombinationType CombinationType , [in] EAnalysisType AnalysisType , [out] long* ResultsPerSection , [out] SAFEARRAY(RTimberDesignResult)* TimberDesignResult , [out] BSTR* Combination)
	<p><i>Same as GetCriticalTimberDesignResults, but PosX is absolute distance from start node of the steel design member.</i></p>

long	GetAllTimberDesignResultsByLoadCaseId ([in] long LoadCaseId, [in] long LoadLevel, [in] EAnalysisType AnalysisType, [out] SAFEARRAY(long)* SectionCounts, [out] long * ResultsPerSection, [out] SAFEARRAY(RTimberDesignResult) * TimberDesignResults, [out] SAFEARRAY(BSTR)* Combinations)
	<p>LoadCaseId load case index ($0 < \text{LoadCaseId} \leq \text{AxisVMLoadCases.Count}$)</p> <p>LoadLevel load level (increment) index</p> <p>AnalysisType Type of Analysis</p> <p>SectionCounts Array (long) with count of sections for each timber design member</p> <p>ResultsPerSection Number of results per section</p> <p>TimberDesignResults The form of the one dimensional array with $N * \text{ResultsPerSection} * n$ items:</p> <ul style="list-style-type: none"> - First SteelDesignMember results: [1..ResultsPerSection], [ResultsPerSection + 1..2*ResultsPerSection], ..., [(n-1)*ResultsPerSection..n*ResultsPerSection], where between [] are the results of one section. - Then continues the sameway with next timber design member - Until the last timber design member <p>See here</p> <p>PosX is relative distance from start node of the line</p>
	<p>Combinations Array of loadcase names</p> <p>Returns $N * \text{ResultsPerSection} * n$ where N is number of timber design members and n is number of sections. Otherwise returns an error code (errDatabaseNotReady, tdreInvalidAnalysisType, tdreCOMError, tdreLoadCaseIdIndexOutOfBounds).</p>
long	GetAllTimberDesignResultsByLoadCombinationId ([in] long LoadCombinationId, [in] long LoadLevel, [in] EAnalysisType AnalysisType, [out] SAFEARRAY(long)* SectionCounts, [out] long * ResultsPerSection, [out] SAFEARRAY(RTimberDesignResult) * TimberDesignResults, [out] SAFEARRAY(BSTR)* Combinations)
	<p>LoadCombinationId load combination index ($0 < \text{LoadCombinationId} \leq \text{AxisVMLoadCombinations.Count}$)</p> <p>LoadLevel load level (increment) index</p> <p>AnalysisType Type of Analysis</p> <p>SectionCounts Array (long) with count of sections for each timber design member</p> <p>ResultsPerSection Number of results per section</p> <p>TimberDesignResults The form of the one dimensional array with $N * \text{ResultsPerSection} * n$ items:</p> <ul style="list-style-type: none"> - First SteelDesignMember results: [1..ResultsPerSection], [ResultsPerSection+1..2*ResultsPerSection], ..., [(n-1)*ResultsPerSection..n*ResultsPerSection], where between [] are the results of one section. - Then continues the sameway with next timber design member - Until the last timber design member <p>See here</p> <p>PosX is relative distance from start node of the line</p>
	<p>Combinations Array of loadcase names</p> <p>Returns $N * \text{ResultsPerSection} * n$ where N is number of timber design members and n is number of sections. Otherwise returns an error code (errDatabaseNotReady, tdreInvalidAnalysisType, tdreCOMError, tdreLoadCombinationIdIndexOutOfBounds).</p>

long	GetAllEnvelopeTimberDesignResults ([in] EAnalysisType AnalysisType, [out] SAFEARRAY(long)* SectionCounts , [out] long* ResultsPerSection , [out] SAFEARRAY(RTimberDesignResult) * TimberDesignResults , [out] SAFEARRAY(BSTR)* Combinations)
	<p>AnalysisType <i>Type of Analysis</i></p> <p>SectionCounts <i>Array (long) with count of sections for each timber design member</i></p> <p>ResultsPerSection <i>Number of results per section</i></p> <p>TimberDesignResults <i>The form of the one dimensional array with N*ResultsPerSection*n items:</i></p> <ul style="list-style-type: none"> - <i>First SteelDesignMember results:</i> $[1..ResultsPerSection],[ResultsPerSection+1..2*ResultsPerSection]$ $,...,[(n-1)*ResultsPerSection..n*ResultsPerSection]$, where between $[]$ are the results of one section. - <i>Then continues the sameway with next timber design member</i> <p><i>Until the last timber design member. See here</i></p> <p><i>PosX is relative distance from start node of the line</i></p> <p>Combinations <i>Array of strings with names of critical combinations</i></p> <p><i>Envelope is identified by EnvelopeUID property. Returns N*ResultsPerSection*n where N is number of timber design members and n is number of sections. Otherwise returns an error code (errDatabaseNotReady, tdreInvalidAnalysisType, tdreCOMError).</i></p>
long	GetAllCriticalTimberDesignResults ([in] ECombinationType CombinationType, [in] EAnalysisType AnalysisType, [out] SAFEARRAY(long) * SectionCounts , [out] long * ResultsPerSection , [out] SAFEARRAY(RTimberDesignResult) * TimberDesignResults , [out] SAFEARRAY(BSTR)* Combinations)
	<p>CombinationType <i>Combination Type</i></p> <p>AnalysisType <i>Type of Analysis</i></p> <p>SectionCounts <i>Array (long) with count of sections for each timber design member</i></p> <p>ResultsPerSection <i>Number of results per section</i></p> <p>TimberDesignResults <i>The form of the one dimensional array with N*ResultsPerSection*n items:</i></p> <ul style="list-style-type: none"> - <i>First SteelDesignMember results:</i> $[1..ResultsPerSection],[ResultsPerSection+1..2*ResultsPerSection]$, $,...,[(n-1)*ResultsPerSection..n*ResultsPerSection]$, where between $[]$ are the results of one section. - <i>Then continues the sameway with next timber design member</i> <p><i>Until the last timber design member. See here</i></p> <p><i>PosX is relative distance from start node of the line</i></p> <p>Combinations <i>Array with names of combinations</i></p> <p><i>Returns N*ResultsPerSection*n where N is number of timber design members and n is number of sections. Otherwise returns an error code (errDatabaseNotReady, tdreInvalidAnalysisType, tdreCOMError, tdreCombinationTypeNotValidForCurrentNationalDesignCode).</i></p>
long	TimberDesignResultsByLoadCaseId ([in] long TimberDesignMemberId , [out] long * ResultsPerSection , [out] SAFEARRAY(RTimberDesignResult) * TimberDesignResults , [out] BSTR* Combination)
	<p>TimberDesignMemberId <i>index of timber design member</i></p> <p>ResultsPerSection <i>Number of results per section</i></p> <p>TimberDesignResults <i>The form of the one dimensional array with ResultsPerSection*n items:</i> $[1..ResultsPerSection],[ResultsPerSection+1..2*ResultsPerSection]$, $,...,[(n-1)*ResultsPerSection..n*ResultsPerSection]$, where between $[]$ are the results of one section. See here</p> <p><i>PosX is relative distance from start node of the line</i></p> <p>Combination <i>Name of loadcase</i></p>

Returns `ResultsPerSection*n` where `n` is number of sections, otherwise returns an error code ([errDatabaseNotReady](#), [errIndexOutOfBounds](#), [errNotSupportedByNationalDesignCode](#), [tdreInvalidAnalysisType](#), [tdreLoadCaseIdIndexOutOfBounds](#), [tdreCOMError](#)).

long	TimberDesignResultsByLoadCombinationId ([in] long TimberDesignMemberId , [out] long ResultsPerSection , [out] SAFEARRAY(RTimberDesignResult)* TimberDesignResult , [out] BSTR* Combination)
	TimberDesignMemberId index of timber design member
	ResultsPerSection Number of results per section
	TimberDesignResult The form of the one dimensional array with <code>ResultsPerSection*n</code> items: [1.. <code>ResultsPerSection</code>], [<code>ResultsPerSection+1..2*ResultsPerSection</code>], ..., [<code>(n-1)*ResultsPerSection..n*ResultsPerSection</code>], where between [] are the results of one section. See here
	PosX is relative distance from start node of the line
	Combination Name of combination

Returns `ResultsPerSection*n` where `n` is number of sections, otherwise returns an error code ([errDatabaseNotReady](#), [errIndexOutOfBounds](#), [errNotSupportedByNationalDesignCode](#), [tdreInvalidAnalysisType](#), [tdreLoadCombinationIdIndexOutOfBounds](#), [tdreCOMError](#)).

long	CriticalTimberDesignResults ([in] long TimberDesignMemberId , [out] long* ResultsPerSection , [out] SAFEARRAY(RTimberDesignResult)* TimberDesignResult , [out] BSTR* Combination)
	TimberDesignMemberId index of timber design member
	ResultsPerSection Number of results per section
	TimberDesignResult The form of the one dimensional array with <code>ResultsPerSection*n</code> items: [1.. <code>ResultsPerSection</code>], [<code>ResultsPerSection+1..2*ResultsPerSection</code>], ..., [<code>(n-1)*ResultsPerSection..n*ResultsPerSection</code>], where between [] are the results of one section. See here
	PosX is relative distance from start node of the line
	Combination Name of critical combination

Returns `ResultsPerSection*n` where `n` is number of sections, otherwise returns an error code ([errDatabaseNotReady](#), [errIndexOutOfBounds](#), [tdreInvalidAnalysisType](#), [tdreCOMError](#), [tdreCombinationTypeNotValidForCurrentNationalDesignCode](#)).

long	EnvelopeTimberDesignResults ([in] long TimberDesignMemberId , [out] long* ResultsPerSection , [out] SAFEARRAY(RTimberDesignResult)* TimberDesignResult , [out] BSTR* Combination)
	TimberDesignMemberId index of timber design member
	TimberDesignResult The form of the one dimensional array with <code>ResultsPerSection*n</code> items: [1.. <code>ResultsPerSection</code>], [<code>ResultsPerSection+1..2*ResultsPerSection</code>], ..., [<code>(n-1)*ResultsPerSection..n*ResultsPerSection</code>], where between [] are the results of one section. See here
	PosX is relative distance from start node of the line
	Combination Name of critical combination

Envelope is identified by `EnvelopeUID` property. Returns `ResultsPerSection*n` where `n` is number of sections, otherwise returns an error code ([errDatabaseNotReady](#), [errIndexOutOfBounds](#), [tdreInvalidAnalysisType](#), [tdreCOMError](#)).

Properties

EAnalysisType	AnalysisType • Get or set type of analysis
double	Count
	Get number of timber design members in the model, if 0 then results not calculated or invalid.
ECombinationType	CombinationType • Get or set the combination type
long	EnvelopeUID • Get or set the unique index of the envelope used in functions for reading envelope results

long **LoadCaseId** • Get or set load case index
 $0 < LoadCaseId \leq \text{AxisVMLoadCases.Count}$

long **LoadCombinationId** • Get or set the load combination index
 $0 < LoadCombinationId \leq \text{AxisVMLoadCombinations.Count}$

long **LoadLevel** • Get or set load level (increment) index

IAxisVMVelocity

Interface containing velocity results within the model.

If property returning this interface is null (nil) then the extension module DYN is not available.

Error codes

```
enum EVelocityError = {
    veeLoadCaseldIndexOutOfBounds = -100001           LoadCaseld is out of bounds
    veeInvalidCombinationOfLoadCaseAndTimeStep = -100002
    veeInvalidAnalysisType = -100003                  invalid combination of LoadCaseld and TimeStep
    veeLoadCombinationHasNoDynamicResult = -100004 }   AnalysisType is incompatible with the function
                                                       dynamic results for the LoadCaseld are missing
```

Enumerated types

```
enum EVelocity = {
    veX = 0      velocity in local x direction
    veY = 1      velocity in local y direction
    veZ = 2      velocity in local z direction
    veXX = 3     angular velocity about local x direction
    veYY = 4     angular velocity about local y direction
    veZZ = 5     angular velocity about local z direction
    veR = 6      resultant velocity
    veRR = 7 }   resultant angular velocity
    Velocity types
```

Records / structures

```
RVelocityValues = (
    double vX   velocity in local x direction [m/s]
    double vY   velocity in local y direction [m/s]
    double vZ   velocity in local z direction [m/s]
    double vXX  angular velocity about local x direction [rad/s]
    double vYY  angular velocity about local y direction [rad/s]
    double vZZ  angular velocity about local z direction [rad/s]
    double vR   resultant velocity [m/s]
    double vRR  resultant angular velocity [rad/s]
)
```

Functions

```
long NodalVelocityByLoadCaseld ([in] long Nodeld,
                                [i/o] RVelocityValues VelocityValues, [out] BSTR Combination)
```

Nodeld node index or line midpoint index
($0 < \text{Nodeld} \leq \text{AxisVMMModel.Nodes.Count}$) or a value returned by AxisVMMModel.Lines.MidpointId[LineIndex]

VelocityValues velocity results

Combination name of the load case

Retrieves velocity values of a node or line midpoint according to the LoadCaseld (and TimeStep) property. Returns Nodeld if successful l, otherwise an error code code ([errDatabaseNotReady](#), [errIndexOutOfBoundsException](#), [veeLoadCaseldIndexOutOfBoundsException](#), [veeInvalidCombinationOfLoadCaseAndTimeStep](#), [veeInvalidAnalysisType](#), [veeLoadCombinationHasNoDynamicResult](#)).

long **EnvelopeNodalVelocity** ([in] long **NodeId**,
[i/o] **RVelocityValues** **VelocityValues**, [out] BSTR **Combination**)

NodeId node index or line midpoint index

($0 < \text{NodeId} \leq \text{AxisVMMModel.Nodes.Count}$) or a value returned by
`AxisVMMModel.Lines.MidpointId[LineIndex]`

VelocityValues velocity results

Combination name of the load case

Retrieves velocity values of a node or line midpoint. Envelope is identified by `EnvelopeUID`, `Component` and `MinMaxType` properties. Returns `NodeId` if successful 1, otherwise an error code ([errDatabaseNotReady](#), [errIndexOutOfBoundsException](#), [veeLoadCaseIdIndexOutOfBoundsException](#), [veeInvalidCombinationOfLoadCaseAndTimeStep](#), [veeInvalidAnalysisType](#), [veeLoadCombinationHasNoDynamicResult](#)).

Properties

EAnalysisType	AnalysisType • Get or set type of analysis
EVelocity	Component • Get or set velocity component. Used for critical and envelope min/max values .
long	EnvelopeUID • Get or set unique index of the envelope used in functions for reading envelope results
long	LoadCaseId • Get or set load case index ($0 < \text{LoadCaseId} \leq \text{AxisVMMModel.LoadCases.Count}$)
EMaxMinType	MinMaxType • Get or set if minimum or maximum values of the component should be read
long	TimeStep • Get or set time step increment.

IAxisVMRebarSteelGrades

Intefrace for defining rebar steel grades in the model.

Error codes

```
enum ERebarSteelGradesError = {
    rsgeIllegalNationalDesignCode = -100001      process information is not compatible with the shape
    rsgeNonPositive_E = -100002                   E ≤ 0
    rsgeNonPositive_ssh = -100003                 ssh ≤ 0

    rsgeNonPositive_es0 = -100004                 es0 ≤ 0
    rsgeNonPositive_esh = -100005                 esh ≤ 0
    rsgeNonPositive_fyd = -100006                 fyd ≤ 0
    rsgeNonPositive_es1= -100007                  es1 ≤ 0
    rsgeNonPositive_esu = -100008                 esu ≤ 0
    rsgeNonPositive_Ra = -100009                 Ra ≤ 0
    rsgeNonPositive_mat = -100010                 mat ≤ 0
    rsgeNonPositive_fsrep = -100011                fsrep ≤ 0
    rsgeNonPositive_fs = -100012                  fs ≤ 0
    rsgeNonPositive_fyk = -100013                 fyk ≤ 0
    rsgeNonPositive_Epsuk = -100014                Epsuk ≤ 0
    rsgeNonPositive_GammaS = -100015               GammaS ≤ 0
    rsgeNonPositive_fsk = -100016                 Fsk ≤ 0
    rsgeNonPositive_ks = -100017                  Ks ≤ 0
    rsgeNonPositive_Epsud = -100018                Epsud ≤ 0
    rsgeNotFound = -100019                         Rebate steel grade not found
}
```

Rebar steel grades error codes.

Records / structures

```
RRebarSteelGrade_EC_ITA = (
    double fyd                      limiting stress
    double es1                      elastic limiting strain
    double esu                      plastic limiting strain
)

RRebarSteelGrade_MSZ = (
    double ssh                      Sigma sh limit stress
    double es0                      limiting strain, see MSZ design code
    double esh                      limiting strain, see MSZ design code
)

RRebarSteelGrade_STAS = (
    double Ra                       limit stress
    double es1                      elastic limiting strain
    double esu                      plastic limiting strain
    double mat                      see STAS design code
)

RRebarSteelGrade_DIN = (
    double fyk                      design yield strength
    double Epsuk                     strain
    double GammaS                   partial safety factor
)

RRebarSteelGrade_SIA
= (
    double fsk                      yield strength
    double ks                       see SIA design code
    double Epsuk                     limiting strain, see SIA design code
    double Epsud                     limiting strain, see SIA design code
    double GammaS                   partial safety factor
)

RRebarSteelGrade_NEN
= (
    double fsrep                    characteristics strength
    double fs                       design strength
    double esu                      limiting strain, see NEN design code
)
```

<code>ENationalDesignCode</code>	<code>RRebarSteelGrade = (</code>	<i>National Design Code</i>
double	<code>NationalDesignCode</code>	<i>Young's modulus of elasticity [kN/m²]</i>
<code>RRebarSteelGrade_EC_ITA</code>	<code>E</code>	<i>Rebar steel grade parameters according to EC & ITA national design codes</i>
	<code>RebarSteelGrade_MSZ</code>	<i>Rebar steel grade parameters according MSZ national design codes</i>
<code>RRebarSteelGrade_MSZ</code>	<code>RebarSteelGrade_STAS</code>	<i>Rebar steel grade parameters according to STAS national design codes</i>
<code>RRebarSteelGrade_STAS</code>	<code>RebarSteelGrade_DIN</code>	<i>Rebar steel grade parameters according to DIN national design codes</i>
<code>RRebarSteelGrade_DIN</code>	<code>RebarSteelGrade_SIA</code>	<i>Rebar steel grade parameters according to SIA national design codes</i>
<code>RRebarSteelGrade_SIA</code>	<code>RebarSteelGrade_NEN</code>	<i>Rebar steel grade parameters according to NEN national design codes</i>
<code>RRebarSteelGrade_NEN</code>)	

Functions

long	Add ([in] BSTR Name, [i/o] <code>RRebarSteelGrade</code> RebarSteelGrade)	
	Name name of rebar steel grade	
	RebarSteelGrade rebar steel grade parameters	
	Adds a new rebar steel grade. If successful, returns the rebar steel grade index, otherwise returns an error code (errDatabaseNotReady).	
long	AddFromCatalog ([in] <code>ENationalDesignCode</code> NationalDesignCode, [in] BSTR Name)	
	NationalDesignCode national design code of the steel grade	
	Name name of the rebar steel grade	
	Adds a new rebar steel grade from the catalog. If successful, returns the rebar steel index, otherwise returns an error code(errDatabaseNotReady , rsgeNotFound , errInternalException).	
long	AddFromCatalogFile ([in] <code>ENationalDesignCode</code> NationalDesignCode, [in] BSTR FileName, [in] BSTR Name)	
	NationalDesignCode national design code of the steel grade	
	FileName name of the catalog file	
	Name name of the steel grade	
	Adds a new rebar steel grade from the catalog file. If successful, returns the rebar steel grade index, otherwise returns an error code (errDatabaseNotReady , rsgeNotFound , errInternalException).	
long	Clear	
	Clear everything from the interface. If successful, returns number of deleted rebar steel grades, otherwise returns an error code (errDatabaseNotReady).	
long	Delete ([in] long Index)	
	Index index of the rebar steel grade	
	Deletes rebar steel grade by index. If successful, returns the rebar steel grade index, otherwise returns an error code (errDatabaseNotReady , errIndexOutOfBoundsException).	
long	GetData ([in] long Index, [out] BSTR Name, [i/o] <code>RRebarSteelGrade</code> RebarSteelGrade)	
	Index index of the rebar steel grade	
	Name name of the rebar steel grade	
	RebarSteelGrade rebar steel grade parameters	
	Get rebar steel material parameters. If successful, returns the rebar steel grade index, otherwise returns an error code(errDatabaseNotReady , errIndexOutOfBoundsException).	

long **IndexOf** ([in] BSTR Name)

Name name of the rebar steel grade

Get index of the steel grade by name. If successful, returns the rebar steel grade index, otherwise returns an error code ([errDatabaseNotReady](#), [rsgeNotFound](#)).

long **SetData** ([in] long Index, [in] BSTR Name, [i/o] RRebarSteelGrade RebarSteelGrade)

Index index of the rebar steel grade

Name name of the rebar steel grade

RebarSteelGrade rebar steel grade parameters

Set rebar steel material parameters. If successful, returns the rebar steel grade index, otherwise returns an error code ([errDatabaseNotReady](#), [errIndexOutOfBounds](#) or from [ERebarSteelGradesError](#)).

Properties

long **Count** Get number of rebar steel grades in the model

long **IndexOfUID** [long UID] Get index of the rebar steel grade

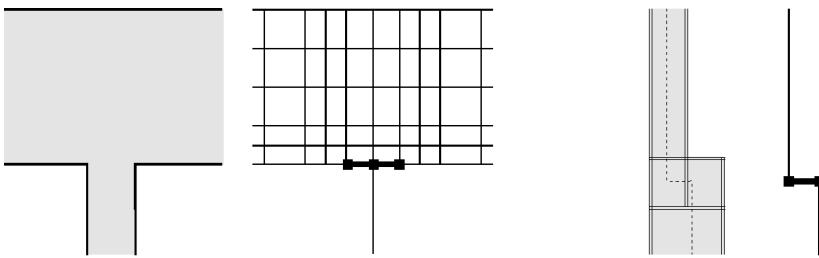
UID unique index of the rebar steel grade

[**ENationalDesignCode**](#) **NationalDesignCode** Get national design code of rebar steel grade in the model

long **UID** Get unique index of the rebar steel grade which remains the same while exists in the model (read only property)

IAxisVMRigidBodies

Interface used for defining rigid bodies in the model.



Error codes

enum	ERigidBodiesError = { rbeLineListIsEmpty = -100001 rbeNoLinesAreSelected = -100002 rbeLineIndexOutOfBounds = -100003}	<i>array with LineIDs is empty when none of the lines is selected when LineID is out bounds</i>
------	--	---

Functions

long	Add ([in] SAFEARRAY(long) LineIDs) LineIDs Array with LineIDs <i>If successful returns number of rigid bodies, otherwise returns an error code (errDatabaseNotReady, rbeLineIndexOutOfBounds)</i>
long	Add_vb (Visual Basic compatible function of Add)
long	AddSelectedLines <i>If successful returns number of rigid bodies, otherwise returns an error code (errDatabaseNotReady, rbeNoLinesAreSelected)</i>
long	Clear <i>If successful returns number of rigid bodies before delete, otherwise returns an error code (errDatabaseNotReady)</i>
long	DefineSelectedLinesAsRigidBody <i>If successful returns number of rigid bodies, otherwise returns an error code (errDatabaseNotReady, rbeNoLinesAreSelected)</i>
long	Delete ([in] long Index) Index index of the rigid body, $1 \leq \text{Index} \leq \text{Count}$ <i>If successful returns index, otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBounds)</i>
long	GetLines ([in] long Index , [out] SAFEARRAY(long) LineIDs) Index index of the rigid body, $1 \leq \text{Index} \leq \text{Count}$ LineIDs Array with LineIDs <i>If successful returns number of lines in rigid body with index Index, otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBounds)</i>
long	RemoveLinesFromRigidBodies ([in] SAFEARRAY(long) LineIDs) LineIDs Array with LineIDs <i>If successful returns number of rigid bodies, otherwise returns an error code (errDatabaseNotReady, rbeLineIndexOutOfBounds, rbeLineListIsEmpty)</i>
long	RemoveLinesFromRigidBodies_vb (Visual Basic compatible function of RemoveLinesFromRigidBodies)

Properties

long **Count** *Get number of rigid bodies in the model.*

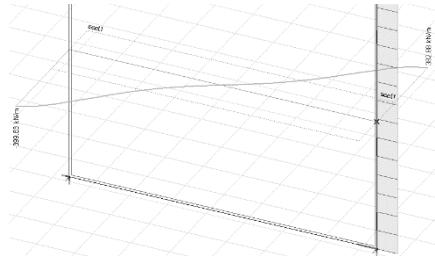
IAxisVMSections

Interface for sections on the model.

Enumerated types

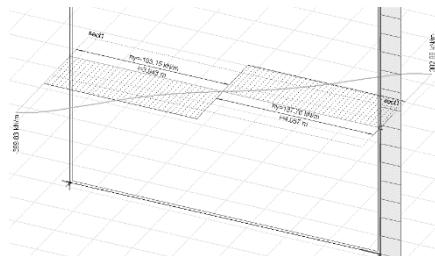
```
enum ESectionType = {  
    stPlane= 0x00  
    stSegment= 0x01 }  
  
enum ESectionDisplayMode= {  
    sdmDiagramOnly = 0x00
```

Displays diagram only



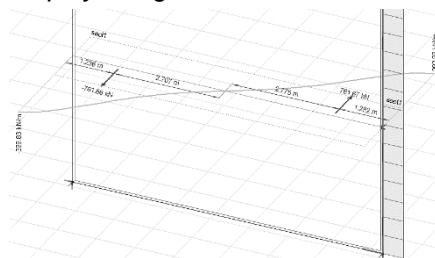
sdmDiagramAvg = 0x01

Displays average diagram of positive and negative values.



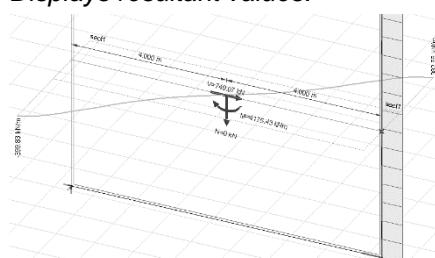
sdmDiagramRes = 0x02

Displays diagram and resultant value with position.



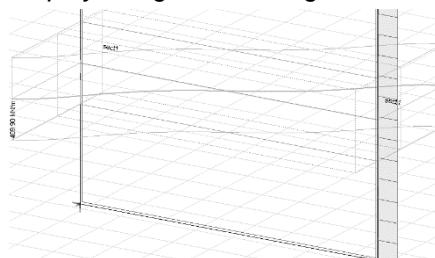
sdmResultant = 0x03

Displays resultant values.



sdmDiagramSegWidth = 0x04 }

Displays diagram with segmented width



enum	EResultType = {	
	rtLoadCase = 0x00	<i>results of a load case</i>
	rtLoadCombination = 0x01	<i>results of a load combination</i>
	rtEnvelope = 0x02	<i>results of a specific envelope</i>
	rtCritical = 0x03 }	<i>critical results</i>
enum	ESectionSegmentChainIntegratedResultant = {	
	sscir_N = 0x00	<i>normal force [kN]</i>
	sscir_Q = 0x01	<i>shear force [kN]</i>
	sscir_M = 0x02 }	<i>bending moment [kNm]</i>
enum	EResultComponent = {	
	<u>ELineStress:</u>	
	rc_IsSmin = 0	<i>[Smin] cross-section minimum of the axial stress [kN/m²]</i>
	rc_IsSmax = 1	<i>[Smax] cross-section maximum of the axial stress [kN/m²]</i>
	rc_IsVmin = 2	<i>[Vmin] cross-section minimum of shear stress [kN/m²]</i>
	rc_IsVmax = 3	<i>[Vmax] cross-section maximum of shear stress [kN/m²]</i>
	rc_IsSomin = 4	<i>[Somin] cross-section minimum of VonMises stress [kN/m²]</i>
	rc_IsSomax = 5	<i>[Somax] cross-section maximum of VonMises stress [kN/m²]</i>
	rc_IsVymean = 6	<i>[Vy] mean shear stress in local y direction [kN/m²]</i>
	rc_IsVzmean = 7	<i>[Vz] mean shear stress in local z direction [kN/m²]</i>
	rc_IsSminMax = 8	<i>[Sminmax] cross-section minimum and maximum of the axial stress [kN/m²]</i>
	rc_IsVminMax = 9	<i>[Vminmax] cross-section minimum and maximum of the shear stress, [kN/m²]</i>
	rc_IsSominMax = 10	<i>[Sominmax] cross-section minimum and maximum of the VonMises stress [kN/m²]</i>
	rc_IsSeffMin = 11	<i>[(NL) Seff Min] cross-section minimum of the effective stress (for nonlinear materials) [kN/m²]</i>
	rc_IsSeffMax = 12	<i>[(NL) Seff Max] cross-section maximum of the effective stress (for nonlinear materials) [kN/m²]</i>
	rc_IsSeffMinMax = 13	<i>[(NL) Seff MinMax] cross-section minimum and maximum of the effective stress (for nonlinear materials) [kN/m²]</i>
	rc_IsfyMin = 14	<i>[(NL) fymin] cross-section minimum of the actual yield strength (for nonlinear materials) [kN/m²]</i>
	rc_IsfyMax = 15	<i>[(NL) fymax] cross-section maximum of the actual yield strength (for nonlinear materials) [kN/m²]</i>
	rc_IsfyMinMax = 16	<i>[(NL) fyminmax] cross-section minimum and maximum of the actual yield strength (for nonlinear materials) [kN/m²]</i>
	rc_IsUMin = 17	<i>[(NL) Utilization min] cross-section minimum of the utilization (for nonlinear materials) []</i>
	rc_IsUMax = 18	<i>[(NL) Utilization max] cross-section maximum of the utilization (for nonlinear materials) []</i>
	rc_IsUMinMax = 19	<i>[(NL) Utilization minmax] cross-section minimum and maximum of the utilization (for nonlinear materials) []</i>
	rc_IsBMin = 20	<i>[(NL) B min] cross-section minimum of the back stress (for nonlinear materials) [kN/m²]</i>
	rc_IsBMax = 21	<i>[(NL) B max] cross-section maximum of the back stress (for nonlinear materials) [kN/m²]</i>
	rc_IsBMax = 22	<i>[(NL) B minmax] cross-section minimum and maximum of the back stress (for nonlinear materials) [kN/m²]</i>
	<u>ESurfaceStress:</u>	
	rc_ssSxx_Top = 100	<i>[Sxx] axial stress in local x direction at the top [kN/m²]</i>
	rc_ssSyy_Top = 101	<i>[Syy] axial stress in local y direction at the top [kN/m²]</i>
	rc_ssSxy_Top = 102	<i>[Sxy] torsional/shear stress at the top [kN/m²]</i>

rc_ssSxz_Top = 103	[Sxz] torsional/shear stress at the top [kN/m ²]
rc_ssSyz_Top = 104	[Syz] torsional/shear stress at the top [kN/m ²]
rc_ssSVM_Top = 105	[SVM] VonMises stress at the top [kN/m ²]
rc_ssS1_Top = 106	[S1] 1st principal stress at the top [kN/m ²]
rc_ssS2_Top = 107	[S2] 2st principal stress at the top [kN/m ²]
rc_ssAs_Top = 108	[aS] principal direction angle the at top [°]
rc_ssSxx_Middle = 110	[Sxx] axial stress in local x direction in the middle [kN/m ²]
rc_ssSyy_Middle = 111	[Syy] axial stress in local y direction in the middle [kN/m ²]
rc_ssSxy_Middle = 112	[Sxy] torsional/shear stress in the middle [kN/m ²]
rc_ssSxz_Middle = 113	[Sxz] torsional/shear stress in the middle [kN/m ²]
rc_ssSyz_Middle = 114	[Syz] torsional/shear stress in the middle [kN/m ²]
rc_ssSVM_Middle = 115	[SVM] VonMises stress in the middle [kN/m ²]
rc_ssS1_Middle = 116	[S1] 1st principal stress in the middle [kN/m ²]
rc_ssS2_Middle = 117	[S2] 2st principal stress in the middle [kN/m ²]
rc_ssAs_Middle = 118	[aS] principal direction angle in the middle [°]
rc_ssSxx_Bottom = 120	[Sxx] axial stress in local x direction at the bottom [kN/m ²]
rc_ssSyy_Bottom = 121	[Syy] axial stress in local y direction at the bottom [kN/m ²]
rc_ssSxy_Bottom = 122	[Sxy] torsional/shear stress at the bottom [kN/m ²]
rc_ssSxz_Bottom = 123	[Sxz] torsional/shear stress at the bottom [kN/m ²]
rc_ssSyz_Bottom = 124	[Syz] torsional/shear stress at the bottom [kN/m ²]
rc_ssSVM_Bottom = 125	[SVM] VonMises stress at the bottom [kN/m ²]
rc_ssS1_Bottom = 126	[S1] 1st principal stress at the bottom [kN/m ²]
rc_ssS2_Bottom = 127	[S2] 2st principal stress at the bottom [kN/m ²]
rc_ssAs_Bottom = 128	[aS] principal direction angle at the bottom [°]
rc_ssSzz_Top = 129	[Szz] axial stress in local z direction at the top [kN/m ²]
rc_ssSeff_Top = 130	[NL Seff] effective stress (for nonlinear materials) at the top [kN/m ²]
rc_ssfy_Top = 131	[NL fy] actual yield strength (for nonlinear materials) at the top [kN/m ²]
rc_ssU_Top = 132	[NL Utilization] utilization (for nonlinear materials) at the top []
rc_ssState_Top = 133	[NL State] stress state (for nonlinear materials) at the top []
rc_ssByy_Top = 134	[NL Byy] back stress in local y direction (for nonlinear materials) at the top [kN/m ²]
rc_ssBzz_Top = 135	[NL Bzz] back stress in local z direction (for nonlinear materials) at the top [kN/m ²]
rc_ssBxy_Top = 136	[NL Bxy] shear back stress (for nonlinear materials) at the top [kN/m ²]
rc_ssSzz_Middle = 137	[Szz] axial stress in local z direction at the middle [kN/m ²]
rc_ssSeff_Middle = 138	[NL Seff] effective stress (for nonlinear materials) at the middle [kN/m ²]
rc_ssfy_Middle = 139	[NL fy] actual yield strength (for nonlinear materials) at the middle [kN/m ²]
rc_ssU_Middle = 140	[NL Utilization] utilization (for nonlinear materials) at the middle []
rc_ssState_Middle = 141	[NL State] stress state (for nonlinear materials) at the middle []
rc_ssByy_Middle = 142	[NL Byy] back stress in local y direction (for nonlinear materials) at the middle [kN/m ²]
rc_ssBzz_Middle = 143	[NL Bzz] back stress in local z direction (for nonlinear materials) at the middle [kN/m ²]
rc_ssBxy_Middle = 144	[NL Bxy] shear back stress (for nonlinear materials) at the middle [kN/m ²]
rc_ssSzz_Bottom = 145	[Szz] axial stress in local z direction at the bottom [kN/m ²]
rc_ssSeff_Bottom = 146	[NL Seff] effective stress (for nonlinear materials) at the bottom [kN/m ²]
rc_ssfy_Bottom = 147	[NL fy] actual yield strength (for nonlinear materials) at the bottom [kN/m ²]

rc_ssU_Bottom = 148	<i>[(NL) Utilization] utilization (for nonlinear materials) at the bottom []</i>
rc_ssState_Bottom = 149	<i>[(NL) State] stress state (for nonlinear materials) at the bottom []</i>
rc_ssByy_Bottom = 150	<i>[(NL) Byy] back stress in local y direction (for nonlinear materials) at the bottom [kN/m²] </i>
rc_ssBzz_Bottom = 151	<i>[(NL) Bzz] back stress in local z direction (for nonlinear materials) at the bottom [kN/m²] </i>
rc_ssBxy_Bottom = 152	<i>[(NL) Bxy] shear back stress (for nonlinear materials) at the bottom [kN/m²] </i>

ELineForce:

rc_ifNx = 200	<i>[Nx] normal force [kN]</i>
rc_ifVy = 201	<i>[Vy] shear force [kN]</i>
rc_ifVz = 202	<i>[Vz] shear force [kN]</i>
rc_ifTx = 203	<i>[Tx] twisting moment [kNm]</i>
rc_ifMy = 204	<i>[My] bending moment [kNm]</i>
rc_ifMz = 205	<i>[Mz] bending moment [kNm]</i>
rc_ifMyD = 206	<i>[MyD] design flexural moment about local y axis [kNm]</i>
rc_ifVxz = 207	<i>[Vxz] longitudinal shear connection force [kN/m]</i>

ESurfaceForce:

rc_sfNx = 300	<i>[nx] cross-section force [kN/m]</i>
rc_sfNy = 301	<i>[ny] cross-section force [kN/m]</i>
rc_sfNxy = 302	<i>[nxy] cross-section torsional force [kN/m]</i>
rc_sfMx = 303	<i>[mx] bending moment [kNm/m]</i>
rc_sfMy = 304	<i>[my] bending moment [kNm/m]</i>
rc_sfMxy = 305	<i>[mxy] torsional moment [kNm/m]</i>
rc_sfVxz = 306	<i>[vxz] shear force [kN/m]</i>
rc_sfVyz = 307	<i>[vyz] shear force [kN/m]</i>
rc_sfvSz = 308	<i>[vRz] resultant shear force [kN/m]</i>
rc_sfN1 = 309	<i>[n1] 1st principal force [kN/m]</i>
rc_sfN2 = 310	<i>[n2] 2nd principal force [kN/m]</i>
rc_sfAn = 311	<i>[an] principal force direction [°]</i>
rc_sfM1 = 312	<i>[m1] 1st principal moment [kNm/m]</i>
rc_sfM2 = 313	<i>[m2] 2nd principal moment [kNm/m]</i>
rc_sfAm = 314	<i>[am] principal moment direction [°]</i>
rc_sfNxD = 315	<i>[nxD] design force [kN/m]</i>
rc_sfNyD = 316	<i>[nyD] design force [kN/m]</i>
rc_sfMxDp = 319	<i>[mxD+] design moment (plus) [kNm/m]</i>
rc_sfMxDm = 320	<i>[mxD-] design moment (minus) [kNm/m]</i>
rc_sfMyDp = 321	<i>[myD+] design moment (plus) [kNm/m]</i>
rc_sfMyDm = 322	<i>[myD-] design moment (minus) [kNm/m]</i>
rc_sfAvRz = 323	<i>[avRz] direction of the resultant shear force [°]</i>
rc_sfAn1 = 324	<i>[an1] direction of principal force 1 [°]</i>
rc_sfAn2 = 325	<i>[an2] direction of principal force 2 [°]</i>
rc_sfAm1 = 326	<i>[am1] direction of principal moment 1 [°]</i>
rc_sfAm2 = 327	<i>[am2] direction of principal moment 2 [°]</i>

ENodalSupportForce:

rc_nsfRx = 400	<i>[Rx] support force in local x direction [kN]</i>
rc_nsfRy = 401	<i>[Ry] support force in local y direction [kN]</i>
rc_nsfRz = 402	<i>[Rz] support force in local z direction [kN]</i>
rc_nsfRxx = 403	<i>[Rxx] support moment about local x axis [kNm]</i>
rc_nsfRyy = 404	<i>[Ryy] support moment about local y axis [kNm]</i>
rc_nsfRzz = 405	<i>[Rzz] support moment about local z axis [kNm]</i>
rc_nsfRr = 406	<i>[Rr] resultant support force [kN]</i>

rc_nsfRrr = 407	<i>[Rrr] resultant support moment [kNm]</i>
rc_nsfRxyz = 408	<i>[Rxyz] support forces in local directions [kN]</i>
rc_nsfRxxyyzz = 409	<i>[Rxxyyzz] support forces about local directions [kN]</i>
rc_nsfRalpha = 410	<i>[AR] ratio of support force component in loc. xy plane to support force in loc. z direction [-]</i>

[ELineSupportForce:](#)

rc_lsfRx = 500	<i>[Rx] support force in local x direction [kN/m]</i>
rc_lsfRy = 501	<i>[Ry] support force in local y direction [kN/m]</i>
rc_lsfRz = 502	<i>[Rz] support force in local z direction [kN/m]</i>
rc_lsfRxx = 503	<i>[Rxx] support moment about local x axis [kNm/m]</i>
rc_lsfRyy = 504	<i>[Ryy] support moment about local y axis [kNm/m]</i>
rc_lsfRzz = 505	<i>[Rzz] support moment about local z axis [kNm/m]</i>
rc_lsfRr = 506	<i>[Rr] resultant support force [kN/m]</i>
rc_lsfRrr = 507	<i>[Rrr] resultant support moment [kNm/m]</i>

ESurfaceSupportForce:

rc_ssfx = 600	[Rx] support force in local x direction [kN/m ²]
rc_ssfy = 601	[Ry] support force in local y direction [kN/m ²]
rc_ssfrz = 602	[Rz] support force in local z direction [kN/m ²]

ESpringForce:

rc_sfRx = 700	[Rx] spring force in local x direction [kN]
rc_sfRy = 701	[Ry] spring force in local y direction [kN]
rc_sfRz = 702	[Rz] spring force in local z direction [kN]
rc_sfRxx = 704	[Rxx] spring moment about local x axis [kNm]
rc_sfRyy = 705	[Ryy] spring moment about local y axis [kNm]
rc_sfRzz = 706	[Rzz] spring moment about local z axis [kNm]

gap force:

rc_gfx = 800	[Nx] force in local x direction [kN]
---------------------	--------------------------------------

EEdgeConnectionForce:

Warning! line-line link forces use the same constants as edge connections

rc_ecfnx = 900	[nx] normal force [kN/m]
rc_ecfvy = 901	[ny] shear force [kN/m]
rc_ecfvz = 902	[nz] shear force [kN/m]
rc_ecftx = 903	[mx] twisting moment [kNm/m]
rc_ecfmy = 904	[my] bending moment [kNm/m]
rc_ecfmz = 905	[mz] bending moment [kNm/m]

ELinkElementForce:

Warning! line-line link forces use the same constants as edge connections

rc_lefnx_nn = 1000	[R(x)] normal force for node-node link [kN]
rc_lefvy_nn = 1001	[R(y)] shear force for node-node link [kN]
rc_lefvz_nn = 1002	[R(z)] shear force for node-node link [kN]
rc_leftx_nn = 1003	[R(xx)] twisting moment for node-node link [kNm]
rc_lefmy_nn = 1004	[R(yy)] bending moment for node-node link [kNm]
rc_lefmz_nn = 1005	[Rzz] bending moment for node-node link [kNm]
rc_lefnx_ll = 1010	use rc_ecfnx instead
rc_lefvy_ll = 1011	use rc_ecfvy instead
rc_lefvz_ll = 1012	use rc_ecfvz instead
rc_leftx_ll = 1013	use rc_ecftx instead
rc_lefmy_ll = 1014	use rc_ecfmy instead
rc_lefmz_ll = 1015	use rc_ecfmz instead

EDisplacement:

displacements on the static tab

<i>nodal displacements:</i>		<i>line displacements:</i>
rc_d_ex = 1101	[ex] displacement in global X direction [m]	displacement in local x direction
rc_d_ey = 1102	[ey] displacement in global Y direction [m]	displacement in local y direction
rc_d_ez = 1103	[ez] displacement in global Z direction [m]	displacement in local z direction
rc_d_fx = 1104	[fx] rotation about the global X axis [rad]	rotation about the local x axis
rc_d_fy = 1105	[fy] rotation about the global Y axis [rad]	rotation about the local y axis
rc_d fz = 1106	[fz] rotation about the global Z axis [rad]	rotation about the local z axis
rc_d_er = 1107	[eR] resultant displacement [m]	
rc_d_fr = 1108	[fR] resultant rotation [rad]	

EReinforcement:

rc_rAsbx = 1200
rc_rAsby = 1201
rc_rAstx = 1202
rc_rAsty = 1203
rc_rAsxbt = 1204
rc_rAsybt = 1205
rc_rAsxyb = 1206
rc_rAsxyt = 1207

[*axb*] Bottom reinforcement in local x direction [m^2/m]
 [*ayb*] Bottom reinforcement in y direction [m^2/m]
 [*axt*] Top reinforcement in x direction [m^2/m]
 [*ayt*] Top reinforcement in y direction [m^2/m]
 [*axb,axt*] Reinforcement in x direction at top and bottom [m^2/m]
 [*ayb,ayt*] Reinforcement in y direction at top and bottom [m^2/m]
 [*axb,ayb*] Reinforcement in x and y direction at bottom [m^2/m]
 [*axt,ayt*] Reinforcement in x and y direction at top [m^2/m]

EShearCapacity:

rc_scVRdc = 1300
rc_scVEdMinusVRdc = 1301
rc_scVRdmax = 1302
rc_scVEdDivVRdmax = 1303
rc_scaVED = 1304
rc_scAsw = 1305

[*VRd,c*] shear resistance without shear reinforcement [kN/m]
 [*(vEd-VRd,c)*] resultant shear force minus shear resistance [kN/m]
 [*VRd,max*] maximum shear resistance without shear reinforcement [kN/m]
 [*(vEd/VRd,max)*] resultant shear force divided by maximum shear resistance []
 [*avEd*] principal direction for shear [$^\circ$]
 [*asw*] required specific shear reinforcement [m^2/m]

ECrackWidth:

rc_cw_wkb = 1400
rc_cw_wkt = 1401
rc_cw_wk2b = 1402
rc_cw_wk2t = 1403
rc_cw_wRb = 1404
rc_cw_wRt = 1405
rc_cw_wS2b = 1406
rc_cw_wS2t = 1407

[*wk(b)*] bottom cracking at reinforcement bar [m]
 [*wk(t)*] top cracking at reinforcement bar [m]
 [*wk2(b)*] bottom cracking at extreme fibre [m]
 [*wk2(t)*] top cracking at extreme fibre [m]
 [*wR(b)*] angle of primary crack relative to the local x direction at the bottom [$^\circ$]
 [*wR(t)*] angle of primary crack relative to the local x direction at the top [$^\circ$]
 [*S2b*] stress in the bottom rebar [kN/m^2]
 [*S2t*] stress in the top rebar [kN/m^2]

EVelocity:

rc_veX = 1500
rc_veY = 1501
rc_veZ = 1502
rc_veXX = 1503
rc_veYY = 1504
rc_veZZ = 1505
rc_ver = 1506
rc_veRR = 1507

[*vX*] velocity in x direction [m/s]
 [*vY*] velocity in y direction [m/s]
 [*vZ*] velocity in z direction [m/s]
 [*vXX*] angular velocity about the x axis [rad/s]
 [*vYY*] angular velocity about the y axis [rad/s]
 [*vZZ*] angular velocity about the z axis [rad/s]
 [*vR*] resultant velocity [m/s]
 [*vRR*] resultant angular velocity [rad/s]

EAcceleration:

rc_acX = 1600
rc_acY = 1601
rc_acZ = 1602
rc_acXX = 1603
rc_acYY = 1604
rc_acZZ = 1605
rc_acR = 1606
rc_acRR = 1607

[*aX*] acceleration in x direction [m/s^2]
 [*aY*] acceleration in y direction [m/s^2]
 [*aZ*] acceleration in z direction [m/s^2]
 [*aXX*] angular acceleration about the x axis [rad/s^2]
 [*aYY*] angular acceleration about the y axis [rad/s^2]
 [*aZZ*] angular acceleration about the z axis [rad/s^2]
 [*aR*] resultant acceleration [m/s^2]
 [*aRR*] resultant angular acceleration [rad/s^2]

Actual reinforcement

rc_arAsxb = 1700
rc_arAsyb = 1701
rc_arAsxt = 1702
rc_arAsyt = 1703
rc_arAsxbt = 1704
rc_arAsybt = 1705

[*xb*] actual reinforcement in x direction at bottom [m^2/m]
 [*yb*] actual reinforcement in y direction at bottom [m^2/m]
 [*xt*] actual reinforcement in x direction at top [m^2/m]
 [*yt*] actual reinforcement in y direction at top [m^2/m]
 [*xb,xt*] actual reinforcement in x direction at top and bottom [m^2/m]
 [*yb,yt*] actual reinforcement in y direction at top and bottom [m^2/m]

Reinforcement difference

rc_rdAsxb = 1800	[$xb - axb$] reinforcement difference (calculated-actual) in x direction at bottom [m^2/m]
rc_rdAsyb = 1801	[$yb - ayb$] reinforcement difference (calculated-actual) in y direction at bottom [m^2/m]
rc_rdAsxt = 1802	[$xt - axt$] reinforcement difference (calculated-actual) in x direction at top [m^2/m]
rc_rdAsyt = 1803	[$yt - ayt$] reinforcement difference (calculated-actual) in y direction at top [m^2/m]

Influence lines

rc_ilPx1 = 1900	[$PX'1$] influence line ordinate x []
rc_ilPy1 = 1901	[$PY'1$] influence line ordinate y []
rc_ilPz1 = 1902	[$PZ'1$] influence line ordinate z []

Surface strain

rc_sstExx = 2000,	[exx] strain exx []
rc_sstEyy = 2001,	[eyy] strain eyy []
rc_sstExy = 2002,	[exy] strain exy []
rc_sstFzz = 2003,	[fzz] rotation fzz []
rc_sstKxx = 2004,	[kxx] curvature kxx [1/m]
rc_sstKyy = 2005,	[kyy] curvature kyy [1/m]
rc_sstKxy = 2006,	[kxy] curvature kxy [1/m]
rc_sstExz = 2007,	[exz] strain exz []
rc_sstEyz = 2008,	[eyz] strain eyz []
rc_sstEsz = 2009,	[eSz] strain exxyzR []
rc_sstE1 = 2010,	[$e1$] principal strain 1 []
rc_sstE2 = 2011,	[$e2$] principal strain 2 []
rc_sstAe = 2012,	[ae] principal strain direction [°]
rc_sstK1 = 2013,	[$k1$] principal curvature 1 [1/m]
rc_sstK2 = 2014,	[$k2$] principal curvature 2 [1/m]
rc_sstAk = 2015,	[ak] principal curvature direction [°]

Intensity variation

rc_ivDnx = 2100,	[dnx] dnx [%]
rc_ivDny = 2101,	[dny] dny [%]
rc_ivDnxy = 2102,	[$dnxy$] dnxy [%]
rc_ivDmx = 2103,	[dmx] dmx [%]
rc_ivDmy = 2104,	[dmy] dmy [%]
rc_ivDmxy = 2105,	[$dmxy$] dmxy [%]
rc_ivDqx = 2106,	[dqx] dqx [%]
rc_ivDqy = 2107,	[dqy] dqy [%]

Displacements on the buckling tab

rc_bd_eX = 2200,	[ex] displacement in global X direction [m]
rc_bd_eY = 2201,	[ey] displacement in global Y direction [m]
rc_bd_eZ = 2202,	[ez] displacement in global Z direction [m]
rc_bd_fx = 2203,	[fx] rotation about the global X axis [rad]
rc_bd_fY = 2204,	[fy] rotation about the global Y axis [rad]
rc_bd_fZ = 2205,	[fz] rotation about the global Z axis [rad]
rc_bd_eR = 2206,	[eR] resultant displacement [m]
rc_bd_fR = 2207,	[fR] resultant rotation [rad]

Rebar spacing

rc_rsSxb = 2300,	[sxb] local x direction bottom reinforcement spacing [m]
rc_rsSyb = 2301,	[syb] local y direction bottom reinforcement spacing [m]
rc_rsSxt = 2302,	[sxt] local x direction top reinforcement spacing [m]
rc_rsSyt = 2303,	[syt] local y direction top reinforcement spacing [m]
rc_rsSxbt = 2304,	[sxb,sxt] local x direction bottom and top reinforcement spacing [m]
rc_rsSybt = 2305,	[syb,syt] local y direction bottom and top reinforcement spacing [m]

Reinforcement check

rc_rReinfCheck = 2400,	<i>[reinforcement check] reinforcement check []</i>
Young modulus	
rc_ymEx = 2500,	<i>[Ex] calculated modulus of elasticity in local x direction [kN/m²]</i>
rc_ymEy = 2501,	<i>[Ey] calculated modulus of elasticity in local y direction [kN/m²]</i>
Mode shape ordinates	
rc_vd_eX = 2600,	<i>[eX] translation in X direction []</i>
rc_vd_eY = 2601,	<i>[eY] translation in Y direction []</i>
rc_vd_eZ = 2602,	<i>[eZ] translation in Z direction []</i>
rc_vd_fX = 2603,	<i>[fX] rotation in X direction []</i>
rc_vd_fY = 2604,	<i>[fY] rotation in Y direction []</i>
rc_vd_fZ = 2605,	<i>[fZ] rotation in Z direction []</i>
rc_vd_eR = 2606,	<i>[eR] resultant translation []</i>
rc_vd_fR = 2607,	<i>[fR] resultant rotation []</i>
Reinforcement design forces ULS	
rc_sfMxD = 2700,	<i>[mxD] mx design reinforcement moment (ULS) [kNm/m]</i>
rc_sfMyD = 2701,	<i>[myD] my design reinforcement moment (ULS) [kNm/m]</i>
rc_sfMxU = 2702,	<i>[mxU] mx ultimate reinforcement moment (ULS) [kNm/m]</i>
rc_sfMyU = 2703,	<i>[myU] my ultimate reinforcement moment (ULS) [kNm/m]</i>
rc_sfMxDU = 2704,	<i>[mxD,mxU] mx design and ultimate reinforcement moment (ULS) [kNm/m]</i>
rc_sfMyDU = 2705,	<i>[myD,myU] my design and ultimate reinforcement moment (ULS) [kNm/m]</i>
Reinforcement design forces SLS	
rc_sfMxDcr = 2800,	<i>[mxD(cr)] mx design reinforcement moment (SLS) [kNm/m]</i>
rc_sfMyDcr = 2801,	<i>[myD(cr)] my design reinforcement moment (SLS) [kNm/m]</i>
rc_sfMxUcr = 2802,	<i>[mxU(cr)] mx ultimate reinforcement moment (SLS) [kNm/m]</i>
rc_sfMyUcr = 2803,	<i>[myU(cr)] my ultimate reinforcement moment (SLS) [kNm/m]</i>
rc_sfMxDUcr = 2804,	<i>[mxD(cr),mxU(cr)] mx design and ultimate reinforcement moment (SLS) [kNm/m]</i>
rc_sfMyDUcr = 2805,	<i>[myD(cr),myU(cr)] my design and ultimate reinforcement moment (SLS) [kNm/m]</i>
Arbo/cret utilization	
rc_arcrUtil = 2901,	<i>[Utilization] utilization []</i>
rc_arcrAvgUtil = 2902,	<i>[Average utilization] average utilization []</i>
Arbo/cret results	
rc_arcrVx = 3000,	<i>[vx] shear force in local x direction [kN/m]</i>
rc_arcrN = 3001,	<i>[n] axial force [kN/m]</i>
rc_arcrVz = 3002,	<i>[vz] shear force in local z direction [kN/m]</i>
rc_arcrM = 3003,	<i>[m] torsional moment [kNm/m]</i>
rc_arcrDez = 3004,	<i>[CRET dez] relative displacement in local z direction [m]</i>
Beam end release relative displacements	
rc_berrdEx = 3100,	<i>[ex] beam end release relative displacement in x direction [m]</i>
rc_berrdEy = 3101,	<i>[ey] beam end release relative displacement in y direction [m]</i>
rc_berrdEz = 3102,	<i>[ez] beam end release relative displacement in z direction [m]</i>
rc_berrdFx = 3103,	<i>[fx] beam end release relative rotation about local x axis [rad]</i>
rc_berrdFy = 3104,	<i>[fy] beam end release relative rotation about local y axis [rad]</i>
rc_berrdFz = 3105,	<i>[fz] beam end release relative rotation about local z axis [rad]</i>
rc_berrdEr = 3106,	<i>[eR] beam end release resultant relative translation [m]</i>
rc_berrdFr = 3107,	<i>[fR] beam end release resultant relative rotation [rad]</i>
Beam strain	
rc_bstExx = 3200,	<i>[exx] strain exx []</i>
rc_bstKyy = 3201,	<i>[kyy] curvature kyy [1/m]</i>
rc_bstKzz = 3202,	<i>[kzz] curvature kzz [1/m]</i>
rc_bstEyz = 3203,	<i>[eyz] strain eyz []</i>
rc_bstExy = 3204,	<i>[exy] strain exy []</i>

rc_bstExz = 3205,	[exz] strain exz []
Virtual beam internal forces	
rc_vbifNx = 3300, rc_vbifVy = 3301, rc_vbifVz = 3302, rc_vbifTx = 3303, rc_vbifMy = 3304, rc_vbifMz = 3305,	[Nx] axial force [kN] [Vy] shear force in local y direction [kN] [Vz] shear force in local z direction [kN] [Tx] torsional moment [kNm] [My] flexural moment about local y axis [kNm] [Mz] flexural moment about local z axis [kNm]
Beam strain at stress point	
rc_bstspExxTMin = 3400, rc_bstspExxTMax = 3401, rc_bstspExxTMinMax = 3402, rc_bstspExxEMin = 3403, rc_bstspExxEMax = 3404, rc_bstspExxEMinMax = 3405, rc_bstspExxPMin = 3406, rc_bstspExxPMax = 3407, rc_bstspExxPMinMax = 3408, rc_bstspEeffMin = 3409, rc_bstspEeffMax = 3410, rc_bstspEeffMinMax = 3411, rc_bstspDeeffMin = 3412, rc_bstspDeeffMax = 3413, rc_bstspDeeffMinMax = 3414, rc_bstspLeeffMin = 3415, rc_bstspLeeffMax = 3416, rc_bstspLeeffMinMax = 3417,	[exx T Min] minimum stress point strain [] [exx T Max] maximum stress point strain [] [exx T MinMax] minimum,maximum stress point strain [] [(NLP) exx E Min] minimum elastic stress point strain [] [(NLP) exx E Max] maximum elastic stress point strain [] [(NLP) exx E MinMax] minimum, maximum elastic stress point strain [] [(NLP) exx P Min] minimum plastic stress point strain [] [(NLP) exx P Max] maximum plastic stress point strain [] [(NLP) exx P MinMax] minimum, maximum plastic stress point strain [] [(NLP) eeff Min] minimum effective stress point strain [] [(NLP) eeff Max] maximum effective stress point strain [] [(NLP) eeff MinMax] minimum, maximum effective stress point strain [] [(NLP) deeff Min] minimum effective plastic stress point strain increment [] [(NLP) deeff Max] maximum effective plastic stress point strain increment [] [(NLP) deeff MinMax] minimum, maximum effective plastic stress point strain increment [] [(NLE) eeff Min] minimum effective stress point strain [] [(NLE) eeff Max] maximum effective stress point strain [] [(NLE) eeff MinMax] minimum, maximum effective stress point strain []
Vibration response factor analysis	
rc_vrfARSelf = 3500, rc_vrfARestr = 3501, rc_vrfARFull = 3502,	[R self] vibration response factor (self) [] [R extr.] vibration response factor (extr.) [] [R full] vibration response factor (full) []
Virtual beam displacements	
rc_vbdEx = 3600, rc_vbdEy = 3601, rc_vbdEz = 3602, rc_vbdFx = 3603, rc_vbdFy = 3604, rc_vbdFz = 3605, rc_vbdEr = 3606, rc_vbdFr = 3607,	[eX] translation in x direction [m] [eY] translation in y direction [m] [eZ] translation in z direction [m] [fX] rotation in x direction [rad] [fY] rotation in y direction [rad] [fZ] rotation in z direction [rad] [erR] resultant translation [m] [frR] resultant rotation [rad]
Surface strain at stress point	
rc_sstspExxTT = 3700, rc_sstspEyyTT = 3701, rc_sstspExyTT = 3702, rc_sstspE1TT = 3703, rc_sstspE2TT = 3704, rc_sstspAeTT = 3705, rc_sstspExxET = 3706, rc_sstspEyyET = 3707, rc_sstspExyET = 3708,	[exx T T] strain exx T at the top [] [eyy T T] strain eyy T at the top [] [exy T T] strain exy T at the top [] [e1 T T] principal strain 1 at the top [] [e2 T T] principal strain 2 at the top [] [ae T T] principal strain direction at the top [°] [(NLP) exx E T] elastic strain exx at the top [] [(NLP) eyy E T] elastic strain eyy at the top [] [(NLP) exy E T] elastic strain exy at the top []

<code>rc_sstspE1ET</code> = 3709,	$[(NLP) e1 E T]$ elastic principal strain 1 at the top []
<code>rc_sstspE2ET</code> = 3710,	$[(NLP) e2 E T]$ elastic principal strain 2 at the top []
<code>rc_sstspAeET</code> = 3711,	$[(NLP) ae E T]$ elastic principal strain direction at the top [°]
<code>rc_sstspExxPT</code> = 3712,	$[(NLP) exx P T]$ plastic strain exx at the top []
<code>rc_sstspEyyPT</code> = 3713,	$[(NLP) eyy P T]$ plastic strain eyy at the top []
<code>rc_sstspExyPT</code> = 3714,	$[(NLP) exy P T]$ plastic strain exy at the top []
<code>rc_sstspE1PT</code> = 3715,	$[(NLP) e1 P T]$ plastic principal strain 1 at the top []
<code>rc_sstspE2PT</code> = 3716,	$[(NLP) e2 P T]$ plastic principal strain 2 at the top []
<code>rc_sstspAePT</code> = 3717,	$[(NLP) ae P T]$ plastic principal strain direction at the top [°]
<code>rc_sstspEeffPT</code> = 3718,	$[(NLP) eeff P T]$ effective plastic strain at the top []
<code>rc_sstspDeeffPT</code> = 3719,	$[(NLP) deeff P T]$ effective plastic strain increment at the top []
<code>rc_sstspExxTC</code> = 3720,	$[exx T C]$ strain exx T at the center []
<code>rc_sstspEyyTC</code> = 3721,	$[eyy T C]$ strain eyy T at the center []
<code>rc_sstspExyTC</code> = 3722,	$[exy T C]$ strain exy T at the center []
<code>rc_sstspE1TC</code> = 3723,	$[e1 T C]$ principal strain 1 at the center []
<code>rc_sstspE2TC</code> = 3724,	$[e2 T C]$ principal strain 2 at the center []
<code>rc_sstspAeTC</code> = 3725,	$[ae T C]$ principal strain direction at the center [°]
<code>rc_sstspExxEc</code> = 3726,	$[(NLP) exx E C]$ elastic strain exx at the center []
<code>rc_sstspEyyEc</code> = 3727,	$[(NLP) eyy E C]$ elastic strain eyy at the center []
<code>rc_sstspExyEc</code> = 3728,	$[(NLP) exy E C]$ elastic strain exy at the center []
<code>rc_sstspE1Ec</code> = 3729,	$[(NLP) e1 E C]$ elastic principal strain 1 at the center []
<code>rc_sstspE2Ec</code> = 3730,	$[(NLP) e2 E C]$ elastic principal strain 2 at the center []
<code>rc_sstspAeEc</code> = 3731,	$[(NLP) ae E C]$ elastic principal strain direction at the center [°]
<code>rc_sstspExxPc</code> = 3732,	$[(NLP) exx P C]$ plastic strain exx at the center []
<code>rc_sstspEyyPc</code> = 3733,	$[(NLP) eyy P C]$ plastic strain eyy at the center []
<code>rc_sstspExyPc</code> = 3734,	$[(NLP) exy P C]$ plastic strain exy at the center []
<code>rc_sstspE1Pc</code> = 3735,	$[(NLP) e1 P C]$ plastic principal strain 1 at the center []
<code>rc_sstspE2Pc</code> = 3736,	$[(NLP) e2 P C]$ plastic principal strain 2 at the center []
<code>rc_sstspAePc</code> = 3737,	$[(NLP) ae P C]$ plastic principal strain direction at the center [°]
<code>rc_sstspEeffPc</code> = 3738,	$[(NLP) eeff P C]$ effective plastic strain at the center []
<code>rc_sstspDeeffPc</code> = 3739,	$[(NLP) deeff P C]$ effective plastic strain increment at the center []
<code>rc_sstspExxB</code> = 3740,	$[exx T B]$ strain exx T at the bottom []
<code>rc_sstspEyyB</code> = 3741,	$[eyy T B]$ strain eyy T at the bottom []
<code>rc_sstspExyB</code> = 3742,	$[exy T B]$ strain exy T at the bottom []
<code>rc_sstspE1TB</code> = 3743,	$[e1 T B]$ principal strain 1 at the bottom []
<code>rc_sstspE2TB</code> = 3744,	$[e2 T B]$ principal strain 2 at the bottom []
<code>rc_sstspAeTB</code> = 3745,	$[ae T B]$ principal strain direction at the bottom [°]
<code>rc_sstspExxEb</code> = 3746,	$[(NLP) exx E B]$ elastic strain exx at the bottom []
<code>rc_sstspEyyEb</code> = 3747,	$[(NLP) eyy E B]$ elastic strain eyy at the bottom []
<code>rc_sstspExyEb</code> = 3748,	$[(NLP) exy E B]$ elastic strain exy at the bottom []
<code>rc_sstspE1Eb</code> = 3749,	$[(NLP) e1 E B]$ elastic principal strain 1 at the bottom []
<code>rc_sstspE2Eb</code> = 3750,	$[(NLP) e2 E B]$ elastic principal strain 2 at the bottom []
<code>rc_sstspAeEb</code> = 3751,	$[(NLP) ae E B]$ elastic principal strain direction at the bottom [°]
<code>rc_sstspExxPb</code> = 3752,	$[(NLP) exx P B]$ plastic strain exx at the bottom []
<code>rc_sstspEyyPb</code> = 3753,	$[(NLP) eyy P B]$ plastic strain eyy at the bottom []
<code>rc_sstspExyPb</code> = 3754,	$[(NLP) exy P B]$ plastic strain exy at the bottom []
<code>rc_sstspE1Pb</code> = 3755,	$[(NLP) e1 P B]$ plastic principal strain 1 at the bottom []
<code>rc_sstspE2Pb</code> = 3756,	$[(NLP) e2 P B]$ plastic principal strain 2 at the bottom []
<code>rc_sstspAePb</code> = 3757,	$[(NLP) ae P B]$ plastic principal strain direction at the bottom [°]
<code>rc_sstspEeffPb</code> = 3758,	$[(NLP) eeff P B]$ effective plastic strain at the bottom []
<code>rc_sstspDeeffPb</code> = 3759,	$[(NLP) deeff P B]$ effective plastic strain increment at the bottom []
<code>rc_sstspLeeffT</code> = 3760,	$[(NLE) eeff T]$ effective strain at the top []
<code>rc_sstspLeeffC</code> = 3761,	$[(NLE) eeff C]$ effective strain at the center []
<code>rc_sstspLeeffB</code> = 3762,	$[(NLE) eeff B]$ effective strain at the bottom []

Vertical deflection

<code>rc_vdW1</code> = 3800,	$[w1] w1 [m]$
<code>rc_vdW2</code> = 3801,	$[w2] w2 [m]$
<code>rc_vdW3</code> = 3802,	$[w3] w3 [m]$
<code>rc_vdWTot</code> = 3803,	$[wtot] wtot [m]$
<code>rc_vdWbij</code> = 3804,	$[wbij] wbij [m]$

Design results of RC cores and walls

rc_rccwUBeam = 3900,
rc_rccwUStrip = 3901,
rc_rccwUOverall = 3902,

[Utilization beam] utilization beam []
[Utilization strip] utilization strip []
[Overall utilization] overall utilization []

Spring deformations

rc_rsdEx = 4000,
rc_rsdEy = 4001,
rc_rsdEz = 4002,
rc_rsdFx = 4003,
rc_rsdFy = 4004,
rc_rsdFz = 4005,

[ex] spring deformation in local x direction [m]
[ey] spring deformation in local y direction [m]
[ez] spring deformation in local z direction [m]
[fx] spring rotational deformation about the local x axis [rad]
[fy] spring rotational deformation about the local y axis [rad]
[fz] spring rotational deformation about the local z axis [rad]

Spring nonlinear results

rc_snIFyx = 4100,
rc_snIUX = 4101,
rc_snIEEx = 4102,
rc_snIPEx = 4103,
rc_snIPEeffx = 4104,

[(NL) Fy,x] spring actual limit force in local x direction [kN]
[(NL) Util.,x] spring utilization in local x direction []
[(NLP) e,x E] spring elastic deformation in local x direction [m]
[(NLP) e,x P] spring plastic deformation in local x direction [m]
[(NLP) eeff,x P] spring effective plastic deformation in local x direction [m]

rc_snIPdeeffx = 4105,

[(NLP) deeff,x P] spring effective plastic deformation increment in local x direction [m]

rc_snIBx = 4106,
rc_snIFyy = 4107,
rc_snIUY = 4108,
rc_snIEEy = 4109,
rc_snIPEy = 4110,
rc_snIPEeffy = 4111,

[(NLP) B,x] back stress in local x direction [kN]
[(NL) Fy,y] spring actual limit force in local y direction [kN]
[(NL) Util.,y] spring utilization in local y direction []
[(NLP) e,y E] spring elastic deformation in local y direction [m]
[(NLP) e,y P] spring plastic deformation in local y direction [m]
[(NLP) eeff,y P] spring effective plastic deformation in local y direction [m]

rc_snIPdeeffy = 4112,

[(NLP) deeff,y P] spring effective plastic deformation increment in local y direction [m]

rc_snIBy = 4113,
rc_snIFyz = 4114,
rc_snIUz = 4115,
rc_snIEEz = 4116,
rc_snIP Ez = 4117,
rc_snIPEeffz = 4118,

[(NLP) B,y] back stress in local y direction [kN]
[(NL) Fy,z] spring actual limit force in local z direction [kN]
[(NL) Util.,z] spring utilization in local z direction []
[(NLP) e,z E] spring elastic deformation in local z direction [m]
[(NLP) e,z P] spring plastic deformation in local z direction [m]
[(NLP) eeff,z P] spring effective plastic deformation in local z direction [m]

rc_snIPdeeffz = 4119,

[(NLP) deeff,z P] spring effective plastic deformation increment in local z direction [m]

rc_snIBz = 4120,
rc_snIMyxx = 4121,

[(NLP) B,z] back stress in local z direction [kN]
[(NL) My,xx] spring actual limit moment about the local x axis [kNm]

rc_snIUxx = 4122,
rc_snIEExx = 4123,

[(NL) Util.,xx] spring rotational utilization about the local x axis []
[(NLP) e,xx E] spring elastic rotational deformation about the local x axis [rad]

rc_snIPExx = 4124,

[(NLP) e,xx P] spring plastic rotational deformation about the local x axis [rad]

rc_snIPEeffxx = 4125,

[(NLP) eeff,xx P] spring effective plastic rotational deformation increment about the local x axis [rad]

rc_snIPdeeffxx = 4126,

[(NLP) deeff,xx P] spring effective plastic rotational deformation about the local x axis [rad]

rc_snIBxx = 4127,
rc_snIMyyy = 4128,

[(NLP) B,xx] back stress about the local x axis [kNm]
[(NL) My,yy] spring actual limit moment about the local y axis [kNm]

rc_snIUyy = 4129,
rc_snIEEyy = 4130,

[(NL) Util.,yy] spring rotational utilization about the local y axis []
[(NLP) e,yy E] spring elastic rotational deformation about the local y axis [rad]

rc_snIPEffyy = 4131,

[(NLP) e,yy P] spring plastic rotational deformation about the local y axis [rad]

rc_snIPEeffyy = 4132,

[(NLP) eeff,yy P] spring effective plastic rotational deformation increment about the local y axis [rad]

rc_snIPdeeffyy = 4133,

[(NLP) deeff,yy P] spring effective plastic rotational deformation about the local y axis [rad]

rc_snlByy = 4134,	<i>[(NL) B,yy] back stress about the local y axis [kNm]</i>
rc_snlMyzz = 4135,	<i>[(NL) My,zz] spring actual limit moment about the local z axis [kNm]</i>
rc_snlUzz = 4136,	<i>[(NL) Util.,zz] spring rotational utilization about the local z axis []</i>
rc_snlEEzz = 4137,	<i>[(NL) e,zz E] spring elastic rotational deformation about the local z axis [rad]</i>
rc_snlPEzz = 4138,	<i>[(NL) e,zz P] spring plastic rotational deformation about the local z axis [rad]</i>
rc_snIPeffzz = 4139,	<i>[(NL) eeff,zz P] spring effective plastic rotational deformation increment about the local z axis [rad]</i>
rc_snIPdeeffzz = 4140,	<i>[(NL) deeff,zz P] spring effective plastic rotational deformation about the local z axis [rad]</i>
rc_snlBzz = 4141,	<i>[(NL) B,zz] back stress about the local z axis [kNm]</i>
XLAM stresses	
rc_xsSxxmT = 4200,	<i>[Sxx m T] normal stress from bending in local x direction at the top [kN/m²]</i>
rc_xsSyyymT = 4201,	<i>[Syy m T] normal stress from bending in local y direction at the top [kN/m²]</i>
rc_xsSxymT = 4202,	<i>[Sxy m T] Sxy m at the top [kN/m²]</i>
rc_xsSxxmB = 4203,	<i>[Sxx m B] normal stress from bending in local x direction at the bottom [kN/m²]</i>
rc_xsSyyymB = 4204,	<i>[Syy m B] normal stress from bending in local y direction at the bottom [kN/m²]</i>
rc_xsSxymB = 4205,	<i>[Sxy m B] Sxy m at the bottom [kN/m²]</i>
rc_xsSxxn = 4206,	<i>[Sxx n] normal stress from membrane forces in local x direction [kN/m²]</i>
rc_xsSyyyn = 4207,	<i>[Syy n] normal stress from membrane forces in local y direction [kN/m²]</i>
rc_xsSxyn = 4208,	<i>[Sxy n] Sxy n [kN/m²]</i>
rc_xsSxzmax = 4209,	<i>[Sxz max] maximum shear stress perpendicular to local x direction [kN/m²]</i>
rc_xsSyzmax = 4210,	<i>[Syz max] maximum shear stress perpendicular to local y direction [kN/m²]</i>
rc_xsSrxmax = 4211,	<i>[Srx max] relevant rolling shear stress in a direction with x normal [kN/m²]</i>
rc_xsSrymax = 4212,	<i>[Sry max] relevant rolling shear stress in a direction with y normal [kN/m²]</i>
XLAM utilization	
rc_xuMNO = 4300,	<i>[M-N-0 utilization] maximum utilization from bending moments and normal force parallel to grain direction []</i>
rc_xuMN90 = 4301,	<i>[M-N-90 utilization] maximum utilization from bending moments and normal force perpendicular to grain direction []</i>
rc_xuVt = 4302,	<i>[V-T utilization] maximum utilization from shear and torsion []</i>
rc_xuVrN = 4303,	<i>[Vr-N utilization] maximum utilization from rolling shear and normal force []</i>
rc_xuMax = 4304,	<i>[Maximum utilization] maximum utilization []</i>
Design check of masonry wall	
rc_dcmwU = 4400,	<i>[Utilization] utilization []</i>
Spring dynamic results	
rc_sdrVx = 4500,	<i>[(D) v,x] spring deformation rate in local x direction [m/s²]</i>
rc_sdrRDx = 4501,	<i>[(D) R,x D] damper force in local x direction [kN]</i>
rc_sdrRSx = 4502,	<i>[(D) R,x S] total spring-damper force in local x direction [kN]</i>
rc_sdrVy = 4503,	<i>[(D) v,y] spring deformation rate in local y direction [m/s²]</i>
rc_sdrRDy = 4504,	<i>[(D) R,y D] damper force in local y direction [kN]</i>
rc_sdrRSy = 4505,	<i>[(D) R,y S] total spring-damper force in local y direction [kN]</i>
rc_sdrVz = 4506,	<i>[(D) v,z] spring deformation rate in local z direction [m/s²]</i>
rc_sdrRDz = 4507,	<i>[(D) R,z D] damper force in local z direction [kN]</i>
rc_sdrRSz = 4508,	<i>[(D) R,z S] total spring-damper force in local z direction [kN]</i>

rc_sdrVxx = 4509,	$[(D) v_{xx}]$ spring rotational deformation rate about the local x axis [rad/s]
rc_sdrRDxx = 4510,	$[(D) R_{xx} D]$ damper moment about the local x axis [kNm]
rc_sdrRSxx = 4511,	$[(D) R_{xx} S]$ total spring-damper moment about the local x axis [kNm]
rc_sdrVyy = 4512,	$[(D) v_{yy}]$ spring rotational deformation rate about the local y axis [rad/s]
rc_sdrRDyy = 4513,	$[(D) R_{yy} D]$ damper moment about the local y axis [kNm]
rc_sdrRSyy = 4514,	$[(D) R_{yy} S]$ total spring-damper moment about the local y axis [kNm]
rc_sdrVzz = 4515,	$[(D) v_{zz}]$ spring rotational deformation rate about the local z axis [rad/s]
rc_sdrRDzz = 4516,	$[(D) R_{zz} D]$ damper moment about the local z axis [kNm]
rc_sdrRSzz = 4517,	$[(D) R_{zz} S]$ total spring-damper moment about the local z axis [kNm]

Steel design results

rc_sd_eff = 10000

[Utilization] critical steel design utilisation (SLS + ULS envelope) []

rc_sd_1 - rc_sd_31 have design code dependet interpretation

Design code : EC3, NTC

rc_sd_1 = 10001

[N-M-V] interaction of normal force, bending moments and shear forces []

rc_sd_2 = 10002

[N-M-Buckl] interaction of normal force, bending moments and flexural buckling []

rc_sd_3 = 10003

[N-M-LtBuckl] interaction of normal force, bending and lateral torsional buckling []

rc_sd_4 = 10004

[Vy] capacity ratio of shear, y-y axis []

rc_sd_5 = 10005

[Vz] capacity ratio of shear including shear web buckling in z axis []

rc_sd_6 = 10006

[Vw-M-N] interaction of normal force, bending and shear web buckling []

rc_sd_7 = 10007

[NplRd] plastic resistance to axial forces of the gross cross-section for class 1,2 or 3 cross-sections [kN]

rc_sd_8 = 10008

[NeffRd] design effective resistance to normal forces for class 4 cross-sections [kN]

rc_sd_9 = 10009

[VyRd] resistance to shear in y direction [kN]

rc_sd_10 = 10010

[VzRd] resistance to shear in local z direction [kN]

rc_sd_11 = 10011

[VbwRd] resistance to shear buckling, contribution from the web [kN]

rc_sd_12 = 10012

[MelyRd] elastic resistance to bending moment about y axis [kNm]

rc_sd_13 = 10013

[MelzRd] elastic resistance to bending moment about z axis [kNm]

rc_sd_14 = 10014

[MpLyRd] plastic resistance to bending moment about y axis [kNm]

rc_sd_15 = 10015

[MpLzRd] plastic resistance to bending moment about z axis [kNm]

rc_sd_16 = 10016

[MeffyRd] effective resistance to bending moment for class 4 cross-sections about y axis [kNm]

rc_sd_17 = 10017

[MeffzRd] effective resistance to bending moment for class 4 cross-sections about z axis [kNm]

rc_sd_18 = 10018

[NbRd] buckling resistance of the compression member [kN]

rc_sd_19 = 10019

[MbRd] buckling resistance moment [kNm]

rc_sd_20 = 10020

[section class] section class[]

rc_sd_21 = 10021

[Mcr] critical lateral torsional buckling moment [kNm]

rc_sd_22 = 10022

[khiN] flexural buckling resistance reduction factor []

rc_sd_23 = 10023

[khiLT] lateral torsional buckling resistance reduction factor []

rc_sd_24 = 10024

[curve class N] curve class N []

rc_sd_25 = 10025

[curve class LT] curve class LT []

rc_sd_26 = 10026

[Ky] buckling factor []

rc_sd_27 = 10027

[Kz] buckling factor []

rc_sd_28 = 10028

[Critical temperature] critical temperature [C°]

rc_sd_29 = 10029

not used

rc_sd_30 = 10030

[LLT] maximum relative LTB slenderness []

rc_sd_31 = 10031

[LN] maximum relative buckling slenderness []

rc_sd_util_ULS = 10090
rc_sd_util_SLS = 10091

[Utilization ULS] utilization in ultimate limit state (ULS) []
[Utilization SLS] utilization in serviceability limit state (SLS) []

Design code : SIA 26x

rc_sd_1 = 10001

[N-M-V] interaction of normal force, bending moments and shear forces []

rc_sd_2 = 10002

[N-M-Stab] axial force - bending (stability) []

rc_sd_3 = 10003

[Vy] shear y []

rc_sd_4 = 10004

[Vz] shear z []

rc_sd_5 = 10005

[Nrd] axial resistance [kN]

rc_sd_6 = 10006

[NeffRd] effective resistance [kN]

rc_sd_7 = 10007

[VyRd] shear resistance y [kN]

rc_sd_8 = 10008

[VzRd] shear resistance z [kN]

rc_sd_9 = 10009

[MyRd] moment resistance y [kNm]

rc_sd_10 = 10010

[MzRd] moment resistance z [kNm]

rc_sd_11 = 10011

[MeffyRd] effective moment resistance y [kNm]

rc_sd_12 = 10012

[MeffzRd] effective moment resistance z [kNm]

rc_sd_13 = 10013

[NKRd] buckling resistance [kN]

rc_sd_14 = 10014

not used

rc_sd_15 = 10015

[khiK] flexural buckling resistance reduction factor []

rc_sd_16 = 10016

[Curve class K] curve class k []

rc_sd_17 = 10017

[MDRd] lateral-torsional buckling resistance [kNm]

rc_sd_18 = 10018

[Mcr] critical lateral-torsional buckling moment [kNm]

rc_sd_19 = 10019

[khiD] lateral-torsional buckling resistance reduction factor []

rc_sd_20 = 10020

[Curve class D] curve class D []

rc_sd_21 = 10021

[section class] section class []

rc_sd_22 = 10022

[Ky] buckling factor []

rc_sd_23 = 10023

[Kz] buckling factor []

rc_sd_24 = 10024

not used

rc_sd_25 = 10025

not used

rc_sd_26 = 10026

[Critical temperature] critical temperature [C°]

rc_sd_27 = 10027

not used

rc_sd_28 = 10028

not used

rc_sd_29 = 10029

not used

rc_sd_30 = 10030

[LD] maximum relative LTB slenderness []

rc_sd_31 = 10031

[LK] maximum relative buckling slenderness []

rc_sd_util_ULS = 10090

[Utilization ULS] utilization in ultimate limit state (ULS) []

rc_sd_util_SLS = 10091

[Utilization SLS] utilization in serviceability limit state (SLS) []

Design code : NEN

rc_sd_1 = 10001

[N-M-V] interaction of normal force, bending moments and shear forces []

rc_sd_2 = 10002

[N-M-Buckl] interaction of normal force, bending moments and flexural buckling []

rc_sd_3 = 10003

[N-M-LtBuckl] interaction of normal force, bending and lateral torsional buckling []

rc_sd_4 = 10004

[V] shear []

rc_sd_5 = 10005

[Nud] axial resistance [kN]

rc_sd_6 = 10006

[VyuRd] shear resistance y [kN]

rc_sd_7 = 10007

[VzuRd] shear resistance z [kN]

rc_sd_8 = 10008

not used

rc_sd_9 = 10009

not used

rc_sd_10 = 10010

not used

rc_sd_11 = 10011

not used

rc_sd_12 = 10012

[C1] C1 []

rc_sd_13 = 10013

[C2] C2 []

rc_sd_14 = 10014

[oLTB] lateral-torsional buckling coefficient []

rc_sd_15 = 10015

not used

rc_sd_16 = 10016

not used

rc_sd_17 = 10017

not used

rc_sd_18 = 10018

not used

rc_sd_19 = 10019

not used

rc_sd_20 = 10020

not used

rc_sd_21 = 10021	<i>not used</i>
rc_sd_22 = 10022	<i>not used</i>
rc_sd_23 = 10023	<i>not used</i>
rc_sd_24 = 10024	<i>not used</i>
rc_sd_25 = 10025	<i>not used</i>
rc_sd_26 = 10026	<i>not used</i>
rc_sd_27 = 10027	<i>not used</i>
rc_sd_28 = 10028	<i>not used</i>
rc_sd_29 = 10029	<i>not used</i>
rc_sd_30 = 10030	<i>not used</i>
rc_sd_31 = 10031	<i>not used</i>
rc_sd_util_ULS = 10090	<i>not used</i>
rc_sd_util_SLS = 10091	<i>not used</i>

Design code : MSZ

rc_sd_1 = 10001	<i>[N-M-V] interaction of normal force, bending moments and shear forces []</i>
rc_sd_2 = 10002	<i>[N-M-Buckl] interaction of normal force, bending moments and flexural buckling []</i>
rc_sd_3 = 10003	<i>[N-M-LtBuckl] interaction of normal force, bending and lateral torsional buckling []</i>
rc_sd_4 = 10004	<i>[Vy] shear y []</i>
rc_sd_5 = 10005	<i>[Vz] shear z []</i>
rc_sd_6 = 10006	<i>[Sr] stress resultant in the web at the flange []</i>
rc_sd_7 = 10007	<i>[Bkl-w] web buckling []</i>
rc_sd_8 = 10008	<i>[Bkl-f] flange buckling []</i>
rc_sd_9 = 10009	<i>[NH] axial resistance [kN]</i>
rc_sd_10 = 10010	<i>[VyH] shear resistance y [kN]</i>
rc_sd_11 = 10011	<i>[VzH] shear resistance z [kN]</i>
rc_sd_12 = 10012	<i>[MyH] moment resistance yy [kNm]</i>
rc_sd_13 = 10013	<i>[MzH] moment resistance zz [kNm]</i>
rc_sd_14 = 10014	<i>[Nkjh] buckling resistance [kN]</i>
rc_sd_15 = 10015	<i>not used</i>
rc_sd_16 = 10016	<i>not used</i>
rc_sd_17 = 10017	<i>not used</i>
rc_sd_18 = 10018	<i>not used</i>
rc_sd_19 = 10019	<i>not used</i>
rc_sd_20 = 10020	<i>not used</i>
rc_sd_21 = 10021	<i>not used</i>
rc_sd_22 = 10022	<i>not used</i>
rc_sd_23 = 10023	<i>not used</i>
rc_sd_24 = 10024	<i>not used</i>
rc_sd_25 = 10025	<i>not used</i>
rc_sd_26 = 10026	<i>not used</i>
rc_sd_27 = 10027	<i>not used</i>
rc_sd_28 = 10028	<i>not used</i>
rc_sd_29 = 10029	<i>not used</i>
rc_sd_30 = 10030	<i>not used</i>
rc_sd_31 = 10031	<i>not used</i>
rc_sd_util_ULS = 10090	<i>not used</i>
rc_sd_util_SLS = 10091	<i>not used</i>

Design code : STAS

rc_sd_1 = 10001	<i>[N-M-Strength] axial force - bending (strength) []</i>
rc_sd_2 = 10002	<i>[N-M-Stab] axial force - bending (stability) []</i>
rc_sd_3 = 10003	<i>[Vy] shear y []</i>
rc_sd_4 = 10004	<i>[Vz] shear z []</i>
rc_sd_5 = 10005	<i>[Sr] equivalent stress in the web at the flange []</i>
rc_sd_6 = 10006	<i>[Shy] cross-section shear factor y []</i>
rc_sd_7 = 10007	<i>[Shz] cross-section shear factor z []</i>
rc_sd_8 = 10008	<i>[Wely] cross-section elasticity modulus y [m^3] [Welz] cross-section elasticity modulus z [m^3]</i>
rc_sd_9 = 10009	
rc_sd_10 = 10010	<i>[Lmax] maximum slenderness []</i>

<code>rc_sd_11</code> = 10011	<i>[Fimin] minimum buckling reduction factor []</i>
<code>rc_sd_12</code> = 10012	<i>[FiG] lateral buckling reduction factor []</i>
<code>rc_sd_13</code> = 10013	<i>not used</i>
<code>rc_sd_14</code> = 10014	<i>not used</i>
<code>rc_sd_15</code> = 10015	<i>not used</i>
<code>rc_sd_16</code> = 10016	<i>not used</i>
<code>rc_sd_17</code> = 10017	<i>not used</i>
<code>rc_sd_18</code> = 10018	<i>not used</i>
<code>rc_sd_19</code> = 10019	<i>not used</i>
<code>rc_sd_20</code> = 10020	<i>not used</i>
<code>rc_sd_21</code> = 10021	<i>not used</i>
<code>rc_sd_22</code> = 10022	<i>not used</i>
<code>rc_sd_23</code> = 10023	<i>not used</i>
<code>rc_sd_24</code> = 10024	<i>not used</i>
<code>rc_sd_25</code> = 10025	<i>not used</i>
<code>rc_sd_26</code> = 10026	<i>not used</i>
<code>rc_sd_27</code> = 10027	<i>not used</i>
<code>rc_sd_28</code> = 10028	<i>not used</i>
<code>rc_sd_29</code> = 10029	<i>not used</i>
<code>rc_sd_30</code> = 10030	<i>not used</i>
<code>rc_sd_31</code> = 10031	<i>not used</i>
<code>rc_sd_util_ULS</code> = 10090	<i>not used</i>
<code>rc_sd_util_SLS</code> = 10091	<i>not used</i>

Timber design results

`rc_td_eff` = 10100

Timber design utilisation (SLS + ULS envelope)

`rc_td_1 - rc_td_15` have design code dependet interpretation

Design code : EC5, NTC, SIA26x

<code>rc_td_1</code> = 10101 244	<i>[N-M] axial force and bending []</i>
<code>rc_td_2</code> = 10102 245	<i>[N-M-Buckl] compression, bending, and flexural buckling []</i>
<code>rc_td_3</code> = 10103 246	<i>[N-M-LTBuckl] axial force, bending, and lateral-torsional buckling []</i>
<code>rc_td_4</code> = 10104 247	<i>[Vy-Vz-Tx] shear in y and z direction and torsion []</i>
<code>rc_td_5</code> = 10105 248	<i>[My-Vz] apex zone tensile stress perpendicular to the axis []</i>
<code>rc_td_6</code> = 10106 249	<i>[LambdaRely] relative slenderness ratio corresponding to bending about y axis []</i>
<code>rc_td_7</code> = 10107 250	<i>[LambdaRelz] relative slenderness ratio corresponding to bending about z axis []</i>
<code>rc_td_8</code> = 10108 251	<i>[LambdaRelm] relative slenderness for bending []</i>
<code>rc_td_9</code> = 10109 252	<i>[kcy] design compression strength reducing factor due to axial instability []</i>
<code>rc_td_10</code> = 10110 253	<i>[kcj] design compression strength reducing factor due to axial instability []</i>
<code>rc_td_11</code> = 10111 254	<i>[kcrit] design bending strength reducing factor due to lateral torsional instability []</i>
<code>rc_td_12</code> = 10112 255	<i>[kmod] design resistance modification factor for duration of load and moisture content []</i>
<code>rc_td_13</code> = 10113 256	<i>[st90d] tension perpendicular to the grain [kN/m²]</i>
<code>rc_td_14</code> = 10114	<i>not used</i>
<code>rc_td_15</code> = 10115	<i>not used</i>
<code>rc_td_util_ULS</code> = 10190 267	<i>[Utilization ULS] utilization in ultimate limit state (ULS) []</i>
<code>rc_td_util_SLS</code> = 10191 259	<i>[Utilization SLS] utilization in serviceability limit state (SLS) []</i>
}	

Error codes

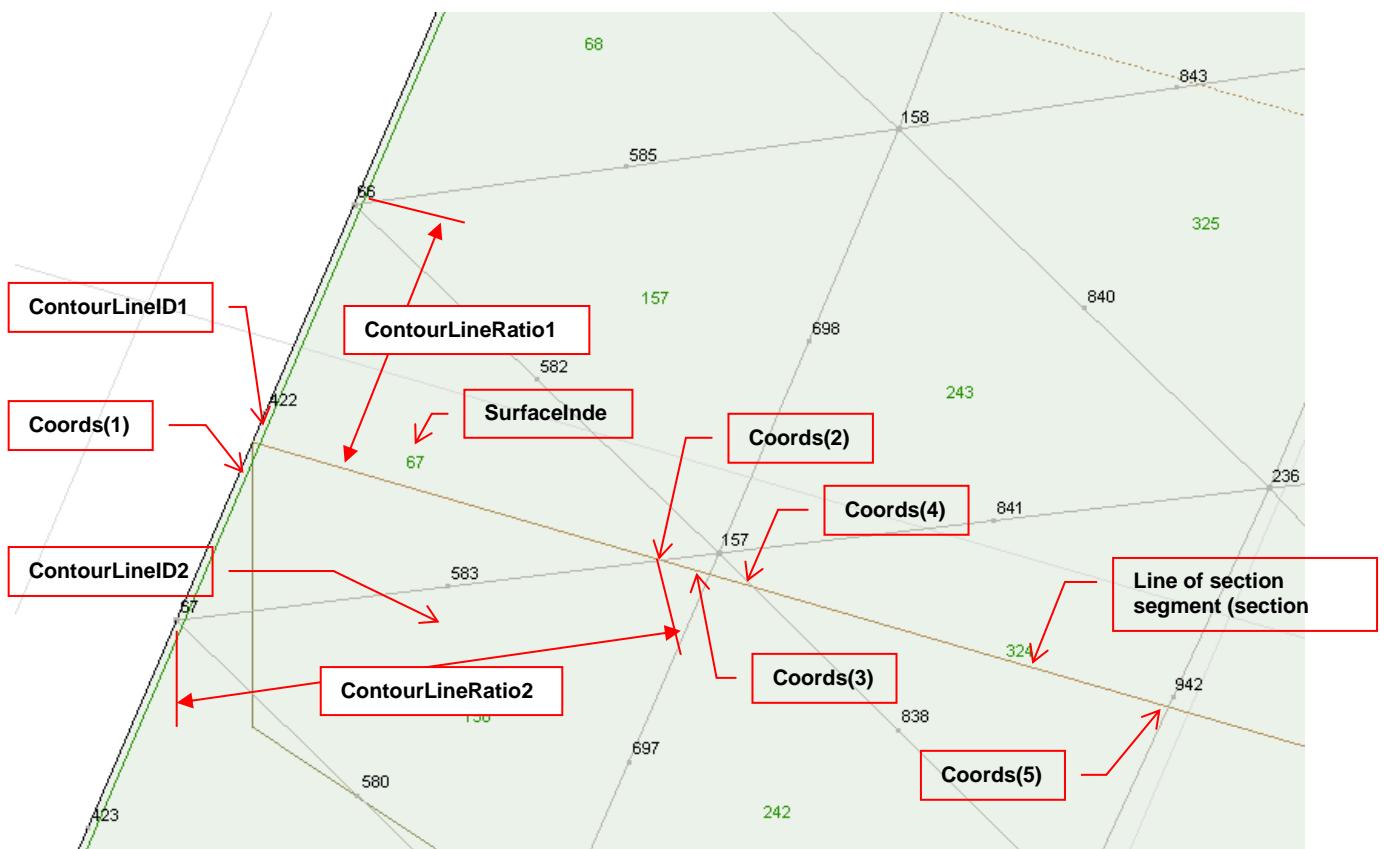
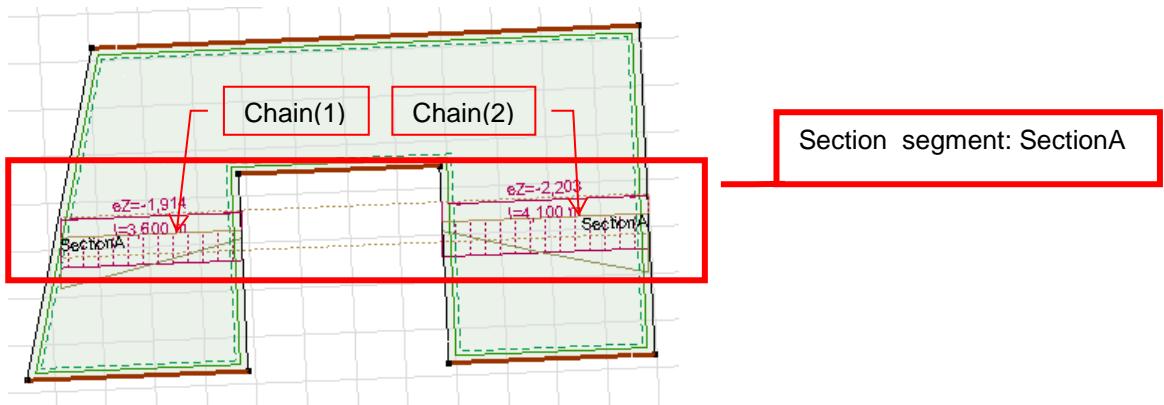
enum ESectionsError = {	
<code>seSegmentDefinitionError</code> = -100001	Segment definition error
<code>seSectionTypeIsNotSegment</code> = -100002	Section type is not a segment
<code>seLoadCaseIndexOutOfBounds</code> = -100003	Load case index invalid
<code>seLoadCombinationIndexOutOfBounds</code> = -100004	Load combination index invalid
<code>seInvalidAnalysisType</code> = -100005 }	Analysis type invalid

Records / structures

RNodeCrackWidthValues = (
RCrackWidthValues	covBottom <i>Crack width values at bottom</i>
RCrackWidthValues	covTop <i>Crack width values at top</i>
)	
RNodeReinforcementValues = (
double	Asbx <i>Reinforcement area in x direction at bottom</i>
double	Asby <i>Reinforcement area in y direction at bottom</i>
double	Astx <i>Reinforcement area in x direction at top</i>
double	Asty <i>Reinforcement area in y direction at top</i>
)	
RResultBlock = (
EAnalysisType	AnalysisType <i>type of analysis</i>
ECombinationType	CombinationType <i>combination type</i>
long	EnvelopeUID <i>unique envelope index</i>
long	LoadCasId <i>load case index (0 < LoadCasId ≤ AxisVMLoadCases.Count)</i>
long	LoadCombinationId <i>load combination index (0 < LoadCombinationId ≤ AxisVMLoadCombinations.Count)</i>
long	LoadLevel <i>load level (increment) index</i>
EMinMaxType	MinMaxType <i>minimum or maximum value</i>
EResultType	ResultType <i>type of the result</i>
)	
RSection = (
ESectionType	SectionType <i>type of the section</i>
BSTR	Name <i>name of the section</i>
ELongBoolean	Visible <i>Supesceded in COM 9.3, use functions GetVisibleSectionIDs and SetVisibleSectionIDs in IAxisVMWindows</i>
RPoint3d	P <i>start point (point in plane) of the section</i>
RPoint3d	N <i>normal vector of the sections plane</i>
RPoint3d	SegmentEndP <i>end point of the section's segment (only if SectionType=stSegment)</i>
ELongBoolean	InAllResultBlocks <i>If true then visible for all loadcases, combinations and envelopes</i>
RResultBlock	ResultBlock <i>Result block (valid only if InAllResultBlocks = lbFalse)</i>
ELongBoolean	ForAllResultComponents <i>If true then visible for all types of results</i>
EResultComponent	ResultComponent <i>component block (valid only if ForAllResultComponents = lbFalse)</i>
ESectionDisplayStyle	DisplayStyle <i>display mode of the section</i>
double	L <i>segment width on the left hand side of the section line (valid only if DisplayMode = sdmDiagramSegWidth) [m]</i>
double	R <i>segment width on the right hand side of the section line (valid only if DisplayMode = sdmDiagramSegWidth) [m]</i>
ELongBoolean	DiagOnBothSide <i>if true then shows diagrams on both sides of the strip (see sections in AxisVM manual) (valid only if DisplayMode = sdmDiagramSegWidth)</i>
ELongBoolean	DiagInPlane <i>show diagram in plane of the element? (valid only if DisplayMode <> sdmResultant)</i>
)	
RSectionElementData = (
long	SurfaceIndex <i>Index of section-crossed surface</i>
long	ContourLineID1 <i>first contour line ID of section-crossed surface</i>
double	ContourLineRatio1 <i>first ratio of cross- section point of contour line and section to surface's vertex point</i>
long	ContourLineID2 <i>second contour line ID of section-crossed surfcae</i>
double	ContourLineRatio2 <i>second ratio of cross- section point of contour line and section to surface's vertex point</i>
)	
RSectionSegmentIntegratedResultant = (
double	N <i>normal force [kN]</i>
double	V <i>shear force [kN]</i>
double	M <i>bending moment [kNm]</i>
)	
RSectionSegmentChainIntegratedParameters = (
RPoint3d	StartPoint <i>first point's coordinates of section's integrated resultant chain</i>
RPoint3d	EndPoint <i>end point's coordinates of section's integrated resultant chain</i>
double	CentreRatio <i>centre ratio</i>
)	
RSurfaceStressValuesTMB = (
RSurfaceStressValues	ssvTop <i>Surface stress values at top</i>
RSurfaceStressValues	ssvMiddle <i>Surface stress values in middle</i>
RSurfaceStressValues	ssvBottom <i>Surface stress values at bottom</i>

)

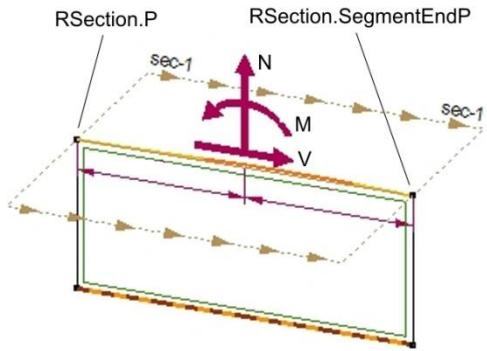
Functions



Section segment labelling

NOTE: Section can consist from 1 or more chain(s), therefore you need to also refer to the actual chain (section segment) with an index.

The picture shows the positive directions of the integrated resultant components. For more information see Section 2.16.15 in the User's Manual.



Functions

long **Add** ([i/o] [RSection](#) **Section**)

Section Section parameters, all parameters should be set

Adds a section to the model. New section will be added to the root directory of sections of same type. Section data must be entered in the proper fields of the item. If successful, returns index of the new section, otherwise returns an error code ([errDatabaseNotReady](#), [seSegmentDefinitionError](#)).

long **Clear**

Deletes all sections from the model.

If successful, returns number of deleted sections, otherwise an error code ([errDatabaseNotReady](#)).

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

Index section index, special index values: 0 = delete all ; -1 = delete all plane sections ; -2 delete all segment sections

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

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

long **GetItem** ([in] long **Index**, [i/o] [RSection](#) **Value**)

Index section index

Get a section by index. If successful, returns index otherwise returns an error code ([errDatabaseNotReady](#), [errIndexOutOfBounds](#)).

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

Index section index

Get number of chains of the section segment otherwise returns an error code ([errDatabaseNotReady](#), [seSectionTypeIsNotSegment](#)). If the section is defined on the boundary of two finite elements, two sections will be created related to both sides.

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

Index section index

Get number of integrated resultant chains of the section segment otherwise returns an error code ([errDatabaseNotReady](#), [seSectionTypeIsNotSegment](#)). If the section is defined on the boundary of two finite elements, two sections will be created related to both sides.

long **GetSegmentChainCoords** ([in] long **Index**, [in] long **ChainIndex**,

[out] SAFEARRAY([RPoint3D](#))* **Coords**)

Index section index

ChainIndex section chain index

Coords section chain 's coordinates

Get section chain's coordinates. If successful, returns number of coordinates, otherwise returns an error code ([errDatabaseNotReady](#), [errIndexOutOfBounds](#), [seSectionTypeIsNotSegment](#)).

long	GetSegmentIntegratedResultantChainCoords ([in] long Index , [in] long IntegratedResultantChainIndex , [out] SAFEARRAY(RPoint3D)* Coords)
	Index section index
	IntegratedResultantChainIndex section integrated resultant chain index
	Coords section integrated resultant chain's coordinates
	Get section integrated resultant chain's coordinates. If successful, returns number of coordinates, otherwise returns an error code (errDatabaseNotReady , errIndexOutOfBounds , seSectionTypeIsNotSegment).
long	GetSegmentChainData ([in] long Index , [in] long ChainIndex , [out] SAFEARRAY(RSectionElementData)* ChainData)
	Index section index
	ChainIndex section chain index, $0 \leq \text{ChainIndex} \leq \text{GetSegmentChainCount}$
	ChainData section chain element data
	Get section chain's element data. If successful, returns number of data elements (corresponds to number of chain-crossed surface elements), otherwise returns an error code (errDatabaseNotReady , errIndexOutOfBounds , seSectionTypeIsNotSegment).
long	GetSegmentIntegratedResultantChainData ([in] long Index , [in] long IntegratedResultantChainIndex , [out] SAFEARRAY(RSectionElementData)* ChainData)
	Index section index
	IntegratedResultantChainIndex section chain index, $0 \leq \text{IntegratedResultantChainIndex} \leq \text{GetSegmentIntegratedResultantChainCount}$
	ChainData section chain element data
	Get section integrated resultant chain's element data. If successful, returns number of data elements (corresponds to number of chain-crossed surface elements), otherwise returns an error code (errDatabaseNotReady , errIndexOutOfBounds , seSectionTypeIsNotSegment).
long	GetSegmentIntegratedResultantParams ([in] long Index , [in] long IntegratedResultantChainIndex , [i/o] RSectionSegmentChainIntegratedParameters IntegratedResultantChainParams)
	Index section index
	IntegratedResultantChainIndex section chain index, $0 \leq \text{IntegratedResultantChainIndex} \leq \text{GetSegmentIntegratedResultantChainCount}$
	IntegratedResultantChainParams section integrated resultant chain's parameters
	Get section integrated resultant chain's parameters. If successful, returns chain index otherwise returns an error code (errDatabaseNotReady , errIndexOutOfBounds , seSectionTypeIsNotSegment).
long	GetSegmentChainDisplacementsByLoadCasId ([in] long Index , [in] long ChainIndex , [in] long LoadCasId , [in] long LoadLevelOrModeShapeOrTimeStep , [in] EAnalysisType AnalysisType , [in] ELongBoolean WithReinforcement , [out] SAFEARRAY(RPoint3D)* Coords , [out] SAFEARRAY(RDisplacementValues)* Displacements , [out] SAFEARRAY (BSTR)* Combinations)
	Index section index
	ChainIndex section chain index
	LoadCasId load case index ($0 < \text{LoadCasId} \leq \text{AxisVMLoadCases.Count}$)
	LoadLevelOrModeShapeOrTimeStep load level (increment) index
	AnalysisType Type of Analysis
	WithReinforcement True if actual reinforcement is considered (this setting is valid only if AnalysisType = atNonLinearStatic)
	Coords section chain's coordinates
	Displacements section chain's displacements
	Combinations array of string with name of load cases.
	Get section chain's displacements. If successful, returns number of displacement values, otherwise returns an error code (errDatabaseNotReady , errIndexOutOfBounds , seSectionTypeIsNotSegment).

long	GetSegmentChainDisplacementsByLoadCombinationId ([in] long Index , [in] long ChainIndex , [in] long LoadCombinationId , [in] long LoadLevelOrModeShapeOrTimeStep , [in] EAnalysisType AnalysisType , [in] ELongBoolean WithReinforcement , [out] SAFEARRAY(RPoint3D)* Coords , [out] SAFEARRAY(RDisplacementValues)* Displacements , [out] SAFEARRAY (BSTR)* Combinations)
	Index section index
	ChainIndex section chain index
	LoadCombinationId load combination index ($0 < LoadCombinationId \leq \text{AxisVMLoadCombinations.Count}$)
	LoadLevelOrModeShapeOrTimeStep load level (increment) index
	AnalysisType Type of Analysis
	WithReinforcement True if actual reinforcement is considered (this setting is valid only if AnalysisType = atNonLinearStatic)
	Coords section chain 's coordinates
	Displacements chain's displacements
	Combinations Array of strings with name of combinations.
	Get section chain's displacements. If successful, returns number of displacements ,otherwise returns an error code (errDatabaseNotReady , errIndexOutOfBounds , seSectionTypeIsNotSegment).
long	GetSegmentChainIntegratedResultantByLoadCaseld ([in] long Index , [in] long ChainIndex , [in] long LoadCaseld , [in] long LoadLevelOrModeShapeOrTimeStep , [in] EAnalysisType AnalysisType , [i/o] RSectionSegmentIntegratedResultant IntegratedResultant)
	Index section index
	ChainIndex section chain index
	LoadCaseld load case index
	LoadLevelOrModeShapeOrTimeStep load level (increment)
	AnalysisType Type of Analysis
	IntegratedResultant integrated resultants: N, V and M, respectively
	Get section chain's integrated resultants by LoadCaseld. If successful, returns the section's index ,otherwise returns an error code(errDatabaseNotReady , errIndexOutOfBounds , seSectionTypeIsNotSegment).
long	GetSegmentChainIntegratedResultantByLoadCombinationId ([in] long Index , [in] long ChainIndex , [in] long LoadCombinationId , [in] long LoadLevelOrModeShapeOrTimeStep , [in] EAnalysisType AnalysisType , [i/o] RSectionSegmentIntegratedResultant IntegratedResultant)
	Index section index
	ChainIndex section chain index
	LoadCombinationId load combination index
	LoadLevelOrModeShapeOrTimeStep load level (increment)
	AnalysisType Type of Analysis
	IntegratedResultant integrated resultants: N, V and M, respectively
	Get section chain's integrated resultants by LoadCombinationId. If successful, returns the section's index ,otherwise returns an error code(errDatabaseNotReady , errIndexOutOfBounds , seSectionTypeIsNotSegment).

long	GetEnvelopeSegmentChainDisplacements ([in] long Index , [in] long ChainIndex , [in] EMinMaxType MinMaxType , [in] EAnalysisType AnalysisType , [in] ELongBoolean WithReinforcement , [in] EDisplacement Component , [out] SAFEARRAY(RPoint3D)* Coords , [out] SAFEARRAY(double) (RDisplacementValues)* Displacements , [out] SAFEARRAY (long)* Combinations)
	Index <i>section index</i>
	ChainIndex <i>section chain index</i>
	MinMaxType <i>Minimum or maximum values</i>
	AnalysisType <i>Type of Analysis</i>
	WithReinforcement <i>True if actual reinforcement is considered (this setting is valid only if AnalysisType = atNonLinearStatic)</i>
	Component <i>Displacement type</i>
	Coords <i>section chain's coordinates</i>
	Displacements <i>section chain's displacements</i>
	Combinations <i>Array of strings with name of combinations.</i>
	<i>Get section chain's displacements. If successful, returns number of displacement values, otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBounds, seSectionTypesIsNotSegment).</i>
long	GetCriticalSegmentChainDisplacements ([in] long Index , [in] long ChainIndex , [in] EMinMaxType MinMaxType , [in] ECombinationType CombinationType , [in] EAnalysisType AnalysisType , [in] ELongBoolean WithReinforcement , [in] EDisplacement Component , [out] SAFEARRAY(RPoint3D)* Coords , [out] SAFEARRAY(RDisplacementValues)* Displacements , [out] SAFEARRAY (BSTR)* Combinations)
	Index <i>section index</i>
	ChainIndex <i>section chain index</i>
	MinMaxType <i>Minimum or maximum values</i>
	CombinationType <i>Combination type</i>
	AnalysisType <i>Type of Analysis</i>
	WithReinforcement <i>True if actual reinforcement is considered (this setting is valid only if AnalysisType = atNonLinearStatic)</i>
	Component <i>Displacement type</i>
	Coords <i>section chain's coordinates</i>
	Displacements <i>section chain's displacements</i>
	Combinations <i>Array of strings with name of combinations.</i>
	<i>Get section chain's displacements. If successful, returns number of displacement values, otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBounds, seSectionTypesIsNotSegment).</i>
long	GetCriticalSegmentChainIntegratedResultant ([in] long Index , [in] long ChainIndex , [in] ECombinationType CombinationType , [in] EMaxMinType MinMaxType , [in] EAnalysisType AnalysisType , [in] ESectionSegmentChainIntegratedResultant SectionSegmentChainIntegratedResultant , [i/o] RSectionSegmentIntegratedResultant IntegratedResultant , [out] ECombinationType CriticalCombinationType , [out] SAFEARRAY(double)* Factors , [out] SAFEARRAY (long)* LoadCaselds)
	Index <i>section index</i>
	ChainIndex <i>section chain index</i>
	CombinationType <i>Combination type</i>
	MinMaxType <i>Minimum or maximum values</i>
	AnalysisType <i>Type of Analysis</i>
	SectionSegmentChainIntegratedResultant <i>governing integrated resultants for envelope: N, V and M, respectively</i>
	IntegratedResultant <i>integrated resultants: N, V and M, respectively</i>
	CriticalCombinationType <i>critical combination type</i>
	Factors <i>factors related to critical combination</i>
	LoadCaselds <i>critical load case index</i>
	<i>Get section chain's critical integrated resultants. If successful, returns the section's index otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBounds, seSectionTypesIsNotSegment).</i>

long **GetEnvelopeSegmentChainIntegratedResultant** ([in] long **Index**, [in] long **ChainIndex**,
 [in] long **EnvelopeUID**, [in] **EMinMaxType** **MinMaxType**, [in] **EAnalysisType** **AnalysisType**,
 [in] **ESectionSegmentChainIntegratedResultant** **SectionSegmentChainIntegratedResultant**,
 [i/o] **RSectionSegmentIntegratedResultant** **IntegratedResultant**,
 [out] long **LoadCaseOrCombinationId**, [out] long **LoadLevelOrModeShapeOrTimeStep**)

Index	<i>section index</i>
ChainIndex	<i>section chain index</i>
EnvelopeUID	<i>unique envelope index</i>
MinMaxType	<i>Minimum or maximum values</i>
AnalysisType	<i>Type of Analysis</i>
SectionSegmentChainIntegratedResultant	<i>governing integrated resultants for envelope: N, V and M, respectively</i>
IntegratedResultant	<i>integrated resultants: N, V and M, respectively</i>
LoadCaseOrCombinationId	<i>load combination index</i>
LoadLevelOrModeShapeOrTimeStep	<i>load level (increment)</i>

Get section chain's integrated resultants envelope. If successful, returns the section's index ,otherwise returns an error code([errDatabaseNotReady](#), [errIndexOutOfBounds](#), [seSectionTypeIsNotSegment](#)).

long **GetSegmentChainVelocitiesByLoadCaseld** ([in] long **Index**, [in] long **ChainIndex**,
 [in] long **LoadCaseld**, [in] long **TimeStep**, [in] **EAnalysisType** **AnalysisType**,
 [out] SAFEARRAY([RPoint3D](#))* **Coords**, [out] SAFEARRAY([RVelocityValues](#))* **VelocityValues**,
 [out] SAFEARRAY (BSTR)* **Combinations**)

Index	<i>section index</i>
ChainIndex	<i>section chain index</i>
LoadCaseld	<i>load case index</i> <i>(0 < LoadCaseld ≤ AxisVMLoadCases.Count)</i>
TimeStep	<i>Time step</i>
AnalysisType	<i>Type of Analysis</i>
Coords	<i>section chain's coordinates</i>
VelocityValues	<i>section chain's velocity values</i>
Combinations	<i>Array of string with name of load cases.</i>

Get section chain's velocity values. If successful, returns number of velocity values, otherwise returns an error code ([errDatabaseNotReady](#), [errIndexOutOfBounds](#), [seSectionTypeIsNotSegment](#)).

long **GetEnvelopeSegmentChainVelocities** ([in] long **Index**, [in] long **ChainIndex**, [in] **EMinMaxType** **MinMaxType**, [in] **EAnalysisType** **AnalysisType**, [in] **EVelocity** **Component**,
 [out] SAFEARRAY([RPoint3D](#))* **Coords**, [out] SAFEARRAY([RVelocityValues](#))* **VelocityValues**,
 [out] SAFEARRAY (BSTR)* **Combinations**)

Index	<i>section index</i>
ChainIndex	<i>section chain index</i>
MinMaxType	<i>minimum or maximum values</i>
AnalysisType	<i>Type of Analysis</i>
Component	<i>Velocity type</i>
Coords	<i>section chain's coordinates</i>
VelocityValues	<i>section chain's velocity values</i>
Combinations	<i>Array of string with name of load cases.</i>

Get section chain's velocity values. If successful, returns number of velocity values , otherwise returns an error code ([errDatabaseNotReady](#), [errIndexOutOfBounds](#), [seSectionTypeIsNotSegment](#)).

long	GetSegmentChainAccelerationsByLoadCasId ([in] long Index , [in] long ChainIndex , [in] long LoadCasId , [in] long TimeStep , [in] EAnalysisType AnalysisType , [out] SAFEARRAY(RPoint3D)* Coords , [out] SAFEARRAY(RAccelerationValues)* AccelerationValues , [out] SAFEARRAY (BSTR)* Combinations)
	Index <i>section index</i> ChainIndex <i>section chain index</i> LoadCasId <i>load case index</i> <i>(0 < LoadCasId ≤ AxisVMLoadCases.Count)</i> TimeStep <i>Time step</i> AnalysisType <i>Type of Analysis</i> Coords <i>section chain's coordinates</i> AccelerationValues <i>section chain's acceleration values</i> Combinations <i>Array of string with name of load cases.</i>
	<i>Get section chain's acceleration values.</i> <i>If successful, returns number of acceleration values , otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBounds, seSectionTypeIsNotSegment).</i>
long	GetEnvelopeSegmentChainAccelerations ([in] long Index , [in] long ChainIndex , [in] EMinMaxType MinMaxType , [in] EAnalysisType AnalysisType , [in] EAcceleration Component , [out] SAFEARRAY(RPoint3D)* Coords , [out] SAFEARRAY(RAccelerationValues)* VelocityValues , [out] SAFEARRAY (BSTR)* Combinations)
	Index <i>section index</i> ChainIndex <i>section chain index</i> MinMaxType <i>minimum or maximum values</i> AnalysisType <i>Type of Analysis</i> Component <i>acceleration type</i> Coords <i>section chain's coordinates</i> AccelerationValues <i>section chain's acceleration values</i> Combinations <i>Array of string with name of load cases.</i>
	<i>Get section chain's acceleration values</i> <i>If successful, returns number of acceleration values, otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBounds, seSectionTypeIsNotSegment).</i>
long	GetSegmentChainSurfaceForcesByLoadCasId ([in] long Index , [in] long ChainIndex , [in] long LoadCasId , [in] long LoadLevelOrModeShapeOrTimeStep , [in] EAnalysisType AnalysisType , [out] SAFEARRAY(RPoint3D)* Coords , [out] SAFEARRAY(RSurfaceForceValues)* Forces , [out] SAFEARRAY (BSTR)* Combinations)
	Index <i>section index</i> ChainIndex <i>section chain index</i> LoadCasId <i>load case index</i> <i>(0 < LoadCasId ≤ AxisVMLoadCases.Count)</i> LoadLevelOrModeShapeOrTimeStep <i>load level (increment) index</i> AnalysisType <i>Type of Analysis</i> Coords <i>section chain's coordinates</i> Forces <i>section chain's forces</i> Combinations <i>array of string with name of load cases.</i>
	<i>Get section chain's surface forces. Output arrays may contain duplications, but values can also vary at the same location. If successful, returns number of force values , otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBounds, seSectionTypeIsNotSegment).</i>

long **GetSegmentChainSurfaceForcesByLoadCombinationId** ([in] long **Index**, [in] long **ChainIndex**, [in] long **LoadCombinationId**, [in] long **LoadLevelOrModeShapeOrTimeStep**, [in] [EAnalysisType AnalysisType](#), [out] SAFEARRAY([RPoint3D](#))* **Coords**, [out] SAFEARRAY([RSurfaceForceValues](#))* **Forces**, [out] SAFEARRAY (BSTR)* **Combinations**)

Index section index

ChainIndex section chain index

LoadCombinationId load combination index

($0 < LoadCombinationId \leq \text{AxisVMLoadCombinations.Count}$)

LoadLevelOrModeShapeOrTimeStep load level (increment) index

AnalysisType Type of [Analysis](#)

Coords section chain 's coordinates

Forces section chain's forces

Combinations Array of strings with name of combinations.

Get section chain's surface forces. Output arrays may contain duplications, but values can also vary at the same location.If successful, returns number of force values, otherwise returns an error code ([errDatabaseNotReady](#), [errIndexOutOfBounds](#), [seSectionTypeIsNotSegment](#)).

long **GetEnvelopeSegmentChainSurfaceForces** ([in] long **Index**, [in] long **ChainIndex**, [in] [EMinMaxType MinMaxType](#), [in] [EAnalysisType AnalysisType](#), [in] [ESurfaceForce Component](#), [out] SAFEARRAY([RPoint3D](#))* **Coords**, [out] SAFEARRAY([RSurfaceForceValues](#))* **Forces**, [out] SAFEARRAY (BSTR)* **Combinations**)

Index section index

ChainIndex section chain index

MinMaxType Minimum or maximum values

AnalysisType Type of [Analysis](#)

Component Force type

Coords section chain's coordinates

Forces section chain's forces

Combinations Array of strings with name of combinations.

Get section chain's surface forces. Output arrays may contain duplications, but values can also vary at the same location.If successful, returns number of force values , otherwise returns an error code ([errDatabaseNotReady](#), [errIndexOutOfBounds](#), [seSectionTypeIsNotSegment](#)).

long **GetCriticalSegmentChainSurfaceForces** ([in] long **Index**, [in] long **ChainIndex**, [in] [EMinMaxType MinMaxType](#), [in] [ECombinationType CombinationType](#), [in] [EAnalysisType AnalysisType](#), [in] [ESurfaceForce Component](#), [out] SAFEARRAY([RPoint3D](#))* **Coords**, [out] SAFEARRAY([RSurfaceForceValues](#))* **Forces**, [out] SAFEARRAY (BSTR)* **Combinations**)

Index section index

ChainIndex section chain index

MinMaxType Minimum or maximum values

CombinationType Combination type

AnalysisType Type of [Analysis](#)

Component Force type

Coords section chain's coordinates

Forces section chain's forces

Combinations Array of strings with name of combinations.

Get section chain's surface forces. Output arrays may contain duplications, but values can also vary at the same location.If successful, returns number of force values , otherwise returns an error code ([errDatabaseNotReady](#), [errIndexOutOfBounds](#), [seSectionTypeIsNotSegment](#)).

long	GetSegmentChainCalculatedReinforcementsByLoadCaseId ([in] long Index , [in] long ChainIndex , [in] long LoadCaseId , [in] long LoadLevelOrModeShapeOrTimeStep , [in] EAnalysisType AnalysisType , [out] SAFEARRAY(RPoint3D)* Coords , [out] SAFEARRAY(RNodeReinforcementValues)* Reinforcements , [out] SAFEARRAY (BSTR)* Combinations)
	Index section index ChainIndex section chain index LoadCaseId load case index <i>(0 < LoadCaseId ≤ AxisVMLoadCases.Count)</i> LoadLevel load level (increment) index AnalysisType Type of <u>Analysis</u> Coords section chain's coordinates Reinforcements section chain's reinforcement values Combinations array of string with name of load cases.
	<i>Get section chain's reinforcement values. Output arrays may contain duplications, but values can also vary at the same location. If successful, returns number of reinforcement values , otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBounds, seSectionTypeIsNotSegment).</i>
long	GetSegmentChainCalculatedReinforcementsByLoadCombinationId ([in] long Index , [in] long ChainIndex , [in] long LoadCombinationId , [in] long LoadLevelOrModeShapeOrTimeStep , [in] EAnalysisType AnalysisType , [out] SAFEARRAY(RPoint3D)* Coords , [out] SAFEARRAY(RNodeReinforcementValues)* Reinforcements , [out] SAFEARRAY (BSTR)* Combinations)
	Index section index ChainIndex section chain index LoadCombinationId load combination index <i>(0 < LoadCombinationId ≤ AxisVMLoadCombinations.Count)</i> LoadLevel load level (increment) index AnalysisType Type of <u>Analysis</u> Coords section chain 's coordinates Reinforcements section chain's reinforcement values Combinations Array of strings with name of combinations.
	<i>Get section chain's reinforcement values. Output arrays may contain duplications, but values can also vary at the same location. If successful, returns number of reinforcement values , otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBounds, seSectionTypeIsNotSegment).</i>
long	GetEnvelopeSegmentChainCalculatedReinforcements ([in] long Index , [in] long ChainIndex , [in] EMinMaxType MinMaxType , [in] EAnalysisType AnalysisType , [in] EReinforcement Component , [out] SAFEARRAY(RPoint3D)* Coords , [out] SAFEARRAY(RNodeReinforcementValues)* Reinforcements , [out] SAFEARRAY (BSTR)* Combinations)
	Index section index ChainIndex section chain index MinMaxType Minimum or maximum values AnalysisType Type of <u>Analysis</u> Component Reinforcement type Coords section chain's coordinates Reinforcements section chain's reinforcement values Combinations Array of strings with name of combinations.
	<i>Get section chain's reinforcement values. Output arrays may contain duplications, but values can also vary at the same location.If successful, returns number of reinforcement values otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBounds, seSectionTypeIsNotSegment).</i>

long **GetCriticalSegmentChainCalculatedReinforcements** ([in] long **Index**, [in] long **ChainIndex**, [in] **EMinMaxType** **MinMaxType**, [in] **ECombinationType** **CombinationType**, [in] **EAnalysisType** **AnalysisType**, [in] **EReinforcement** **Component**, [out] SAFEARRAY(**RPoint3D**)* **Coords**, [out] SAFEARRAY(**RNodeReinforcementValues**)* **Reinforcements**, [out] SAFEARRAY (BSTR)* **Combinations**)

Index section index

ChainIndex section chain index

MinMaxType Minimum or maximum values

CombinationType Combination type

AnalysisType Type of Analysis

Component Reinforcement type

Coords section chain's coordinates

Reinforcements section chain's reinforcement values

Combinations Array of strings with name of combinations.

Get section chain's reinforcement values. Output arrays may contain duplications, but values can also vary at the same location. If successful, returns number of reinforcement values , otherwise returns an error code ([errDatabaseNotReady](#), [errIndexOutOfBounds](#), [seSectionTypeIsNotSegment](#)).

long **GetSegmentChainSurfaceSupportForcesByLoadCaseId** ([in] long **Index**, [in] long **ChainIndex**, [in] long **LoadCaseId**, [in] long **LoadLevelOrTimeStep**, [in] **EAnalysisType** **AnalysisType**, [out] SAFEARRAY(**RPoint3D**)* **Coords**, [out] SAFEARRAY(**RSurfaceSupportForces**)* **Forces**, [out] SAFEARRAY (BSTR)* **Combinations**)

Index section index

ChainIndex section chain index

LoadCaseId load case index

($0 < LoadCaseId \leq AxisVMLoadCases.Count$)

LoadLevelOrTimeStep load level (increment) index

AnalysisType Type of Analysis

Coords section chain's coordinates

Forces section chain's surface support forces

Combinations array of string with name of load cases.

Get section chain's surface support forces. Output arrays may contain duplications, but values can also vary at the same location. If successful, returns number of surface support forces otherwise returns an error code ([errDatabaseNotReady](#), [errIndexOutOfBounds](#), [seSectionTypeIsNotSegment](#)).

long **GetSegmentChainSurfaceSupportForcesByLoadCombinationId** ([in] long **Index**, [in] long **ChainIndex**, [in] long **LoadCombinationId**, [in] long **LoadLevelOrTimeStep**, [in] **EAnalysisType** **AnalysisType**, [out] SAFEARRAY(**RPoint3D**)* **Coords**, [out] SAFEARRAY(**RSurfaceSupportForces**)* **Forces**, [out] SAFEARRAY (BSTR)* **Combinations**)

Index section index

ChainIndex section chain index

LoadCombinationId load combination index

($0 < LoadCombinationId \leq AxisVMLoadCombinations.Count$)

LoadLevelOrTimeStep load level (increment) index

AnalysisType Type of Analysis

Coords section chain 's coordinates

Forces section chain's surface support forces

Combinations Array of strings with name of combinations.

Get section chain's surface support forces. Output arrays may contain duplications, but values can also vary at the same location.If successful, returns number of surface support forces , otherwise returns an error code ([errDatabaseNotReady](#), [errIndexOutOfBounds](#), [seSectionTypeIsNotSegment](#)).

long	GetEnvelopeSegmentChainSurfaceSupportForces ([in] long Index , [in] long ChainIndex , [in] EMinMaxType MinMaxType , [in] EAnalysisType AnalysisType , [in] ESurfaceSupportForce Component , [out] SAFEARRAY(RPoint3D) [*] Coords , [out] SAFEARRAY(RSurfaceSupportForces) [*] Forces , [out] SAFEARRAY (BSTR) [*] Combinations)
	Index section index
	ChainIndex section chain index
	MinMaxType Minimum or maximum values
	AnalysisType Type of <u>Analysis</u>
	Component surface support force type
	Coords section chain's coordinates
	Forces section chain's surface support forces
	Combinations Array of strings with name of combinations.
	Get section chain's surface support forces. Output arrays may contain duplications, but values can also vary at the same location. If successful, returns number of surface support forces , otherwise returns an error code (errDatabaseNotReady , errIndexOutOfBounds , seSectionTypesNotSegment).
long	GetCriticalSegmentChainSurfaceSupportForces ([in] long Index , [in] long ChainIndex , [in] EMinMaxType MinMaxType , [in] ECombinationType CombinationType , [in] EAnalysisType AnalysisType , [in] ESurfaceSupportForce Component , [out] SAFEARRAY(RPoint3D) [*] Coords , [out] SAFEARRAY(RSurfaceSupportForces) [*] Forces , [out] SAFEARRAY (BSTR) [*] Combinations)
	Index section index
	ChainIndex section chain index
	MinMaxType Minimum or maximum values
	CombinationType Combination type
	AnalysisType Type of <u>Analysis</u>
	Component surface support force type
	Coords section chain's coordinates
	Forces section chain's surface support forces
	Combinations Array of strings with name of combinations.
	Get section chain's surface support forces. Output arrays may contain duplications, but values can also vary at the same location. If successful, returns number of surface support forces , otherwise returns an error code (errDatabaseNotReady , errIndexOutOfBounds , seSectionTypesNotSegment).
long	GetSegmentChainCrackWidthsByLoadCaselId ([in] long Index , [in] long ChainIndex , [in] long LoadCaselId , [in] long LoadLevel , [in] EAnalysisType AnalysisType ,[out] SAFEARRAY(RPoint3D) [*] Coords , [out] SAFEARRAY(RNodeCrackWidthValues) [*] CrackWidths , [out] SAFEARRAY (BSTR) [*] Combinations)
	Index section index
	ChainIndex section chain index
	LoadCaselId load case index ($0 < \text{LoadCaselId} \leq \text{AxisVMLoadCases.Count}$)
	LoadLevel load level (increment) index
	AnalysisType Type of <u>Analysis</u>
	Coords section chain's coordinates
	CrackWidths section chain's crack width values
	Combinations array of string with name of load cases.
	Get section chain's crack width values. Output arrays may contain duplications, but values can also vary at the same location. If successful, returns number of crack width values , otherwise returns an error code (errDatabaseNotReady , errIndexOutOfBounds , seSectionTypesNotSegment).

long **GetSegmentChainCrackWidthsByLoadCombinationId** ([in] long **Index**, [in] long **ChainIndex**, [in] long **LoadCombinationId**, [in] long **LoadLevel**, [in] [EAnalysisType](#) **AnalysisType**, [out] SAFEARRAY([RPoint3D](#))* **Coords**, [out] SAFEARRAY([RNodeCrackWidthValues](#))* **CrackWidths**, [out] SAFEARRAY (BSTR)* **Combinations**)

Index section index

ChainIndex section chain index

LoadCombinationId load combination index

($0 < LoadCombinationId \leq \text{AxisVMLoadCombinations.Count}$)

LoadLevel load level (increment) index

AnalysisType Type of [Analysis](#)

Coords section chain's coordinates

CrackWidths section chain's crack width values

Combinations Array of strings with name of combinations.

Get section chain's crack width values. Output arrays may contain duplications, but values can also vary at the same location. If successful, returns number of crack width values, otherwise returns an error code ([errDatabaseNotReady](#), [errIndexOutOfBounds](#), [seSectionTypeIsNotSegment](#)).

long **GetEnvelopeSegmentChainCrackWidths** ([in] long **Index**, [in] long **ChainIndex**, [in] [EMinMaxType](#) **MinMaxType**, [in] [EAnalysisType](#) **AnalysisType**, [in] [ECrackWidth](#) **Component**, [out] SAFEARRAY([RPoint3D](#))* **Coords**, [out] SAFEARRAY([RNodeCrackWidthValues](#))* **CrackWidths**, [out] SAFEARRAY (BSTR)* **Combinations**)

Index section index

ChainIndex section chain index

MinMaxType Minimum or maximum values

AnalysisType Type of [Analysis](#)

Component crack width component

Coords section chain's coordinates

CrackWidths section chain's crack width values

Combinations Array of strings with name of combinations.

Get section chain's crack width values. Output arrays may contain duplications, but values can also vary at the same location. If successful, returns number of crack width values, otherwise returns an error code ([errDatabaseNotReady](#), [errIndexOutOfBounds](#), [seSectionTypeIsNotSegment](#)).

long **GetCriticalSegmentChainCrackWidths** ([in] long **Index**, [in] long **ChainIndex**, [in] [EMinMaxType](#) **MinMaxType**, [in] [ECombinationType](#) **CombinationType**, [in] [EAnalysisType](#) **AnalysisType**, [in] [ECrackWidth](#) **Component**, [out] SAFEARRAY([RPoint3D](#))* **Coords**, [out] SAFEARRAY([RNodeCrackWidthValues](#))* **CrackWidths**, [out] SAFEARRAY (BSTR)* **Combinations**)

Index section index

ChainIndex section chain index

MinMaxType Minimum or maximum values

CombinationType Combination type

AnalysisType Type of [Analysis](#)

Component crack width component

Coords section chain's coordinates

CrackWidths section chain's crack width values

Combinations Array of strings with name of combinations.

Get section chain's crack width values. Output arrays may contain duplications, but values can also vary at the same location. If successful, returns number of crack width values, otherwise returns an error code ([errDatabaseNotReady](#), [errIndexOutOfBounds](#), [seSectionTypeIsNotSegment](#)).

long **GetSegmentChainShearCapacitiesByLoadCaseId** ([in] long **Index**, [in] long **ChainIndex**, [in] long **LoadCaseId**, [in] long **LoadLevel**, [in] [EAnalysisType AnalysisType](#), [out] SAFEARRAY([RPoint3D](#))* **Coords**, [out] SAFEARRAY([RShearCapacities](#))* **ShearCapacities**, [out] SAFEARRAY (BSTR)* **Combinations**)

Index section index
ChainIndex section chain index
LoadCaseId load case index
($0 < LoadCaseId \leq \text{AxisVMLoadCases.Count}$)
LoadLevel load level (increment) index
AnalysisType Type of [Analysis](#)
Coords section chain's coordinates
ShearCapacities section chain's shear capacities
Combinations array of string with name of load cases.

Get section chain's shear capacity values. Output arrays may contain duplications, but values can also vary at the same location. If successful, returns number of shear capacity values, otherwise returns an error code ([errDatabaseNotReady](#), [errIndexOutOfBounds](#), [seSectionTypeIsNotSegment](#)).

long **GetSegmentChainShearCapacitiesByLoadCombinationId** ([in] long **Index**, [in] long **ChainIndex**, [in] long **LoadCombinationId**, [in] long **LoadLevel**, [in] [EAnalysisType AnalysisType](#), [out] SAFEARRAY([RPoint3D](#))* **Coords**, [out] SAFEARRAY([RShearCapacities](#))* **ShearCapacities**, [out] SAFEARRAY (BSTR)* **Combinations**)

Index section index
ChainIndex section chain index
LoadCombinationId load combination index
($0 < LoadCombinationId \leq \text{AxisVMLoadCombinations.Count}$)
LoadLevel load level (increment) index
AnalysisType Type of [Analysis](#)
Coords section chain 's coordinates
ShearCapacities section chain's shear capacities
Combinations Array of strings with name of combinations.

Get section chain's shear capacity values

If successful, returns number of shear capacity values, otherwise returns an error code ([errDatabaseNotReady](#), [errIndexOutOfBounds](#), [seSectionTypeIsNotSegment](#)).

long **GetEnvelopeSegmentChainShearCapacities** ([in] long **Index**, [in] long **ChainIndex**, [in] [EMinMaxType MinMaxType](#), [in] [EAnalysisType AnalysisType](#), [in] [EShearCapacity Component](#), [out] SAFEARRAY([RPoint3D](#))* **Coords**, [out] SAFEARRAY([RShearCapacities](#))* **ShearCapacities**, [out] SAFEARRAY (BSTR)* **Combinations**)

Index section index
ChainIndex section chain index
MinMaxType Minimum or maximum values
AnalysisType Type of [Analysis](#)
Component Shear capacity type
Coords section chain's coordinates
ShearCapacities section chain's shear capacities
Combinations Array of strings with name of combinations.

Get section chain's shear capacity values. Output arrays may contain duplications, but values can also vary at the same location. If successful, returns number of shear capacity values, otherwise returns an error code ([errDatabaseNotReady](#), [errIndexOutOfBounds](#), [seSectionTypeIsNotSegment](#)).

long	GetCriticalSegmentChainShearCapacities ([in] long Index , [in] long ChainIndex , [in] EMinMaxType MinMaxType , [in] ECombinationType CombinationType , [in] EAnalysisType AnalysisType , [in] EShearCapacity Component , [out] SAFEARRAY(RPoint3D)* Coords , [out] SAFEARRAY(RShearCapacities)* ShearCapacities , [out] SAFEARRAY (BSTR)* Combinations)
	Index <i>section index</i>
	ChainIndex <i>section chain index</i>
	MinMaxType <i>Minimum or maximum values</i>
	CombinationType <i>Combination type</i>
	AnalysisType <i>Type of Analysis</i>
	Component <i>Shear capacity type</i>
	Coords <i>section chain's coordinates</i>
	ShearCapacities <i>section chain's shear capacities</i>
	Combinations <i>Array of strings with name of combinations.</i>
	<i>Get section chain's shear capacity values. Output arrays may contain duplications, but values can also vary at the same location. If successful, returns number of shear capacity values, otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBounds, seSectionTypeIsNotSegment).</i>
long	GetSegmentChainSurfaceStressesByLoadCaselId ([in] long Index , [in] long ChainIndex , [in] long LoadCaselId , [in] long LoadLevelOrTimeStep , [in] EAnalysisType AnalysisType , [out] SAFEARRAY(RPoint3D)* Coords , [out] SAFEARRAY(RSurfaceStressValuesTMB)* Stresses , [out] SAFEARRAY (BSTR)* Combinations)
	Index <i>section index</i>
	ChainIndex <i>section chain index</i>
	LoadCaselId <i>load case index</i> <i>(0 < LoadCaselId ≤ AxisVMLoadCases.Count)</i>
	LoadLevelOrTimeStep <i>load level (increment) index or time step</i>
	AnalysisType <i>Type of Analysis</i>
	Coords <i>section chain's coordinates</i>
	Stresses <i>section chain's surface stress values</i>
	Combinations <i>array of string with name of load cases.</i>
	<i>Get section chain's surface stress values</i>
	<i>If successful, returns number of surface stress values, otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBounds, seSectionTypeIsNotSegment).</i>
long	GetSegmentChainSurfaceStressesByLoadCombinationId ([in] long Index , [in] long ChainIndex , [in] long LoadCombinationId , [in] long LoadLevelOrTimeStep , [in] EAnalysisType AnalysisType , [out] SAFEARRAY(RPoint3D)* Coords , [out] SAFEARRAY(RSurfaceStressValuesTMB)* Stresses , [out] SAFEARRAY (BSTR)* Combinations)
	Index <i>section index</i>
	ChainIndex <i>section chain index</i>
	LoadCombinationId <i>load combination index</i> <i>(0 < LoadCombinationId ≤ AxisVMLoadCombinations.Count)</i>
	LoadLevelOrTimeStep <i>load level (increment) index or time step</i>
	AnalysisType <i>Type of Analysis</i>
	Coords <i>section chain 's coordinates</i>
	Stresses <i>section chain's surface stress values</i>
	Combinations <i>Array of strings with name of combinations.</i>
	<i>Get section chain's surface stress values. Output arrays may contain duplications, but values can also vary at the same location.If successful, returns number of surface stress values, otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBounds, seSectionTypeIsNotSegment).</i>

long **GetEnvelopeSegmentChainSurfaceStresses** ([in] long **Index**, [in] long **ChainIndex**, [in] **EMinMaxType** **MinMaxType**, [in] **EAnalysisType** **AnalysisType**, [in] **ESurfaceStressPosition** **SurfaceStressPosition**, [in] **ESurfaceStress** **Component**, [out] SAFEARRAY(**RPoint3D**)^{*} **Coords**, [out] SAFEARRAY(**RSurfaceStressValuesTMB**)^{*} **Stresses**, [out] SAFEARRAY (BSTR)^{*} **Combinations**)

Index section index
ChainIndex section chain index
MinMaxType Minimum or maximum values
AnalysisType Type of Analysis
SurfaceStressPosition Surface stress position
Component Type of surface stress
Coords section chain's coordinates
Stresses section chain's surface stress values
Combinations Array of strings with name of combinations.

Get section chain's surface stress values. Output arrays may contain duplications, but values can also vary at the same location. If successful, returns number of surface stress values, otherwise returns an error code ([errDatabaseNotReady](#), [errIndexOutOfBounds](#), [seSectionTypeIsNotSegment](#)).

long **GetCriticalSegmentChainSurfaceStresses** ([in] long **Index**, [in] long **ChainIndex**, [in] **EMinMaxType** **MinMaxType**, [in] **ECombinationType** **CombinationType**, [in] **EAnalysisType** **AnalysisType**, [in] **ESurfaceStressPosition** **SurfaceStressPosition**, [in] **ESurfaceStress** **Component**, [out] SAFEARRAY(**RPoint3D**)^{*} **Coords**, [out] SAFEARRAY(**RSurfaceStressValuesTMB**)^{*} **Stresses**, [out] SAFEARRAY (BSTR)^{*} **Combinations**)

Index section index
ChainIndex section chain index
MinMaxType Minimum or maximum values
CombinationType Combination type
AnalysisType Type of Analysis
SurfaceStressPosition Surface stress position
Component Type of surface stress
Coords section chain's coordinates
Stresses section chain's surface stress values
Combinations Array of strings with name of combinations.

Get section chain's surface stress values

If successful, returns number of surface stress values , otherwise returns an error code ([errDatabaseNotReady](#), [errIndexOutOfBounds](#), [seSectionTypeIsNotSegment](#)).

long **SetItem** ([in] long **Index**, [i/o] **RSection**^{*} **Value**)

Index section index

Set a section with a given index. If successful, returns index, otherwise returns an error code ([errDatabaseNotReady](#), [errIndexOutOfBounds](#), [seSegmentDefinitionError](#)).

long **SetUserCreep** ([in] **ELongBoolean** **Creep**)

Creep If *IbTrue* concrete creep will be considered in results of nonlinear analysis, if national design code allows it. More [here...](#)

Enable or disable consideration of concrete creep in nonlinear analysis results. If successful returns 1, otherwise an error code ([errDatabaseNotReady](#), [errCreepNotSupported](#)).

Properties

long **Count**

Get number of sections in the model

long **EnvelopeUID**

Unique index of the envelope used in functions for reading envelope results

[ELongBoolean](#)

UserCreep

Returns `IbTrue` if nonlinear analysis results consider concrete creep. More [here...](#)

IAxisVMSettings

Application settings of AxisVM

Error codes

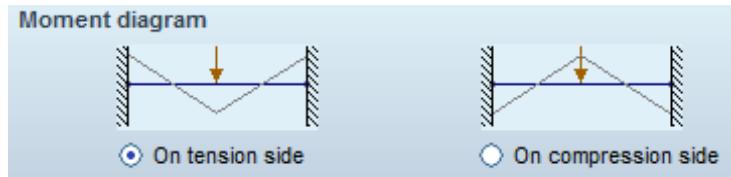
```
enum ESettingsError = {  
    seteInvalidGravityDirection = -100001  
    seteInvalidGravityAcceleration = -100002  
}
```

*gravity direction vector is not valid
gravity acceleration is not valid*

Enumerated types

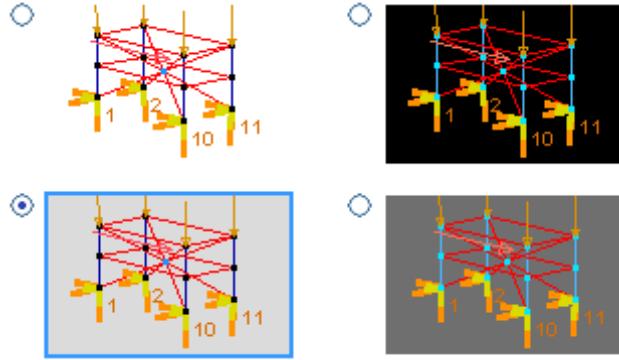
```
enum EPlaneToleranceType = {  
    ptRelativePerThousand = 0x01,  
    ptAbsolute = 0x02 }  
Plane tolerance type.
```

```
enum EMomentDiagramType = {  
    mdtOnTensionSide = 0x01,  
    mdtOnCompressionSide = 0x02 }  
Moment diagram type.
```



```
enum ETempFolderType = {  
    tftModel = 0x01,  
    tftSystem = 0x02,  
    tftCustom = 0x03 }  
Location of temp. files.
```

```
enum EGridType = {  
    gtGridLines = 0x01,  
    gtDotGrid = 0x02 }  
Type of grid on a background.
```

```
enum EBackgroundColour = {  
    bcWhite = 0x00,  
    bcBlack = 0x01,  
    bcLightGrey = 0x02,  
    bcDarkGrey = 0x03 }  

```

These options correspond to Settings/Preferences/Colors in AxisVM menu.

enum	EGeometryUnitType = { gut_Geom_Distance = 0x00, gut_Geom_Angle = 0x01, gut_Geom_Struct_size = 0x02 }	<i>Distance units</i> <i>Angular units</i> <i>Structural units</i>
enum	ECrossSectionUnitType = { csut_Size = 0x00, csut_Area = 0x01, csut_Static_moment = 0x02 , csut_Area_Moment_Inertia = 0x03, csut_Warping_constant = 0x04 }	<i>Cross-section's dimension units</i> <i>Cross-section's area units</i> <i>Cross-section's static moment units</i> <i>Cross-section's moment inertia units</i> <i>Cross-section's warping constant units</i>
enum	EMaterialUnitType = { mut_Young_modulus = 0x00, mut_Mass = 0x01, mut_Limit_stress = 0x02 , mut_Limit_strain = 0x03 }	<i>Material's Young modulus units</i> <i>Material's mass units</i> <i>Material's limit stress units</i> <i>Material's limit strain units</i>
enum	EPropertiesUnitType = { put_Beam_length = 0x00, put_Thickness = 0x01, put_Surface = 0x02 , put_Volume = 0x03, put_Mass = 0x04, put_put_Mass_per_length = 0x05, put_Gap_opening = 0x06 }	<i>Beam length units</i> <i>Thickness units</i> <i>Surface area units</i> <i>Volume units</i> <i>Mass units</i> <i>Mass per length units</i> <i>Gap width units</i>
enum	EStiffnessUnitType = { sut_Translational = 0x00, sut_Rotational = 0x01, sut_Line_translational = 0x02 , sut_Line_rotational = 0x03, sut_Surface = 0x04 }	<i>Translational stiffness units</i> <i>Rotational stiffness units</i> <i>Line translational stiffness units</i> <i>line rotational stiffness units</i> <i>Surface stiffness units</i>
enum	ELoadsUnitType = { lut_Force = 0x00, lut_Moment = 0x01, lut_Line_force = 0x02 , lut_Line_force_moment = 0x03, lut_Surface_force = 0x04, lut_Temperature = 0x05, lut_Temperature_variation = 0x06, lut_Design_fire_load_density = 0x07, lut_Specific_heat = 0x08 , lut_Section_factor = 0x09, lut_Fire_duration = 0x0A }	<i>Force units</i> <i>moment units</i> <i>Linear force units</i> <i>Linear moment units</i> <i>Surface force units</i> <i>Temperature units</i> <i>Temperature change units</i> <i>Fire intensity units</i> <i>Specific heat units</i> <i>Section factor units</i> <i>Fire duration units</i>

enum	ELoadsUnitType = { lut_Force = 0x00, lut_Moment = 0x01, lut_Line_force = 0x02 , lut_Line_force_moment = 0x03, lut_Surface_force = 0x04, lut_Temperature = 0x05, lut_Temperature_variation = 0x06, lut_Design_fire_load_density = 0x07, lut_Specific_heat = 0x08 , lut_Section_factor = 0x09, lut_Fire_duration = 0x0A }	<i>Force units</i> <i>moment units</i> <i>Linear force units</i> <i>Linear moment units</i> <i>Surface force units</i> <i>Temperature units</i> <i>Temperature change units</i> <i>Fire intensity units</i> <i>Specific heat units</i> <i>Section factor units</i> <i>Fire duration units</i>
enum	EStaticUnitType = { sut_Displacement = 0x00, sut_Rotation = 0x01, sut_Force = 0x02 , sut_Moment = 0x03, sut_DistrForce = 0x04, sut_DistrMoment = 0x05, sut_DistrSurfaceForce = 0x06, sut_Stress = 0x07}	<i>Displacement result units</i> <i>Rotation result units</i> <i>Force result units</i> <i>Moment result units</i> <i>Distributed force result units</i> <i>Distributed moment result units</i> <i>Distributed surface force result units</i> <i>Stress result units</i>
enum	ERCDesignUnitType = { rcdut_RebarDia = 0x00, rcdut_RebarDistance = 0x01, rcdut_ReinfArea = 0x02 , rcdut_ShearReinfArea = 0x03, rcdut_Cracking = 0x04, rcdut_Eccentricity = 0x05 }	<i>Rebar diameter units</i> <i>Rebar distance (spacing) units</i> <i>Reinforcement area units</i> <i>Shear reinforcement area units</i> <i>Crack width units</i> <i>Eccentricity units</i>
enum	ESteelDesignUnitType = { sdut_Buckling_factor = 0x00, sdut_Check_components = 0x01 }	<i>Steel buckling factor units</i> <i>Steel design component units</i>
enum	ETimberDesignUnitType = { tdut_Check_components = 0x00 }	<i>Timber design component units</i>
enum	EDimensioningUnitType = { dut_Dim_Distance = 0x00, dut_Dim_Angle = 0x01, dut_Level_symobol = 0x02 , dut_Graphics_size = 0x03 }	<i>Distance units</i> <i>Angular units</i> <i>Storey level units</i> <i>Graphics size units</i>

Records / structures

	RPlaneTolerance = (
<u>EPlaneToleranceType</u>	double	ptType Value <i>plane tolerance type (ptType = ptRelativePerThousand) plane tolerance is defined in per thousands of the biggest extension of the domain polygon</i>

```

        )
    )

RGridOptions = (
EBoolean          if lbTrue then grid is visible
double           Spacing of grids in global x direction
double           Spacing of grids in global y direction
double           Spacing of grids in global z direction
EGridType        Type of grid on a background.
)

RCursorSnap = (
EBoolean          if lbTrue then mouse snap is enabled
double           Cusor snaping in global x direction
double           Cusor snaping in global y direction
double           Cusor snaping in global z direction
CtrlX           Fine cusor snapingwhile holding Ctrl key
)

REditingOptions = (
double          Constraint angle delta alpha
double          Constraint angle custom alpha
double          Editing tolerance
ConstAngle_DeltaAlpha
ConstAngle_CustomAlpha
EditingToler
    Constraint Angle
        Δα [°] = 
        Custom α [°] = 
    Editing Tolerance
        δ [m] = 
)
These options correspond to Options/Editing in AxisVM menu.
)

RUnitParameters = (
long            AvailableUnits array index of unit used as base for conversion
long            AvailableUnits array index of unit used as current
long            number of decimal places, -1 if exponential
double           Multiplier used for conversion
ConversionBaseID
UsedID
DecimalPlaces
Multiplier
)

```

Functions

-
- long **GetCursorSnap** ([*i/o*] **RGridOptions** *Value*)
Value All cursor snap options
Get the cursor snap options. If unsuccessful, returns an error code ([errDatabaseNotReady](#)).
-
- long **GetGravity** ([*i/o*] **RPoint3d** *Direction*, [*out*] double *Acceleration*)
Direction Gravity direction
Acceleration Gravity acceleration
Get the direction and acceleration of gravity. If unsuccessful, returns an error code ([errDatabaseNotReady](#)).
-
- long **GetGridOptions** ([*i/o*] **RGridOptions** *Value*)
Value All grid options
Get the grid options. If unsuccessful, returns an error code ([errDatabaseNotReady](#)).
-
- long **GetEditingOptions** ([*i/o*] **REditingOptions** *Value*)
Value All editing options
Get the editing options. If unsuccessful, returns an error code ([errDatabaseNotReady](#)).

long	GetPlaneTolerance ([i/o] RPlaneTolerance Value)
	Value <i>plane tolerance</i>
	<i>Get the plane tolerance. If unsuccessful, returns an error code (errDatabaseNotReady).</i>
long	GetTempFolderType ([out] ETempFolderType TempFolderType, [out] BSTR Path)
	TempFolderType <i>Type of temp. file folder</i>
	Path <i>Full path of temp. file folder</i>
	<i>Get the temp. folder. If unsuccessful, returns an error code (errDatabaseNotReady).</i>
long	GetUnitParams_Geometry ([in] EGeometryUnitType UnitType, [in] EBoolean MathText, [in] EBoolean Default, [out] SAFEARRAY(BSTR)* AvailableUnits, [i/o] RUnitParameters UnitParameters)
	UnitType <i>Type (category) of the units</i>
	MathText <i>Use MathText format for unit string</i>
	Default <i>If lbTrue, then default units are returned, otherwise user units</i>
	AvailableUnits <i>String array with units</i>
	UnitParameters <i>All unit parameters as output</i>
	<i>Get units for geometry. If unsuccessful, returns an error code (errDatabaseNotReady).</i>
long	GetUnitParams_CrossSection ([in] ECrossSectionUnitType UnitType, [in] EBoolean MathText, [in] EBoolean Default, [out] SAFEARRAY(BSTR)* AvailableUnits, [i/o] RUnitParameters UnitParameters)
	UnitType <i>Type (category) of the units</i>
	MathText <i>Use MathText format for unit string</i>
	Default <i>If lbTrue, then default units are returned, otherwise user units</i>
	AvailableUnits <i>String array with units</i>
	UnitParameters <i>All unit parameters as output</i>
	<i>Get units for geometry. If unsuccessful, returns an error code (errDatabaseNotReady).</i>
long	GetUnitParams_Material ([in] EMaterialUnitType UnitType, [in] EBoolean MathText, [in] EBoolean Default, [out] SAFEARRAY(BSTR)* AvailableUnits, [i/o] RUnitParameters UnitParameters)
	UnitType <i>Type (category) of the units</i>
	MathText <i>Use MathText format for unit string</i>
	Default <i>If lbTrue, then default units are returned, otherwise user units</i>
	AvailableUnits <i>String array with units</i>
	UnitParameters <i>All unit parameters as output</i>
	<i>Get units for geometry. If unsuccessful, returns an error code (errDatabaseNotReady).</i>
long	GetUnitParams_Properties ([in] EPropertiesUnitType UnitType, [in] EBoolean MathText, [in] EBoolean Default, [out] SAFEARRAY(BSTR)* AvailableUnits, [i/o] RUnitParameters UnitParameters)
	UnitType <i>Type (category) of the units</i>
	MathText <i>Use MathText format for unit string</i>
	Default <i>If lbTrue, then default units are returned, otherwise user units</i>
	AvailableUnits <i>String array with units</i>
	UnitParameters <i>All unit parameters as output</i>
	<i>Get units for geometry. If unsuccessful, returns an error code (errDatabaseNotReady).</i>

long	GetUnitParams_Stiffness ([in] EStiffnessUnitType UnitType , [in] ELongBoolean MathText , [in] ELongBoolean Default , [out] SAFEARRAY(BSTR)* AvailableUnits , [i/o] RUnitParameters UnitParameters)
	<p>UnitType Type (category) of the units</p> <p>MathText Use MathText format for unit string</p> <p>Default If <i>IbTrue</i>, then default units are returned, otherwise user units</p> <p>AvailableUnits String array with units</p> <p>UnitParameters All unit parameters as output</p>
	<i>Get units for geometry. If unsuccessful, returns an error code (errDatabaseNotReady).</i>
long	GetUnitParams_Loads ([in] ELoadsUnitType UnitType , [in] ELongBoolean MathText , [in] ELongBoolean Default , [out] SAFEARRAY(BSTR)* AvailableUnits , [i/o] RUnitParameters UnitParameters)
	<p>UnitType Type (category) of the units</p> <p>MathText Use MathText format for unit string</p> <p>Default If <i>IbTrue</i>, then default units are returned, otherwise user units</p> <p>AvailableUnits String array with units</p> <p>UnitParameters All unit parameters as output</p>
	<i>Get units for geometry. If unsuccessful, returns an error code (errDatabaseNotReady).</i>
long	GetUnitParams_Static ([in] EStaticUnitType UnitType , [in] ELongBoolean MathText , [in] ELongBoolean Default , [out] SAFEARRAY(BSTR)* AvailableUnits , [i/o] RUnitParameters UnitParameters)
	<p>UnitType Type (category) of the units</p> <p>MathText Use MathText format for unit string</p> <p>Default If <i>IbTrue</i>, then default units are returned, otherwise user units</p> <p>AvailableUnits String array with units</p> <p>UnitParameters All unit parameters as output</p>
	<i>Get units for geometry. If unsuccessful, returns an error code (errDatabaseNotReady).</i>
long	GetUnitParams_RC_design ([in] ERCDesignUnitType UnitType , [in] ELongBoolean MathText , [in] ELongBoolean Default , [out] SAFEARRAY(BSTR)* AvailableUnits , [i/o] RUnitParameters UnitParameters)
	<p>UnitType Type (category) of the units</p> <p>MathText Use MathText format for unit string</p> <p>Default If <i>IbTrue</i>, then default units are returned, otherwise user units</p> <p>AvailableUnits String array with units</p> <p>UnitParameters All unit parameters as output</p>
	<i>Get units for geometry. If unsuccessful, returns an error code (errDatabaseNotReady).</i>
long	GetUnitParams_Steel_design ([in] ESteelDesignUnitType UnitType , [in] ELongBoolean MathText , [in] ELongBoolean Default , [out] SAFEARRAY(BSTR)* AvailableUnits , [i/o] RUnitParameters UnitParameters)
	<p>UnitType Type (category) of the units</p> <p>MathText Use MathText format for unit string</p> <p>Default If <i>IbTrue</i>, then default units are returned, otherwise user units</p> <p>AvailableUnits String array with units</p> <p>UnitParameters All unit parameters as output</p>
	<i>Get units for geometry. If unsuccessful, returns an error code (errDatabaseNotReady).</i>

long	GetUnitParams_Timber_design ([in] ETimberDesignUnitType UnitType , [in] EBoolean MathText , [in] EBoolean Default , [out] SAFEARRAY(BSTR)* AvailableUnits , [i/o] RUnitParameters UnitParameters)
	<p>UnitType Type (category) of the units</p> <p>MathText Use MathText format for unit string</p> <p>Default If <i>lbTrue</i>, then default units are returned, otherwise user units</p> <p>AvailableUnits String array with units</p> <p>UnitParameters All unit parameters as output</p> <p>Get units for geometry. If unsuccessful, returns an error code (errDatabaseNotReady).</p>
long	GetUnitParams_Dimensioning ([in] EDimensioningUnitType UnitType , [in] EBoolean MathText , [in] EBoolean Default , [out] SAFEARRAY(BSTR)* AvailableUnits , [i/o] RUnitParameters UnitParameters)
	<p>UnitType Type (category) of the units</p> <p>MathText Use MathText format for unit string</p> <p>Default If <i>lbTrue</i>, then default units are returned, otherwise user units</p> <p>AvailableUnits String array with units</p> <p>UnitParameters All unit parameters as output</p> <p>Get units for geometry. If unsuccessful, returns an error code (errDatabaseNotReady).</p>
long	SetEditingOptions ([i/o] REditingOptions Value)
	<p>Value editing options</p> <p>Set the editing options. If unsuccessful, returns an error code (errDatabaseNotReady).</p>
long	SetCursorSnap ([i/o] RGridOptions Value)
	<p>Value All cursor snap options</p> <p>Set the cursor snap options. If unsuccessful, returns an error code (errDatabaseNotReady).</p>
long	SetGravity ([i/o] RPoint3d Direction , [out] double Acceleration)
	<p>Direction Gravity direction</p> <p>Acceleration Gravity acceleration</p> <p>Set the direction and acceleration of gravity. If unsuccessful, returns an error code (setINVALIDGravityAcceleration, setINVALIDGravityDirection, errDatabaseNotReady).</p>
long	SetGridOptions ([i/o] RGridOptions Value)
	<p>Value All grid options</p> <p>Set the grid options. If unsuccessful, returns an error code (errDatabaseNotReady).</p>
long	SetPlaneTolerance ([i/o] RPlaneTolerance Value)
	<p>Value plane tolerance</p> <p>Set the plane tolerance. If unsuccessful, returns an error code (errDatabaseNotReady).</p>
long	SetTempFolderType ([in] ETempFolderType TempFolderType , [in] BSTR Path)
	<p>TempFolderType Type of temp. file folder</p> <p>Path Full path of temp. file folder</p> <p>Set the temp. folder. If unsuccessful, returns an error code (errDatabaseNotReady).</p>
EBoolean	EnvironmentClassIsValid ([in] EEvironmentClass EnvironmentClass)
	<p>EnvironmentClass environment class</p> <p>Determines whether <i>EnvironmentClass</i> is supported by the used national design code or not.</p>

Properties

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

<u>EBackgroundColour</u>	double	BackgroundColour • Get or set the colour of the background
		CrossSectionEditingTolerance • Get or set the editing tolerance in the cross-section editor.
	double	Editing Tolerance • Get or set the editing tolerance.
<u>ELongBoolean</u>		FixedMesh • Get or set whether meshes are created and removed automatically (True). If False, meshes remain editable.
<u>EMomentDiagramType</u>		MomentDiagramType • Get or set the type of the moment diagram.
<u>ENationalDesignCode</u>		NationalDesignCode • Get or set the current national design code.
<u>ELanguage</u>		Program Language • Get or set the used in dialog boxes, menus, etc.
<u>ELanguage</u>	double	ReportLanguage • Get or set the used in documentation.
		StiffnessReductionColumns_A • Get or set axial stiffness reduction globally for columns. Default value is 1. Used only for atLinearStatic and atLinearVibration analysis types .
	double	StiffnessReductionColumns_I • Get or set flexural stiffness reduction globally for columns. Default value is 1. Used only for atLinearStatic and atLinearVibration analysis types .
	double	StiffnessReductionBeams_A • Get or set axial stiffness reduction globally for beams. Default value is 1. Used only for atLinearStatic and atLinearVibration analysis types .
	double	StiffnessReductionBeams_I • Get or set flexural stiffness reduction globally for beams. Default value is 1. Used only for atLinearStatic and atLinearVibration analysis types .
	double	StiffnessReductionOtherMembers_A • Get or set axial stiffness reduction globally for line members except columns and beams . Default value is 1. Used only for atLinearStatic and atLinearVibration analysis types .
	double	StiffnessReductionOtherMembers_I • Get or set flexural stiffness reduction globally for line members except columns and beams. Default value is 1. Used only for atLinearStatic and atLinearVibration analysis types .
	double	StiffnessReductionWalls • Get or set stiffness reduction globally for walls. Default value is 1. Used only for atLinearStatic and atLinearVibration analysis types .
	double	StiffnessReductionSlabs • Get or set stiffness reduction globally for slabs. Default value is 1. Used only for atLinearStatic and atLinearVibration analysis types .
	double	StiffnessReductionOtherDomains • Get or set stiffness reduction globally for other domain elements except walls and slabs. Default value is 1. Used only for atLinearStatic and atLinearVibration analysis types .

IAxisVMSeismicStoreys

Seismic storeys used for generating seismic loads.

Enumerated types

```
enum ESeismicSensitivityResultsError = {
    ssreOK = 0x00,           results are OK
    ssreEmptyStorey = 0x01   empty storey in the model
    ssreInclinedElementExists =
        0x02                 inclined elements in the model
    ssreTorsionResultsMissing =
        0x03                 no torsion results
} Error type of seismic sensitivity results
```

Records / structures

```
RSeismicSensitivityResults = (
    ELongBoolean ResultsValidX      results valid for x direction
    ELongBoolean ResultsValidY      results valid for y direction
    double       ThetaMax_x         interstorey drift sensitivity coefficient in x direction
    double       ThetaMax_y         interstorey drift sensitivity coefficient in y direction
    double       Ptot                total gravity load at and above storey
    double       Vtot_x              total seismic storey shear in x direction
    double       Vtot_y              total seismic storey shear in y direction
    double       d_max_x            max. design interstorey drift in x direction
    double       d_max_y            max. design interstorey drift in y direction
    double       S_x                coordinate of shear (torsion) centre in x direction
    double       S_y                coordinate of shear (torsion) centre in y direction
    double       Gm_x               coordinate of centre of gravity in x direction
    double       Gm_y               coordinate of centre of gravity in y direction
    double       M_x                Mass in x direction
    double       M_y                Mass in y direction
    double       M_z                Mass in z direction
    double       Imz                Mass inertia about z axis relative to centre of gravity
    ESeismicSensitivityResultsError or
)
```

Functions

ELongBoolean **AutoSearch**

Search for stories in the model. Returns *lbTrue* if auto search successful, otherwise *lbFalse*.

long **Add ([in] double z, [in] BSTR Name)**

z Z coordinate of the new seismic storey

Name name of the new seismic storey

Adds a new seismic storey. If successful, returns number of seismic stories in the model, otherwise an error code ([errDatabaseNotReady](#)).

long **Clear**

Deletes all seismic storeys. If successful, returns number of storey, otherwise an error code ([errDatabaseNotReady](#)).

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

index storey index

Storey index starts from top (highest z value) with number 1. If successful, returns storey index of deleted storey, otherwise an error code ([errDatabaseNotReady](#) or [errIndexOutOfBounds](#)).

IAxisVMSpectrum

Seismic spectrum interface for earthquake design.

Error codes

enum	ESpectrumError = {	
	seLoadingFailed = -100001	loading a spectrum from file failed
	seSavingFailed = -100002	saving a spectrum to a file failed
	seSpectrumDataNotAvailable = -100003	Spectrum data are not available
	seSpectrumDataNotParametric = -100004	Spectrum data are not parametric
	seInvalidNationalDesignCode = -100005	Design code is invalid
	seFunctionPointsEmpty = -100006	array with function points is empty
	seSpectrumNotValid = -100007	spectrum is invalid
	seNonParametricSpectrumIsNotAllowed = -100008	only parametric spectrum is allowed
	}	

Records / structures

RSpectrumData_EC = (

<u>ESoilClass</u>	SubsoilClass	See seismic calculation based on Euro code 8 in AxisVM manual
double	agr	See a_{gr} in seismic calculation based on Euro code 8 in AxisVM manual
double	s	See S in seismic calculation based on Euro code 8 in AxisVM manual
double	Beta0	See β in seismic calculation based on Euro code 8 in AxisVM manual
double	TB	See T_B in seismic calculation based on Euro code 8 in AxisVM manual
double	TC	See T_c in seismic calculation based on Euro code 8 in AxisVM manual
double	TD	See T_d in seismic calculation based on Euro code 8 in AxisVM manual
double	gammal	See importance factor γ_l in seismic calculation based on Euro code 8 in AxisVM manual
double	qX	See q in seismic calculation based on Euro code 8 in AxisVM manual (x direction)
double	qY	See q in seismic calculation based on Euro code 8 in AxisVM manual (y direction)
)	

RSpectrumData_ECHU = (

<u>ESoilClass</u>	SubsoilClass	See seismic calculation based on Euro code 8 in AxisVM manual
double	agr	See a_{gr} in seismic calculation based on Euro code 8 in AxisVM manual
double	s	See S in seismic calculation based on Euro code 8 in AxisVM manual
double	Beta0	See β in seismic calculation based on Euro code 8 in AxisVM manual
double	TB	See T_B in seismic calculation based on Euro code 8 in AxisVM manual
double	TC	See T_c in seismic calculation based on Euro code 8 in AxisVM manual
double	TD	See T_d in seismic calculation based on Euro code 8 in AxisVM manual
double	gammal	See importance factor γ_l in seismic calculation based on Euro code 8 in AxisVM manual
double	qX	See q in seismic calculation based on Euro code 8 in AxisVM manual (x direction)
double	qY	See q in seismic calculation based on Euro code 8 in AxisVM manual (y direction)
<u>ELongBoolean</u>	LocalSpectrum	local spectrum according to the Hungarian Chamber of Engineers Local spectra manual
double	F0	F_0 according to the Hungarian Chamber of Engineers Local spectra manual
)	

RSpectrumData_ECNL = (

double	agr	See Dutch NPR 9998:2015
double	p	See Dutch NPR 9998:2015
double	TB	See Dutch NPR 9998:2015
double	TC	See Dutch NPR 9998:2015
double	TD	See Dutch NPR 9998:2015
double	gammal	See importance factor γ_l in seismic calculation based on Euro code 8 in AxisVM manual
double	qX	See q in seismic calculation based on Euro code 8 in AxisVM manual (x direction)
double	qY	See q in seismic calculation based on Euro code 8 in AxisVM manual (y direction)
)	

RSpectrumData_ITA = (

<u>ESoilClass</u>	SubsoilClass	See seismic calculation based on Euro code 8 in AxisVM manual
double	agr	See a_{gr} in seismic calculation based on Euro code 8 in AxisVM manual
double	F0	See F_0 in Italian national code
double	Tsc	See T_c^* in Italian national code
<u>ETopographicCategory</u>	TopographicCategory	Topographic category
double	qx	See q in seismic calculation based on Italian national code in AxisVM manual (Behaviour factor in x direction)
double	qy	See q in seismic calculation based on Italian national code in AxisVM manual (Behaviour factor in y direction)
)	

RSpectrumData_SIA = (

<u>ESoilClass</u>	SubsoilClass	Subsoil class
double	agr	Gravity acceleration [m/s^2]

double	S	<i>see seismic calculation based on SIA in AxisVM manual</i>
double	TB	<i>see seismic calculation based on SIA in AxisVM manual</i>
double	TC	<i>see seismic calculation based on SIA in AxisVM manual</i>
double	TD	<i>see seismic calculation based on SIA in AxisVM manual</i>
double	gammal	<i>Importance factor, see seismic calculation based on SIA in AxisVM manual</i>
double	qx	<i>See q in seismic calculation based on SIA in AxisVM manual (Behaviour factor in x direction)</i>
double	qy	<i>See q in seismic calculation based on SIA in AxisVM manual (Behaviour factor in y direction)</i>
)	
RSpectrumData_STAS = (
ESoilClass SubsoilClass		
double	agr	<i>Subsoil class</i>
double	beta0	<i>Gravity acceleration [m/s²]</i>
double	TB	<i>see seismic calculation based on STAS in AxisVM manual</i>
double	TC	<i>see seismic calculation based on STAS in AxisVM manual</i>
double	TD	<i>see seismic calculation based on STAS in AxisVM manual</i>
double	gammal	<i>Importance factor, see seismic calculation based on STAS in AxisVM manual</i>
double	qx	<i>Behaviour factor in x direction, see seismic calculation in AxisVM manual</i>
double	qy	<i>Behaviour factor in y direction, see seismic calculation in AxisVM manual</i>
)	

```

RspectrumData_DIN = (
  ESubsoilClass SubsoilClass Subsoil class
  double agr Gravity acceleration [m/s2]
  double S see seismic calculation based on DIN in AxisVM manual
  double beta0 see seismic calculation based on DIN in AxisVM manual
  double TB see seismic calculation based on DIN in AxisVM manual
  double TC see seismic calculation based on DIN in AxisVM manual
  double TD see seismic calculation based on DIN in AxisVM manual
  double gammal Importance factor, see seismic calculation based on STAS in AxisVM manual
  double qx Behaviour factor in x direction, see seismic calculation based on DIN in AxisVM
               manual
  double qy Behaviour factor in y direction, see seismic calculation based on DIN in AxisVM
               manual
)

RspectrumData = (
  Warning! This record has become obsolete, it was superseded by
  RspectrumData_V153

  RspectrumData_EC SpectrumData_EC Spectrum parameters according to EC
  RspectrumData_ITA SpectrumData_ITA Spectrum parameters according to Italian national code
  RspectrumData_SIA SpectrumData_SIA Spectrum parameters according to Swiss national code
  RspectrumData_STAS SpectrumData_STAS Spectrum parameters according to Romanian national code
  RspectrumData_DIN SpectrumData_DIN Spectrum parameters according to German national code
)

```

RspectrumData_V153 = (

```

  SpectrumData_EC Spectrum parameters according to EC
  SpectrumData_ITA Spectrum parameters according to Italian national code
  SpectrumData_SIA Spectrum parameters according to Swiss national code
  SpectrumData_ECHU Spectrum parameters according to hungarian EC
  SpectrumData_ECNL Spectrum parameters according to dutch EC
  SpectrumData_STAS Spectrum parameters according to Romanian national code
  SpectrumData_DIN Spectrum parameters according to German national code
)

```

The field corresponding to the active national code will be taken into account.

Functions

long	Disable ()	Disables the spectrum. Only applicable in the case of a vertical spectrum. If successful, returns 1. If called for not a vertical spectrum (horizontal, pushover), errNotImplemented is returned. The possible error codes are (errDatabaseNotReady , errNotImplemented)
long	GetPoints ([out] SAFEARRAY(RPoint2d) * FunctionPoints)	FunctionPoints x and y coordinates of the function, x = coord1, y = coord2 Get points of the spectrum. If successful, returns number of points, otherwise returns an error code (errDatabaseNotReady).
long	GetSpectrumData ([i/o] RspectrumData SpectrumData)	Warning! This function has become obsolete, it was superseded by GetSpectrumData_V153 SpectrumData Parametric spectrum data Reads seismic data. If successful, returns 1, otherwise returns an error code (seSpectrumDataNotAvailable , seInvalidNationalDesignCode , seSpectrumDataNotParametric , errDatabaseNotReady).
long	GetSpectrumData_V153 ([i/o] RspectrumData_V153 SpectrumData)	SpectrumData Parametric spectrum data Reads seismic data. If successful, returns 1, otherwise returns an error code (seSpectrumDataNotAvailable , seInvalidNationalDesignCode , seSpectrumDataNotParametric , errDatabaseNotReady).
long	LoadFromFile ([in] BSTR FileName)	FileName Name of the file to load the spectrum from

Loads a spectrum from a file. If successful, returns 1, otherwise returns an error code ([seLoadingFailed](#), [errDatabaseNotReady](#)).

To recalculate the seismic loads call the IAxisVMLoads.[CreateStandardSeismicLoads](#) function after loading a spectrum.

long **SaveToFile ([in] BSTR FileName)**

FileName Name of the file to save the spectrum to

Saves a spectrum to a file. If successful, returns 1, otherwise returns an error code ([seSavingFailed](#), [errDatabaseNotReady](#)).

long **SetPoints ([in] SAFEARRAY([RPoint2d](#)) * FunctionPoints)**

FunctionPoints x and y coordinates of the function, x = coord1, y = coord2

Set points of the spectrum and spectrum to non-parametric. If successful, returns number of points, otherwise returns an error code ([errDatabaseNotReady](#), [seFunctionPointsEmpty](#)).

long **SetPoints_vb ([i/o] SAFEARRAY([RPoint2d](#)) * FunctionPoints)**

FunctionPoints x and y coordinates of the function, x = coord1, y = coord2

Visual basic compatible version of SetPoints.

long **SetSpectrumData ([i/o] [RSpectrumData](#) SpectrumData)**

Warning! This function has become obsolete, it was superseded by [SetSpectrumData_V153](#)

SpectrumData Parametric spectrum data

Set seismic data. If successful, returns 1, otherwise returns an error code ([errDatabaseNotReady](#)).

long **SetSpectrumData_V153 ([i/o] [RSpectrumData_V153](#) SpectrumData)**

SpectrumData Parametric spectrum data

Set seismic data. If successful, returns 1, otherwise returns an error code ([errDatabaseNotReady](#)).

Properties

[ELongBoolean](#)

Disabled Returns True if the spectrum is disabled (Read only). Valid only for vertical spectrums. Use SetPoints or SetSpectrumData_V153 to make the vertical spectrum enabled, and Disable to make the vertical spectrum disabled.

[ELongBoolean](#)

Parametric Returns True if spectrum was defined by parameters (Read only). Use SetPoints to change to non-parametric, SetSpectrumData_V153 to change to parametric.

IAxisVMSpringParams

Spring characteristics library. As of now, it is applicable only for nodal supports.

Error codes

enum	ESpringParError = {	
	speNLEInconsistency = -100001	<i>Non linear elastic spring with contradictory parameters</i>
	speNegativeValueMustBePositive = -100002	<i>A negative value is present in a field where only positive values are valid</i>
	speNLPInconsistency = -100003	<i>Non linear plastic spring with contradictory parameters</i>

Enumerated types

enum	ESpringParType = {	<i>Spring characteristics for nodal supports (but not for seismic isolators)</i>
	sptNodal = 0	<i>Spring characteristics for nodal seismic isolators</i>
	sptIsolator = 1}	
enum	ESpringParNNType = {	<i>Linear spring characteristics</i>
	spnntLinear = 0	<i>Non linear elastic spring characteristics</i>
	spnntNonLinearElastic = 1	<i>Non linear plastic spring characteristics</i>
enum	ESpringParDOFType = {	
	spdofTranslation = 0	<i>Translational spring characteristics</i>
	spdofRotation = 1}	<i>Rotational spring characteristics</i>
enum	ESpringParDampingType = {	
	spdtKelvin = 0	<i>Type of dynamic damping : Kelvin-Voigt</i>
	spdtMaxwell = 1}	<i>Type of dynamic damping : Maxwell</i>
enum	ESpringParNonLinearity = {	
	spnlTensionAndCompression = 0	<i>Type of nonlinearity : both tension and compression</i>
	spnlTensionOnly = 1	<i>Type of nonlinearity : only for tension</i>
	spnlCompressionOnly = 2	<i>Type of nonlinearity : only for compression</i>
enum	ESpringParNLDefType = {	
	spnldtByParam = 0	<i>Non linear behaviour is defined through numeric parameters</i>
	spnldtByFunction = 1}	<i>Non linear behaviour is defined through a function</i>
enum	ESpringParHardeningRule = {	
	sphrlIsotropic = 0	<i>Hardening rule for plastic spring model : isotropic</i>
	sphrkKinematic = 1}	<i>Hardening rule for plastic spring model : Kinematic</i>
enum	ESpringParMatrixType = {	
	spmtTangentMatrix = 0	<i>Stiffness for plastic spring model : tangent stiffness</i>
	spmtInitialMatrix = 1}	<i>Stiffness for plastic spring model : initial stiffness</i>
enum	ESpringParIsolatorType = {	
	spitRubber = 0	<i>Seismic isolator type : rubber bearing</i>
	spitSlider = 1	<i>Seismic isolator type : curved surface slider</i>
	spitCustom = 2}	<i>Seismic isolator type : custom</i>

Records / structures

	RSpringParam = (
ESpringParType	SpringType	<i>Spring characteristic type (nodal, seismic isolator, ...)</i>
ESpringParNNType	NNType	<i>Linear / non linear type of the spring characteristic</i>
ESpringParDOFType	DOFType	<i>Translational / rotational spring characteristic</i>
EBoolean double	NLESimplified	<i>User selection for non linear elastic case : simplified or not</i>
	K	<i>Initial stiffness [KN/m]</i>

	KVib	<i>Vibration stiffness [KN/m]</i>
<u>ESpringParDampingType</u>	DampingType	<i>Type of dynamic damping</i>
	C	<i>Dynamic damping [KN/m/s]</i>
<u>ESpringParNonLinearity</u>	NonLinearity	<i>Non linearity type : linear, non linear elastic, non linear plastic</i>
<u>ESpringParNLDefType</u>	NLDefType	<i>Non linear characteristics are defined by parameters or by a function</i>
	K_T	<i>Stiffness for tension [KN/m]</i>
	K_C	<i>Stiffness for compression [KN/m]</i>
EBoolean	ResistanceDef_T	<i>Resistance for tension is defined</i>
EBoolean	ResistanceDef_C	<i>Resistance for compression is defined</i>
double	TangentStiffness_T	<i>Stiffness for tension tangent [KN/m]</i>
double	TangentStiffness_C	<i>Stiffness for compression tangent [KN/m]</i>
double	Resistance_T	<i>Resistance for tension [KN]</i>
double	Resistance_C	<i>Resistance for compression [KN]</i>
EBoolean	ResistanceDef_T	<i>Resistance for tension is defined</i>
EBoolean	ResistanceDef_C	<i>Resistance for compression is defined</i>
<u>ESpringParHardeningRule</u>	HardeningRule	<i>Hardening rule for plastic spring model</i>
<u>ESpringParMatrixType</u>	MatrixType	<i>Type of stiffness for plastic spring model</i>
	C_T	<i>Damping for tension [KN/(m/s)]</i>
	C_C	<i>Damping for compression [KN/(m/s)]</i>
	VerticalStiffness	<i>Vertical stiffness for seismic isolator [KN/m]</i>
<u>ESpringParIsolatorType</u>	IsolatorType	<i>Seismic isolator type</i>
	K1	<i>Initial stiffness (only for rubber seismic isolator) [KN/m]</i>
	KT	<i>Tangent stiffness (only for rubber seismic isolator) [KN/m]</i>
	F1	<i>Resistance (only for rubber seismic isolator) [KN]</i>
	Mu	<i>Friction coefficient (only for slider seismic isolator) []</i>
	R	<i>Radius (only for slider seismic isolator) [m]</i>
	HorizontalStiffness	<i>Horizontal stiffness (only for custom seismic isolator) [KN/m]</i>
	Ksi	<i>Damping ratio (only for custom seismic isolator) []</i>

RSpringParam valid field combinations.

Only specific combination of fields are applicable at the same time.

SpringType = sptNodal : fields [NNType..C_C]

NNType = spnntLinear :
DOFType, K, KVib, DampingType, C

NNType = spnntNonLinearElastic :
DOFType, KVib, DampingType, NLESimplified, NonLinearity, NLDefType, K_T, K_C, ResistanceDef_T, ResistanceDef_C, TangentStiffness_T, TangentStiffness_C, Resistance_T, Resistance_C, C_T, C_C

NNType = spnntNonLinearPlastic :
DOFType, KVib, DampingType, NLESimplified, NonLinearity, NLDefType, K_T, K_C, ResistanceDef_T, ResistanceDef_C, TangentStiffness_T, TangentStiffness_C, Resistance_T, Resistance_C, C_T, C_C, HardeningRule, MatrixType

SpringType = sptIsolator : fields [VerticalStiffness..Ksi] :

IsolatorType = spItRubber
VerticalStiffness, K1, KT, F1

IsolatorType = spItSlider
VerticalStiffness, Mu, R

IsolatorType = spitSlider
VerticalStiffness, HorizontalStiffness, Ks

Functions

long **Add** ([in] BSTR **Name**, [in] [RSpringParam](#)* **Value**)

Name *Name of the spring characteristics (should be unique)*
Value *The RSpringParam record defining the spring characteristics.*
See [RSpringParam valid field combinations](#) for further details

Defines a new spring characteristic. If successful, returns the index of the new spring characteristic, otherwise an error code ([errDatabaseNotReady](#), [speNLEInconsistency](#), [speNLPIconsistency](#), [speNegativeValueMustBePositive](#)).

long **Clear**

Clears all spring characteristics (even the precreated default ones). If successful, returns a positive number, otherwise an error code ([errDatabaseNotReady](#)).

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

Index *Index of the spring characteristics*

Deletes a spring characteristic. Can delete even the precreated default ones. If successful, returns the index of the deleted spring characteristic, otherwise an error code ([errDatabaseNotReady](#), [errIndexOutOfBounds](#)).

long **IndexByName** ([in] BSTR **Name**)

Name *Name to be searched*

If successful, returns the index of the first spring characteristic with a matching name. If there is no such name, the result is 0. The possible error codes : ([errDatabaseNotReady](#)).

long **IndexOfUID** ([in] long **UID**)

UID *UID to be searched*

If successful, returns the index of the first spring characteristic with a matching UID. If there is no such item, the result is 0. The possible error codes : ([errDatabaseNotReady](#)).

Properties

long **Count** *Number of spring characteristics*

[AxisVMSpringParam](#) **Item** [long **Index**] *A spring characteristic by index*

long **MaxNameLength** *Maximum length of a spring characteristics name. If a longer string is used, the excess part will be truncated*

BSTR **Name** [long **Index**] *Name by index*

long **UID** [long **Index**] *UID by index*

IAxisVMSpringParam

A spring characteristics. As of now, it is applicable only for nodal supports.

Properties

[RSpringParam](#) **FullRec** *The properties of the spring characteristic. See [RSpringParam valid field combinations](#) for further details*

SAFEARRAY([RPoint2d](#)) **FunctionPoints** *If (SpringType = sptNodal) and (NNType in [spnntNonLinearElastic, spnntNonLinearPlastic]) and (NLDefType = spnlDtByFunction), the function points can be accessed through this property*

IAxisVMSteelDesignMembers

Interface used for defining steel design members

If property returning this interface is null (nil) then the extension module SD1 is not available.

Error codes

enum	ESteelDesignMemberError = {	
	sdmeCOMError = -100001	COM server error
	sdmeLineListIsEmpty = -100002	when LineIDs parameter of IAxisVMSteelDesignMembers.Add is empty
	sdmeInvalidNationalDesignCode = -100003	IAxisVMSteelDesignMembers does not support the selected design code
	sdmeNotConnectingLines = -100004	LineIDs of IAxisVMSteelDesignMembers.Add are not connected
	sdmeDesignParametersNotValidForUsedDesignCode = -100005 }	Design parameters are not valid for used design code

Enumerated types

enum	EDesignApproach = {	
	daClass = 0	<i>Depending on class of the section</i>
	daElastic = 1}	<i>Elastic design approach</i>
	<i>Approach used for steel design</i>	
enum	EMcrMethod = {	
	mcrmAuto = 0	<i>Mcr (Auto Mcr option in AxisVM) is automatically calculated based on FE analysis (C1,C2 and C3 are ignored)</i>
	mcrmC1Lopez = 1	<i>Used prior to version 13 release 2</i>
	mcrmC1C2C3User = 2	<i>C1 value calculated using Lopez's method</i>
	mcrmDutch = 3	<i>C1, C2 and C3 values must be set by user</i>
	mcrmDutchUser = 4	<i>C1 and C2 values are automatically calculated</i>
	mcrmAutoLS = 5	<i>C1 and C2 values must be set by user</i>
	mcrmUser = 6 }	<i>updated version of mcrmAuto method, considering ESteelLateralSupports</i>
	<i>What method will be used for calculation of C1, C2 and C3 parameters see the AxisVM manual and</i>	<i>Used from version 13 release 2 (Auto Mcr option in AxisVM)</i>
	<i>Appendix F.1.2 of ENV 1993-1-1.</i>	<i>Mcr is given by the user</i>
enum	EStiffeners = {	
	sNo = 0	<i>No stiffeners</i>
	sTransversal = 1}	<i>Transversal stiffeners</i>
	<i>Type of stiffener.</i>	
enum	ETorsion = {	
	tFree = 0	<i>Free to rotate about local x</i>
	tPartial = 1	<i>One end of beam fixed in torsion</i>
	tFixed = 2}	<i>Both ends of beam fixed in torsion</i>
	<i>Type of torsion.</i>	
enum	ESteelBucklingLengthMode = {	
	sblm_Factor= 0,	<i>buckling length given by factors</i>
	sblm_Length= 1,	<i>buckling length given by length</i>
	sblm_Auto= 2,	<i>automatic buckling length calculation</i>
	sblm_None = 3}	<i>none</i>
enum	ESteelCantileverFixedEnd = {	
	scfeStartNode = 0,	<i>start node of the element is fixed</i>
	scfeEndNode = 1}	<i>end node of the element is fixed</i>

```
enum ESteelLateralSupports = {
    ammAuto = 0,
    ammEstimatedFromKzKw = 1,
    ammForkSupports = 2,
    ammUserDefined = 3}
```

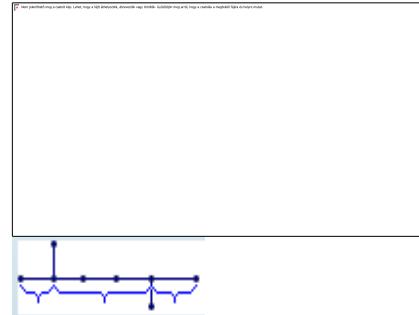
*automatic
estimated from kz, kw
fork supports
user defined (lateral supports must be defined)*

Records / structures

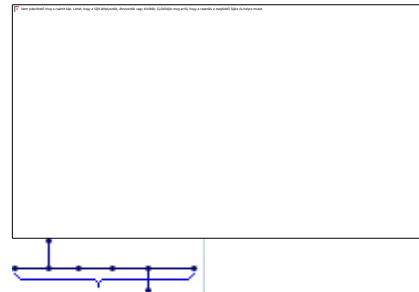
RSteelDesignParameters_EC_SIA_ITA = (

EBoolean BreakAtElements

*If True:
Break members at side at side element connections or nodal supports.*



*If False:
The member becomes only one structural element irrespective of other elements (members etc.) connecting to its nodes.*



double **a**

distance of trans. stiffeners (used only if Stiffeners = sTransversal)

double **akr**
double **C1**

*critical load parameter factor (not used)
steel design factor see steel design in Axis VM manual (depend on **McrMethod** and cross-section, see Calculation of Mcr in AxisVM manual)*

double **C2**

*steel design factor see steel design in Axis VM manual (depend on **McrMethod** and cross-section, see Calculation of Mcr in AxisVM manual)*

double **C3**

*steel design factor see steel design in Axis VM manual (depend on **McrMethod** and cross-section, see Calculation of Mcr in AxisVM manual)*

EDesignApproach **DesignApproach**
double **Ky**

*Design approach
buckling length factor about local y axis (used only if BucklingLengthModeY = sbIm_Factor)*

double **Kz**

buckling length factor about local z axis (used only if BucklingLengthModeZ = sbIm_Factor)

double **kt**

shear buckling coefficient (not used)

	double	Kw	warping constraint factor (see steel design in Axis VM manual; used only if <i>McrMethod</i> <> <i>mcrmAuto</i>)
<u>EMcrMethod</u> <u>EStiffeners</u>		McrMethod Stiffeners	Define how to determine <i>Mcr</i> type of stiffeners (used only if <i>WebShearBuckling</i> = <i>lbTrue</i>)
<u>ELongBoolean</u> <u>EBoolean</u>	long	SDP_Class YBraced ZBraced	section class (automatic if 0) True if braced Y direction True if braced Z direction
	double	Za	Relative load position in local z axis (see steel design parameters - load position in Axis VM manual)
	double	fse	load factor for seismic forces (default is 1 => no change); note: it was <i>ks</i> before.
<u>ELongBoolean</u> <u>ESteelCantileverFixedEnd</u> <u>ELongBoolean</u> <u>ELongBoolean</u> <u>ELongBoolean</u> <u>ESteelBucklingLengthMode</u>		Cantilever CantileverFixedEnd FlexuralBuckling LateralTorsionalBuckling WebShearBuckling BucklingLengthModeY	<i>cantilever</i> checkbox start or end node of cantilever is fixed flexural buckling lateral torsional buckling web shear buckling buckling length calculation mode related to y-y axis
<u>ESteelBucklingLengthMode</u>		BucklingLengthModeZ	buckling length calculation mode related to z-z axis
	double	Ly	buckling length related to y-y axis (used only if <i>BucklingLengthModeY</i> = <i>sblm_Length</i>)
	double	Lz	buckling length related to z-z axis (used only if <i>BucklingLengthModeZ</i> = <i>sblm_Length</i>)
<u>ELongBoolean</u>	double	ConsiderN Eta	consider the effect of normal force η factor for web shear buckling checks (used only if <i>Stiffeners</i> = <i>sTransversal</i>)
<u>ESteelLateralSupports</u>		LateralSupports	<i>lateral supports</i>
	double	Mcr	used defined <i>Mcr</i> (used only if <i>LateralSupports</i> = <i>ammUserDefined</i>)

Steel design parameters in accordance with Euro code, Swiss and Italian national codes. For more info, see steel design in Axis VM manual.

		RSteelDesignParameters_MSZ_STAS = (
<u>ELongBoolean</u>		BreakAtElements See here
double		nuy see <i>v_y</i> in steel design in Axis VM manual
double		nuz see <i>v_z</i> in steel design in Axis VM manual
double		nuw see <i>v_w</i> in steel design in Axis VM manual
double		d see steel design in Axis VM manual [m]
double		a distance of trans. stiffeners [m]
<u>EStiffeners</u>		Stiffeners type of stiffeners

Steel design parameters in accordance with Hungarian and Romanian national codes.

		RSteelDesignParameters_NEN = (
<u>ELongBoolean</u>		BreakAtElements See here
double		Kapy see steel design in Axis VM manual
double		Kapz see steel design in Axis VM manual
double		Y see steel design in Axis VM manual
double		L1F see steel design in Axis VM manual
double		L1A see steel design in Axis VM manual
double		IgF see steel design in Axis VM manual
double		IgA see steel design in Axis VM manual
double		ak see α_{cr} in steel design in Axis VM manual
double		Fytot see steel design in Axis VM manual [kN]
double		Fztot see steel design in Axis VM manual [kN]
<u>ELongBoolean</u>		YBraced see steel design in Axis VM manual
<u>ELongBoolean</u>		ZBraced see steel design in Axis VM manual
<u>ETorsion</u>		Torsion type of torsion

)

Steel design parameters in accordance with Dutch national code.

RSteelLTBSupport= (

double	AbsPos	<i>absolute position of the support</i>
double	Ecc	<i>eccentricity</i>
double	Ry	<i>lateral support stiffness in local y direction</i>
double	Rxx	<i>torsional stiffness about the member's axis</i>
double	Rzz	<i>torsional stiffness about the local z axis</i>
double	Rw	<i>warping stiffness</i>

)

RSteelDesignParameters = (

<u>RSteelDesignParametersMSZ_STAS</u>	MSZ_STAS	<i>design parameters according to MSZ and STAS national design code</i>
---	-----------------	---

<u>RSteelDesignParameters_EC_SIA_ITA</u>	EC_SIA_ITA	<i>design parameters according to EC,SIA and ITA national design code</i>
--	-------------------	---

<u>RSteelDesignParametersNEN</u>	NEN	<i>design parameters according to NEN national design code</i>
--	------------	--

)

Functions

long **Add ([in] SAFEARRAY(byte) DesignParameters, [in] SAFEARRAY(long)* LinelDs)**

DesignParameters Is actually an SAFEARRAY(xxxx_record), where xxxx_record can be RSteelDesignParameters_MSZ_STAS, RSteelDesignParameters_NEN, RSteelDesignParameters_EC_SIA_ITA depending on national code. (Safearray must have only one index and lower bound one)

LinelDs Index array (long) with linelDs

Defines a new steel design member.

If successful, returns the SteelDesignMember index of new SteelDesignMember otherwise an error code ([errDatabaseNotReady](#), [errIndexOutOfBoundsException](#), [sdmeLineListIsEmpty](#), [sdmeNotConnectingLines](#), [sdmeCOMError](#)).

long **Add_vb** (Visual Basic compatible function of Add)

long **Add2 ([i/o] RSteelDesignParameters SteelDesignParameters,**

[in] SAFEARRAY(long)* LinelDs)

SteelDesignParameters Parameter which contain all design parameters from all national design codes

LinelDs Index array (long) with linelDs

Defines a new steel design member.

If successful, returns the SteelDesignMember index of new SteelDesignMember otherwise an error code ([errDatabaseNotReady](#), [errIndexOutOfBoundsException](#), [sdmeLineListIsEmpty](#), [sdmeNotConnectingLines](#), [sdmeCOMError](#)).

long **Add2_vb** (Visual Basic compatible function of Add2)

long **Add3 ([i/o] SAFEARRAY(byte) SteelDesignParameters,**

[in] SAFEARRAY(long)* LinelDs)

SteelDesignParameters Parameters which contain all design parameters of used national design code, size must match the size of the used record (e.g. [RSteelDesignParameters_EC_SIA_ITA](#))

LinelDs Index array (long) with linelDs

Defines a new steel design member. If successful, returns the SteelDesignMember index of new SteelDesignMember otherwise an error code ([errDatabaseNotReady](#), [errIndexOutOfBoundsException](#), [sdmeLineListIsEmpty](#), [sdmeNotConnectingLines](#), [sdmeCOMError](#)).

Note: The SteelDesignParameters must be type casted into the appropriate load record type depending on the national design code!

long	Clear
<i>Removes all SteelDesignMembers. It returns the number of deleted SteelDesignMembers. If it returns a negative number, that is an error code (errDatabaseNotReady).</i>	
long	Delete ([in] long Index)
Index SteelDesignMember index ($0 < \text{Index} \leq \text{IAxisVMSteelDesignMembers.Count}$)	
<i>Deletes a SteelDesignMember. If successful, returns the SteelDesignMember index ($1 \leq \text{Index} \leq \text{Count}$), otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBoundsException).</i>	
long	DeleteSelected
<i>Returns number of deleted SteelDesignMembers, otherwise returns an error code (errDatabaseNotReady, errNotSupportedByNationalDesignCode).</i>	
long	GetDesignParameters ([in] long Index, [out] SAFEARRAY(byte)* DesignParameters)
Index SteelDesignMember index ($0 < \text{Index} \leq \text{IAxisVMSteelDesignMembers.Count}$)	
DesignParameters See IAxisVMSteelDesignMembers.Add	
<i>If successful, returns SteelDesignMember index otherwise an error code (errDatabaseNotReady, errIndexOutOfBoundsException, errNotSupportedByNationalDesignCode, sdmeInvalidNationalDesignCode)</i>	
long	GetDesignParameters2 ([in] long Index, [i/o] RSteelDesignParameters SteelDesignParameters)
Index SteelDesignMember index ($0 < \text{Index} \leq \text{IAxisVMSteelDesignMembers.Count}$)	
SteelDesignParameters Parameters which contain all design parameters from all national design codes	
<i>If successful, returns index otherwise an error code (errDatabaseNotReady, errIndexOutOfBoundsException, errNotSupportedByNationalDesignCode, sdmeInvalidNationalDesignCode)</i>	
long	GetDesignParameters3 ([in] long Index, [out] SAFEARRAY(byte) SteelDesignParameters)
Index SteelDesignMember index ($0 < \text{Index} \leq \text{IAxisVMSteelDesignMembers.Count}$)	
SteelDesignParameters Parameters which contain all design parameters of used national design code, size must match the size of the used record (e.g. RSteelDesignParameters EC SIA ITA)	
<i>If successful it returns index otherwise an error code (errDatabaseNotReady, errIndexOutOfBoundsException, errNotSupportedByNationalDesignCode, sdmeInvalidNationalDesignCode)</i>	
<i>Note: The SteelDesignParameters must be type casted into the appropriate load record type depending on the national design code!</i>	
long	GetLinelds ([in] long Index, [out] SAFEARRAY(long)* Linelds)
Index SteelDesignMember index ($0 < \text{Index} \leq \text{IAxisVMSteelDesignMembers.Count}$)	
Linelds Index array (long) with lineIDs	
<i>Returns number of lines in SteelDesignMember, otherwise returns an error code (errDatabaseNotReady, errNotSupportedByNationalDesignCode).</i>	
long	GetSelectedItemIds ([out] SAFEARRAY (long)* ItemIds)
ItemIds Index list of selected edge connections	
<i>Returns number of selected SteelDesignMembers, otherwise returns an error code (errDatabaseNotReady, errNotSupportedByNationalDesignCode).</i>	

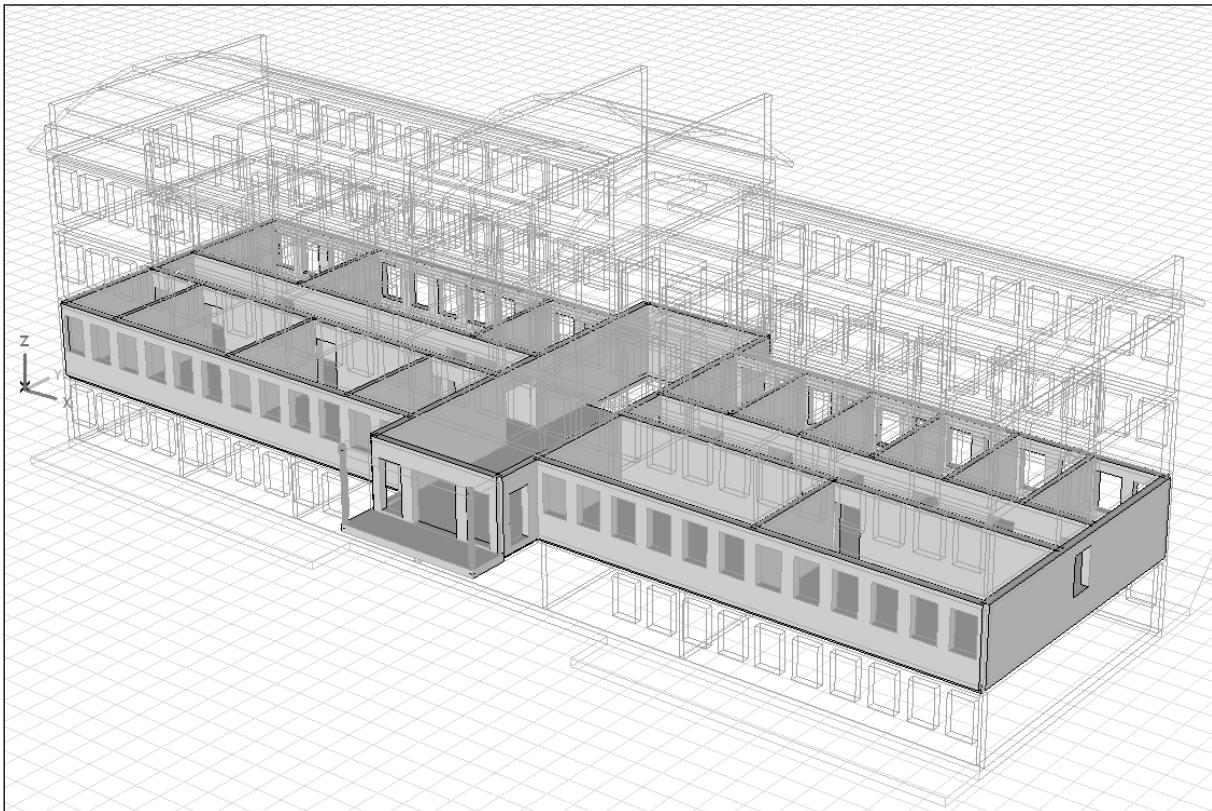
long	SelectAll ([in] ELongBoolean Select)
	Select selection state
	<i>If Select is True, selects all SteelDesignMembers.</i>
	<i>If Select is False, deselects all SteelDesignMembers.</i>
	<i>If successful, returns the number of selected SteelDesignMembers, otherwise returns an error code (errDatabaseNotReady).</i>
	<i>NOTE: Call Refresh function afterwards if not called between functions BeginUpdate and EndUpdate</i>
long	SetDesignParameters ([in] long Index , [out] SAFEARRAY(byte)* DesignParameters)
	Index SteelDesignMember index ($0 < \text{Index} \leq \text{IAxisVMSteelDesignMembers.Count}$)
	DesignParameters The DesignParameters must be type casted into the appropriate DesignParameters record type according to used national design code. See also IAxisVMSteelDesignMembers.Add
	<i>If successful returns SteelDesignMember index, otherwise an error code (errDatabaseNotReady, errIndexOutOfBounds, errNotSupportedByNationalDesignCode, sdmeInvalidNationalDesignCode)</i>
long	SetDesignParameters2 ([in] long Index , [i/o] RSteelDesignParameters SteelDesignParameters)
	Index SteelDesignMember index ($0 < \text{Index} \leq \text{IAxisVMSteelDesignMembers.Count}$)
	SteelDesignParameters Parameter which contain all design parameters from all national design codes
	<i>If successful returns index, otherwise an error code (errDatabaseNotReady, errIndexOutOfBounds, errNotSupportedByNationalDesignCode, sdmeInvalidNationalDesignCode)</i>
long	SetDesignParameters3 ([in] long Index , [i/o] SAFEARRAY(byte) SteelDesignParameters)
	Index SteelDesignMember index ($0 < \text{Index} \leq \text{IAxisVMSteelDesignMembers.Count}$)
	SteelDesignParameters Parameters which contain all design parameters of used national design code, size must match the size of the used record (e.g. RSteelDesignParameters_EC_SIA_ITA)
	<i>If successful returns index, otherwise an error code (errDatabaseNotReady, errIndexOutOfBounds, errNotSupportedByNationalDesignCode, sdmeInvalidNationalDesignCode)</i>
	<i>Note: The SteelDesignParameters must be type casted into the appropriate load record type depending on the national design code!</i>
long	GetLateralSupports ([in] long Index , [out] SAFEARRAY(RSteelLTBSupport)* LateralSupports)
	Index SteelDesignMember index ($0 < \text{Index} \leq \text{IAxisVMSteelDesignMembers.Count}$)
	LateralSupports lateral supports' settings
	<i>Get lateral supports' parameters. If successful returns design member's index, otherwise an error code (errDatabaseNotReady, errIndexOutOfBounds, errNotSupportedByNationalDesignCode, sdmeInvalidNationalDesignCode)</i>
long	SetLateralSupports ([in] long Index , [i/o] SAFEARRAY(RSteelLTBSupport)* LateralSupports)
	Index SteelDesignMember index ($0 < \text{Index} \leq \text{IAxisVMSteelDesignMembers.Count}$)
	LateralSupports lateral supports' settings
	<i>Sets lateral supports' parameters. If successful returns design member's index, otherwise an error code (errDatabaseNotReady, errIndexOutOfBounds, errNotSupportedByNationalDesignCode, sdmeInvalidNationalDesignCode)</i>

Properties

long	Count Get number of SteelDesignMembers.
long	CrossSectionID • Get or set considered cross-section index, 0 for original (default)
long	Length [long Index] Get length of a SteelDesignMember
<u>EBoolean</u>	Selected [long Index] • Get or set the selection status of a SteelDesignMember <i>NOTE:</i> Call Refresh function afterwards if not called between functions BeginUpdate and EndUpdate
long	SelCount Get number of selected SteelDesignMembers in the model

IAxisVMStoreys

Logical storeys in the model.



Enumerated types

```
enum EStoreyAutoSearchStyle = {  
    sassDomain = 0x00,           search by domains  
    sassBeam = 0x01,             search by beams  
    sassBoth = 0x02 }            search by domains and beams  
    Storey search options
```

Functions

long **Add** ([in] double z, [in] BSTR Name)

z Z coordinate of the new seismic storey

Name name of the new storey

Adds a new storey.

If successful, returns number of stories in the model, otherwise an error code ([errDatabaseNotReady](#)).

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

index storey index

Storey index starts from top (highest z value) with number 1. If successful, returns storey index of deleted storey, otherwise an error code ([errDatabaseNotReady](#) or [errIndexOutOfBounds](#)).

long **DeleteEmptyStoreys**

Deletes stories, which do not contain any elements. Returns number of deleted stories, otherwise an error code ([errDatabaseNotReady](#)).

long	Clear
Deletes all storeys. If successful, returns number of deleted stories, otherwise an error code (errDatabaseNotReady).	
long	GetItem ([in] long index , [out] BSTR Name , [out] long LevelID , [out] double z , [out] double Height)
<p style="margin-left: 20px;">index Storey index Name name of the storey LevelID level index according to height in the model z Z coordinate of the storey Height Height of the storey</p> <p>Get all properties of the storey. If successful, returns index of storey, otherwise an error code (errDatabaseNotReady or errIndexOutOfBounds).</p>	
long	IndexOfZ ([in] double z)
<p style="margin-left: 20px;">z z coordinate of the storey</p> <p>If successful, returns storey index, otherwise an error code (errDatabaseNotReady or errNotFound).</p>	
long	SetItem ([in] long index , [in] BSTR Name , [in] double z)
<p style="margin-left: 20px;">index index of the storey Name name of the storey z Z coordinate of the storey</p> <p>Set all properties of the storey. If successful, returns index of storey, otherwise an error code (errDatabaseNotReady or errIndexOutOfBounds).</p>	
long	DeleteEmptyStoreys
Deletes storeys, which do not contain any elements. Returns number of deleted storeys, otherwise an error code (errDatabaseNotReady).	
long	GetItem ([in] long index , [out] BSTR Name , [out] long LevelID , [out] double z , [out] double Height)
<p style="margin-left: 20px;">index Storey index Name name of the seismic storey LevelID index of the seismic storey z Z coordinate of the seismic storey Height Height of the seismic storey</p> <p>Get all properties of the seismic storey. If successful, returns index of seismic storey, otherwise an error code (errDatabaseNotReady or errIndexOutOfBounds).</p>	
long	GetSeismicSensitivityResults ([in] long index , [i/o] RSeismicSensitivityResults SeismicSensitivityResults)
<p style="margin-left: 20px;">index Storey index</p> <p>SeismicSensitivityResults All seismic sensitivity results</p> <p>Get seismic sensitivity results. If successful, returns index of seismic storey, otherwise an error code (errDatabaseNotReady or errIndexOutOfBounds).</p>	
long	IndexOfZ ([in] double z)
<p style="margin-left: 20px;">z z coordinate of the seismic storey</p> <p>If successful, returns storey index, otherwise an error code (errDatabaseNotReady or errNotFound).</p>	

long **SetItem** ([in] long **index**, [in] BSTR **Name**, [in] double **z**)
 index *Storey index*
 Name *name of the seismic storey*
 LevelID *index of the seismic storey*
 z *Z coordinate of the seismic storey*

Set all properties of the seismic storey. If successful, returns index of seismic storey, otherwise an error code ([errDatabaseNotReady](#) or [errIndexOutOfBounds](#)).

Properties

EStoreyAutoSearchStyle	long	AutoSearchStyle • Get or set the auto search style for storey search.
		Count Get number of storeys.
ElongBoolean		HasEmptyStoreys Returns <i>lbTrue</i> if storeys without elements found (Read only)
	double	Height [long Index] Get height of the storey. If successful, returns number otherwise an error code. (errDatabaseNotReady , errIndexOutOfBounds).
	long	LevelID [long Index] <i>Get level index of the storey. Level index was set automatically according to height in the model. If successful, returns positive number, otherwise an error code. (errDatabaseNotReady or errIndexOutOfBounds).</i>
	long	LevelZ [long Index] • Get or set z of the storey with index <i>Index</i> . If successful, returns number 1,2,..., otherwise 0.
	BSTR	Name [long Index] • Get or set name of the storey with index <i>Index</i> .

IAxisVMStructuralGrids

Structural grids in the model

Enumerated types

enum EShowStructuralGridLineTitle = { ssgltStart = 0x00, ssgltEnd = 0x01, ssgltBoth = 0x02}	<i>position of the title of structural grid's line</i>
enum EStructuralGridPlane = { sgp_XY = 0x00, sgp_XZ = 0x01, sgp_YZ = 0x02 sgp_WorkPlane = 0x03 sgp_Story = 0x04}	<i>plane of the structural grid</i> <i>XY plane</i> <i>XZ plane</i> <i>YZ plane</i> <i>workplane</i> <i>story's plane</i>
enum EStructuralGridVisibility = { sgvDefault = 0x00, sgvVisibleAtAllStories = 0x01, sgvOnlyIfActive = 0x02}	<i>structural grids' visibility</i> <i>default</i> <i>visible at all stories</i> <i>only if structural grid is active</i>
enum EStructuralGridLabelType = { sgltLetters = 0x00, sgltNumbers = 0x01}	<i>structural grid's label type</i> <i>letters</i> <i>numbers</i>

Error codes

enum EStructuralGridsError = { sgelInvalidName = -100001 , sgelInvalidWorkPlaneIndex = -100002 , sgelInvalidStoreyIndex = -100003 , sgwInvalidStartCharX = -100004 , sgwInvalidStartCharY = -100005 , sqwNormalVectorVaries = -100006 , sqwGridLineNotInPlane = -100007}	<i>structural grid's name is invalid</i> <i>workplane's index is invalid</i> <i>storey's index is invalid</i> <i>StartCharX is invalid</i> <i>StartCharY is invalid</i> <i>the normal vector varies</i> <i>the grid line is not in plane</i>
---	--

Records / structures

RStructuralGridParams = (

EStructuralGridPlane long double	Plane WorkPlaneOrStoreyIndex PlaneOffset	<i>plane of the structural grid</i> <i>index of workplane or storey</i> <i>distance between the plane and the grid</i>
---	---	--

NOTE: If the plane of the generated grid is identical with a workplane or a plane of a storey, the PlaneOffset is zero. Otherwise, e.g. if the plane of grid is the global XY plane, the PlaneOffset is equal to Z₀ in the software (or X₀, Y₀ related to global YZ and XZ planes, respectively):

The diagram shows five 3D coordinate systems. In each system, the X, Y, and Z axes are shown. A blue plane is defined in each. In the first system, the blue plane is parallel to the XY plane and passes through the origin. In the second, it is tilted at an angle to the XY plane. In the third, it is tilted at a different angle. In the fourth, it is parallel to the XZ plane. In the fifth, it is parallel to the YZ plane. This illustrates how the PlaneOffset value is calculated based on the relationship to the global coordinate system.

Structural grid creation interface:

The screenshot shows a dialog box for creating a structural grid. It includes:

- A preview area showing a 3D scene with a structural grid.
- A "Name" field containing "Structural grid 2".
- A "Color" button set to red.
- A checkbox labeled "Create structural grid" which is checked.
- Input fields for coordinates: X₀ [m] = 0, Y₀ [m] = 0, Z₀ [m] = 0.
- An angle input field: α [°] = 0.

EStructuralGridVisibility	Visibility	<i>visibility</i>
	RStructuralGridGenerationParams= (
	Offset	<i>local offset from origin</i>
	RotDeg	<i>rotation angle in degrees</i>
	Colour	<i>colour</i>
	Extension	<i>the distance of the label from structural grid's edge</i>
	LabelTypeX	<i>label type in X direction</i>
	GenerateInPositiveX	<i>generate grids in positive direction</i>
	LabelTypeY	<i>label type in Y direction</i>
	GenerateInPositiveY	<i>generate grids in positive direction</i>
	ShowStructuralGridLineTitle	<i>show grid line's title</i>
)	

[EStructuralGridLabelType](#) [ELongBoolean](#)
[EStructuralGridLabelType](#) [ELongBoolean](#)
[EShowStructuralGridLineTitle](#)

Functions

long **Add ([in] BSTR Name, [i/o] RStructuralGridParams StructuralGridParams)**

Name *name of the new structural grid*

StructuralGridParams *structural grid parameters*

Adds a new structural grid. If successful, returns structural grid's index, otherwise an error code ([EStructuralGridsError](#)).

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

Index *structural grid's index*

Adds a new structural grid. If successful, returns structural grid's index, otherwise an error code ([EStructuralGridsError](#)).

long **GenerateStructuralGrid ([in] long Index, [in] BSTR PrefixX, [in] BSTR StartCharX, [in] BSTR RelativeSpacingX, [in] BSTR PrefixY, [in] BSTR StartCharY, [in] BSTR RelativeSpacingY, [i/o] RStructuralGridGenerationParams StructuralGridGenerationParams)**

Index *structural grid's index*

PrefixX *prefix of X grid*

StartCharX *start value (e.g. "A") of X grid*

RelativeSpacingX *relative spacing (e.g. "3*4") of X grid*

PrefixY *prefix of Y grid*

StartCharY *start value (e.g. "1") of Y grid*

RelativeSpacingY *relative spacing (e.g. "3*4") of Y grid*

StructuralGridGenerationParams *parameters*

Generates structural grid. If successful, returns structural grid's index, otherwise an error code ([EStructuralGridsError](#)).

long **GetStructuralGridParams ([in] long Index, [out] BSTR Name, [i/o] RStructuralGridParams StructuralGridParams)**

Index *structural grid's index*

Name *structural grid's name*

StructuralGridParams *parameters*

Gets structural grid's parameters. If successful, returns structural grid's index, otherwise an error code ([EStructuralGridsError](#)).

long **SetStructuralGridParams ([in] long Index, [in] BSTR Name, [i/o] RStructuralGridParams StructuralGridParams)**

Index *structural grid's index*

Name *structural grid's name*

StructuralGridParams *parameters*

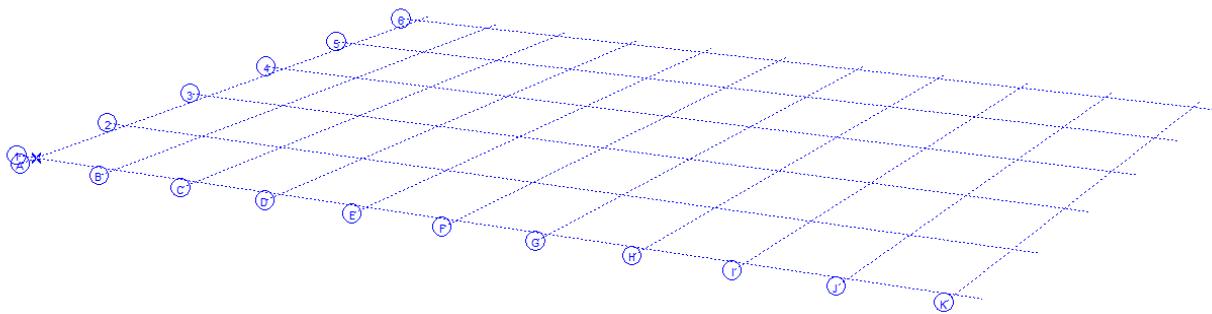
Sets structural grid's parameters. If successful, returns structural grid's index, otherwise an error code ([EStructuralGridsError](#)).

Properties

long	Count <i>Get number of structural grids</i>
<u>AxisVMStructuralGrid*</u>	Item [long Index] <i>Get structural grid by Index</i> Index <i>index of structural grid</i>
BSTR	Name• [long Index] <i>Get or set structural grid' name</i> Index <i>index of structural grid</i>

IAxisVMStructuralGrid

AxisVM StructuralGrid



Enumerated types

```
enum EStructuralGridPlane= {  
    sgp_XY = 0x00,  
    sgp_XZ = 0x01,  
    sgp_YZ = 0x02  
    sgp_WorkPlane = 0x03  
    sgp_Story = 0x04}  
  
    plane of the structural grid  
    XY plane  
    XZ plane  
    YZ plane  
    workplane  
    story's plane
```

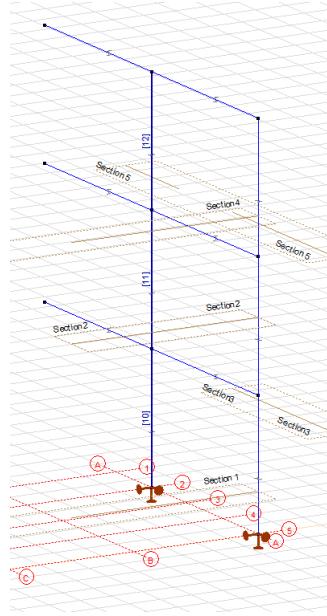
```
enum EGridLineSpacingDirection= {  
    glsd_X= 0x00,  
    glsd_Y= 0x01  
    glsd_Other= 0x02}  
  
    spacing direction of the grid line  
    local X of the grid  
    local Y of the grid  
    other
```

Error codes

see [AxisVMStructuralGrids' error codes](#).

Records / structures

```
RStructuralGridLineParams= (  
    RPoint3d P1 start point of structural grid's line  
    RPoint3d P2 endpoint of structural grid's line  
    RPoint3d NormalVector normal vector of the grid  
    long Colour colour  
    double Extension the distance of the label from structural grid's edge  
    EShowStructuralGridLineTitle show title  
    ELongBoolean the structural grid's line can be added to logical parts (note: see the figure below)  
    double ToLogicalPart the value of tolerance if the structural grid's line is added to logical parts (note: see the figure below)  
    )
```



Functions

long	AddLine ([in] BSTR Title, [i/o] RStructuralGridLineParams StructuralGridLineParams)
	<p>Title title of the new structural grid line StructuralGridParams structural grid line's parameters</p> <p>Adds a new structural grid line. If successful, returns structural grid line's index, otherwise an error code (EStructuralGridsError).</p>
long	GetLine ([in] long Index, [out] BSTR Title, [i/o] RStructuralGridLineParams StructuralGridLineParams)
	<p>Index Structural grid line's index Title title of the new structural grid line StructuralGridParams structural grid line's parameters</p> <p>Gets the title and parameters of a structural grid line by Id. If successful, returns structural grid line's index, otherwise an error code (EStructuralGridsError).</p>
long	SetLine ([in] long Index, [in] BSTR Title, [i/o] RStructuralGridLineParams StructuralGridLineParams)
	<p>Index Structural grid line's index Title title of the new structural grid line StructuralGridParams structural grid line's parameters</p> <p>Sets the title and parameters of a structural grid line by Id. If successful, returns structural grid line's index, otherwise an error code (EStructuralGridsError).</p>
long	DeleteLine ([in] long Index)
	<p>Index Structural grid line's index</p> <p>Deletes a structural grid line by Id. If successful, returns structural grid line's index, otherwise an error code (EStructuralGridsError).</p>

Properties

long	Count Get number of structural grid's lines
EStructuralGridPlane	Plane Get the plane of a structural grid's lines
long	UID [long Index] Get unique index of structural grid Index index of structural grid line

RMatrix3x3 **TrMatrix** transformation matrix of the structural grid's plane
Rpoint3D **NormalVector** normal vector of the grid

EGridLineSpacingDirection **SpacingDirection** [long **Index**] the spacing direction of the grid line
Index index of structural grid's line
BSTR **Name•** [long **Index**] Get or set the name of structural grid's line
Index index of structural grid's line

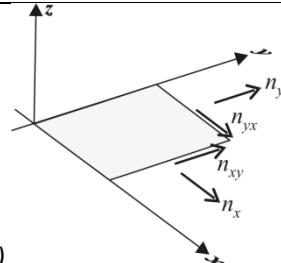
IAxisVMSurfaces

Surface elements of the model. (Elements can be generated by meshing domains)

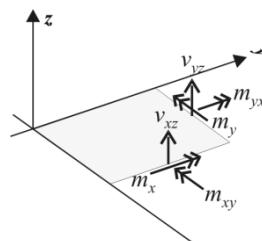
Enumerated types

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

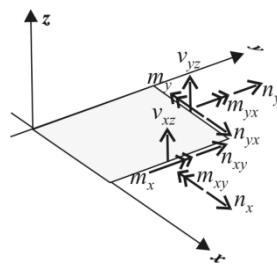
```
enum ESurfaceType = {
    stHole = 0x0,          hole
    stMembraneStress =     membrane (plane stress)
    0x1,
    stMembraneStrain =    Membrane (plane strain)
    0x2,
    stPlate = 0x3,         Plate
    stShell = 0x4 }
```



Membrane (plane strain)



Plate



Shell

Types of surface elements.

Error codes

```
enum ESurfacesError = {
    seLineCountCanBeOnly3Or4 = -100001,
    seCannotModify = -100002,
    seCOMError = -100003,
    seReinforcementParametersNotExists = -100004,
    seConcreteIdIndexOutOfBounds = -100005,
    seRebarSteelGradeIdIndexOutOfBounds = -100006,
    seThicknessMustBePositive = -100007,
    seRebarPosMustBePositive = -100008,
    sePhiMustBePositiveOrZero = -100009,
    seNuMustBePositiveOrZero = -100010,
    seTauaMustBePositiveOrZero = -100011,
    seAggregateSizeMustBePositive = -100012
    sePropertyNotValidForThisSurfaceType= -100013}
```

a surface can have 3 or 4 edges only
cannot modify surface element properties
internal COM server error
reinforcement parameters do not exists
Concrete's material index is out of bounds
rebar's material index index is out of bounds
domain thickness must be a positive number
rebar position must be a positive number
Phi must be a positive number
Nu must be a positive number
Tau_a must be a positive number
aggregate size must be a positive number
StiffnessReduction property can return this error

sefseMustBePositive = -100014 ,	<i>seismic load factor fse must be a positive value</i>
seParametersRecordNotValidForUsedDesignCode = -100015 ,	<i>used parameter record type is not valid for current design code</i>
seConcreteCoverMustBePositive = -100016 ,	<i>concrete cover must be a positive value</i>
seRebarDiameterMustBePositive = -100017 ,	<i>Rebar diameter must be a positive value</i>
seEnvironmentClassNotValidForUsedDesignCode = -100018 ,	<i>EnviromentClass is not valid for selected design code</i>
seAlphaVRdmaxIsInvalid = - 100019 ,	<i>the angle of shear reinforcement is invalid</i>
seThetaVRdmaxIsInvalid = - 100020 ,	<i>the angle of shear crack is invalid</i>
seShrinkageEpsMustBePositive = - 100021 ,	<i>shrinkage strain must be positive</i>
seRCNonlinearSurfTypeIsInvalid = - 100022 ,	<i>nonlinear surface type is invalid</i>
seAlphaAngleIsInvalid = - 100023 ,	<i>the angle of ξ reinforcement direction is invalid</i>
seBetaAngleIsInvalid = - 100024 ,	<i>the angle between ξ and η reinf. directions is invalid</i>
se_k_torsionIsInvalid = - 100025 ,	<i>k_torsion should be ≤ 0.1 and < 1</i>
se_k_shearIsInvalid = - 100026 ,	<i>k_shear should be ≤ 0.1 and < 1</i>
se_k_bendingIsInvalid = - 100027 ,	<i>k_bending should be ≤ 0.1 and < 1</i>
seLimitingCrackWidthIsInvalid = - 100028 ,	<i>the value for limiting crack width is invalid</i>
seMaterialIndex = - 100029 ,	<i>MaterialIndex must be : $1 \leq MaterialIndex \leq AxisVMMaterials.Count$</i>
seSurfaceReferenceIndexOutOfBounds = - 100030 ,	<i>reference index must be : $0 \leq ReferenceIndex \leq AxisVMReferences.Count$</i>
seInvalidType = - 100031 ,	<i>an enumerated type hase violated the valid range for that type</i>
seElasticFoundationNegative = - 100032 }	<i>Elastic foundation cannot be negative</i>

Records / structures

	RElasticFoundationXYZ = (
double	x, y, z	<i>elastic foundation stiffness in x, y, z directions [kN/m/m²]</i>
)	
	RNonLinearityXYZ = (
ELineNonLinearity	x, y, z	<i>nonlinear behaviour in x, y, z directions</i>
)	
	RResistancesXYZ = (
Double	x, y, z	<i>resistance in x, y, z directions [kN/m²]</i>
)	
	RSurface = (
long	N	<i>Number of contour lines (valid values are either 3 or 4). LineIndexes must be in the geometrically correct continuous sequence</i>
long	LineIndex1	<i>first contour line index</i>
long	LineIndex2	<i>second contour line index</i>
long	LineIndex3	<i>third contour line index</i>
long	LineIndex4	<i>fourth contour line index. Only required if N=4</i>
RSurfaceAttr	Attr	<i>surface attributes</i>
long	DomainIndex	<i>index of the domain containing this surface, 0 for none $0 \leq DomainIndex \leq AxisVMDomains.Count$</i>
)	

Functions

long **Add** ([in] SAFEARRAY(long) **LineIds**, [in] long **DomainId**, [i/o] **RSurfaceAttr** **SurfaceAttr**)

LineIds indexes 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](#) [LineIds contains less than 3 or more than 4 line indexes], [seMaterialIndex](#), [seSurfaceReferenceIndexOutOfBounds](#), [seInvalidType](#), [seElasticFoundationNegative](#)).

long **Add_vb** (Visual Basic compatible function of **Add**)

long **BulkAdd** ([in] SAFEARRAY(**RSurface**) **Surfaces**, [out] SAFEARRAY(long) **SurfaceIDs**)

Surfaces list of surface properties. The valid ranges for fields are :

$N : 3 \text{ or } 4$

$\text{LineIndex1} : 1 \leq \text{LineIndex1} \leq \text{AxisVMLines.Count}$

$\text{LineIndex2} : 1 \leq \text{LineIndex2} \leq \text{AxisVMLines.Count}$

$\text{LineIndex3} : 1 \leq \text{LineIndex3} \leq \text{AxisVMLines.Count}$

$\text{LineIndex4} : \text{if } N=3, \text{LineIndex4}=0, \text{if } N=4, 1 \leq \text{LineIndex4} \leq \text{AxisVMLines.Count}$

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

Attr : see : [RSurfaceAttr](#)

SurfaceIDs indexes of the created surfaces

Creates several surfaces. If successful, returns the number of surfaces created. The possible error codes are ([errDatabaseNotReady](#), [errIndexOutOfBounds](#), [errInternalException](#), [errOutOfMemory](#), [seLineCountCanBeOnly3Or4](#), [seMaterialIndex](#), [seSurfaceReferenceIndexOutOfBounds](#), [seInvalidType](#), [seElasticFoundationNegative](#)).

long **BulkGetSurfaces** ([in] SAFEARRAY(long) **SurfaceIDs**, [out] SAFEARRAY(**RSurface**) **Surfaces**)

SurfaceIDs list of surface indexes to query. Each index must satisfy: $(1 \leq \text{Index} \leq \text{AxisVMSurfaces.Count})$

Surfaces The list of surface records

Queries the properties of several surfaces. If successful, returns the number of surfaces queried. The possible error codes are ([errDatabaseNotReady](#), [errIndexOutOfBounds](#), [errInternalException](#), [errOutOfMemory](#)).

long **BulkSetSurfaces** ([in] SAFEARRAY(long) **SurfaceIDs**, [in] SAFEARRAY(**RSurface**) **Surfaces**)

SurfaceIDs list of surface indexes to set. Each index must satisfy: $(1 \leq \text{Index} \leq \text{AxisVMSurfaces.Count})$

Surfaces The list of surface property records

Sets the properties of existing surfaces. To create new surfaces use the BulkAdd function instead. If successful, returns the number of modified surfaces. The possible error codes are ([errDatabaseNotReady](#), [errIndexOutOfBounds](#), [errInternalException](#), [errOutOfMemory](#), [seLineCountCanBeOnly3Or4](#), [seMaterialIndex](#), [seSurfaceReferenceIndexOutOfBounds](#), [seInvalidType](#), [seElasticFoundationNegative](#)).

long **Delete** ([in] long **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](#)).

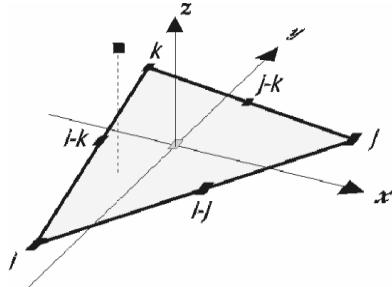
long	DeleteReinforcementParametersForSelectedSurfaces
	<i>If successful, returns 1. Otherwise returns error code (errDatabaseNotReady)</i>
long	GetSelectedItemIds ([out] SAFEARRAY (long) * ItemIds)
	ItemIds list of selected surfaces
	<i>If successful, returns the number of selected elements otherwise returns an error code (errDatabaseNotReady).</i>
long	GetAllCoordinatesOfSurfaces ([in] SAFEARRAY (long) * ItemIds, [out] SAFEARRAY(RSurfaceCoordinates) * SurfacesCoordinates)
	ItemIds list of surface indexes
	SurfacesCoordinates Surface coordinates of surfaces listed in ItemIds
	<i>If successful, returns the number of elements in ItemIds, otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBounds)</i>
long	GetAllCoordinatesOfSurfaces_vb (Visual Basic compatible function of GetAllCoordinatesOfSurfaces)
long	SelectAll ([in] ELongBoolean Select)
	Select selection state
	<i>If Select = True, selects all surface elements.</i>
	<i>If Select = False, deselects all surface elements.</i>
	<i>If successful, returns the number of selected elements otherwise returns an error code (errDatabaseNotReady).</i>
	<i>NOTE: Call Refresh function afterwards if not called between functions BeginUpdate and EndUpdate</i>

Properties

<u>AxisVMAttachments</u> *	Attachments Get the attachments interface
<u>AxisVMAtributes</u> *	Attributes Get the attributes interface
long	Count Get number of surface elements in the model
<u>AxisVMSurface</u> *	Item [long Index] Get surface element by Index
	Index index of the surface
long	IndexOfUID [long UID] Get index of the surface
	UID unique index of the surface
<u>ELongBoolean</u>	Selected [long Index] • Get or set the selection status of a surface element
	Index index of the surface
	<i>NOTE: Call Refresh function afterwards if not called between functions BeginUpdate and EndUpdate</i>
long	SelCount Get number of selected surface elements in the model. Negative number is an error code (errDatabaseNotReady).
double	StiffnessReduction [long Index] • Get or set stiffness reduction. Default value is 1. Used only for atLinearStatic and atLinearVibration analysis types . (only valid for STAS and Eurocode [RO] standards)
long	UID [long Index] Get unique index of the surface which remains the same while exists in the model
	Index index of the surface

IAxisVMSurface

An AxisVM surface element interface.



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, *membrane (plane strain)*
 stPlate = 0x3, *plate*
 stShell = 0x4 } *shell*

Types of surface elements.

enum **ESurfaceVertexIndex**= {
 sviCenterPoint = 0x0, *center point of the surface*
 sviContourPoint1 = 0x1, *corner points (only three for triangular surfaces)*
 sviContourPoint2 = 0x2,
 sviContourPoint3 = 0x3,
 sviContourPoint4 = 0x4,
 sviContourLineMidPoint1= 0x5, *midpoints of surface sides (only three for triangular surfaces)*
 sviContourLineMidPoint2 = 0x6,
 sviContourLineMidPoint3 = 0x7,
 sviContourLineMidPoint4 = 0x8 }
Index of surface vertex.

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/m/m²]*
)
RMatrix3x3 = (
 double **e11, e12, e13**
 double **e21, e22, e23**
 double **e31, e32, e33**
)

<u>ELineNonLinearity</u>	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]
double <u>ESurfaceType</u> long	RSurfaceAttr = (Thickness SurfaceType RefZId	surface thickness [m] surface type reference index for the local z direction (0 < RefZId ≤ <u>AxisVMReferences</u> .Count) or 0, if the reference is automatic
long	RefXId	reference index for the local x direction (0 < RefXId ≤ <u>AxisVMReferences</u> .Count) or 0, if the reference is automatic
long	MaterialId	index of the surface material (0 < MaterialId ≤ <u>AxisVMMaterials</u> .Count)
<u>RElasticFoundationXYZ</u> <u>RNonLinearityXYZ</u> <u>RResistancesXYZ</u>	ElasticFoundation NonLinearity Resistance Characteristics)	elastic foundation of the surface nonlinear behaviour of the elastic foundation resistance values of the elastic foundation nonlinear behaviour of the surface (not used)
<u>ESurfaceCharacteristics</u>	RSurfaceCoordinates = (
long	ContourPointCount	Number of contour points
long	ContourPoint1Id	Contour point index
long	ContourPoint2Id	Contour point index
long	ContourPoint3Id	Contour point index
long	ContourPoint4Id	Contour point index
long	ContourLine1Id	Contour line index
long	ContourLine2Id	Contour line index
long	ContourLine3Id	Contour line index
long	ContourLine4Id	Contour line index
<u>RPoint3d</u>	pContourPoint1	Coordinates of contour point 1
<u>RPoint3d</u>	pContourPoint2	Coordinates of contour point 2
<u>RPoint3d</u>	pContourPoint3	Coordinates of contour point 3
<u>RPoint3d</u>	pContourPoint4	Coordinates of contour point 4
<u>RPoint3d</u>	pContourLineMidPoint1	Coordinates of contour line mid point 1
<u>RPoint3d</u>	pContourLineMidPoint2	Coordinates of contour line mid point 2
<u>RPoint3d</u>	pContourLineMidPoint3	Coordinates of contour line mid point 3
<u>RPoint3d</u>	pContourLineMidPoint4	Coordinates of contour line mid point 4
)	

Functions

long	DeleteReinforcementParameters	If successful, returns surface index. Otherwise returns error code(<u>errIndexOutOfBounds</u> , <u>errDatabaseNotReady</u>)
long	GetContourPoints ([out] SAFEARRAY(long) ContourPoints)	Get 3 or 4 contour point indexes of the surface in an array. Array length = PointCount. If successful returns number of surface's contour points, otherwise returns an error code (<u>errIndexOutOfBounds</u> , <u>errDatabaseNotReady</u>).
long	GetMidPoints ([out] SAFEARRAY(long) MidPoints)	Get 3 or 4 edge midpoint indexes of the surface in an array. Array length = PointCount. If successful returns number of surface's midpoints, otherwise returns an error code (<u>errIndexOutOfBounds</u> , <u>errDatabaseNotReady</u>).
long	GetNormalVector ([i/o] <u>RPoint3d</u> Value)	Get the normal vector of the surface. If successful returns surface index, otherwise returns an error code (<u>errDatabaseNotReady</u>).

long	GetReinforcementParameters ([i/o] RReinforcementParameters ReinforcementParameters, [out] SAFEARRAY(byte) DesignCodeParameters)
	<p>ReinforcementParameters reinforcement parameters</p> <p>DesignCodeParameters a pointer to the reinforcement parameters record in SAFEARRAY format, record type depending on used design code. Typecast to a corresponding record (see here) to the design code, see How to read load data (GetLoad).</p> <p>If successful, returns the surface index, otherwise an error code (errDatabaseNotReady, errIndexOutOfBounds, seReinforcementParametersNotExists).</p>
long	GetSurfaceAttr ([i/o] RSurfaceAttr Value)
	Get the properties of the surface. If successful returns surface index, otherwise returns an error code (errDatabaseNotReady).
long	GetSurfaceStiffnessFactors ([i/o] RSurfaceStiffnessFactors Value)
	Get the stiffness factors of the surface. If successful returns surface index, otherwise returns an error code (errDatabaseNotReady).
long	GetTrMatrix ([i/o] RMatrix3x3 Value)
	Get the transformation matrix of the surface element. If successful returns surface index otherwise, returns an error code (errDatabaseNotReady).
long	GetVariableThickness ([out] SAFEARRAY(double) Thicknesses)
	<p>Thicknesses Array with thickness, with array indexes of:</p> <ul style="list-style-type: none"> <u>corners</u>: 0,2,4 (rectangular and triangular) and 6 (rectangular only) <u>contour midpoints</u>: 1,3,5 (rectangular and triangular) and 7(rectangular only) <u>the centre</u>: 6 (triangular) or 8 (rectangular) <p>If successful, returns length of the, otherwise an error code (errDatabaseNotReady, errIndexOutOfBounds, seReinforcementParametersNotExists).</p>
long	Modify ([in] SAFEARRAY(long) LineIds, [in] long DomainId, [i/o] RSurfaceAttr SurfaceAttr)
	<p>LineIds indexes of coplanar lines (3 or 4) forming edges of a surface. Array length = PointCount.</p> <p>See AxisVMLines</p> <p>DomainId domain index if the surface element is part of a mesh otherwise 0 $0 \leq \text{DomainId} \leq \text{AxisVMDomains.Count}$</p> <p>SurfaceAttr surface properties</p> <p>Redefines a new surface element.</p> <p>If successful, returns the surface index, otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBounds, seLineCountCanBeOnly3Or4 [LineIds contains less than 3 or more than 4 line indexes], seCannotModify).</p>
long	Modify_vb (Visual Basic compatible function of Modify)
long	SetContourLines ([in] SAFEARRAY(long) LineIds)
	Redefines surface edges. If successful, returns the surface index, otherwise returns an error code (errIndexOutOfBounds , errDatabaseNotReady , deEmptyContour , deMoreThanOneContourFound).
long	SetContourLines_vb (Visual Basic compatible function of SetContourLines)
long	SetReinforcementParameters ([i/o] RReinforcementParameters * ReinforcementParameters, [in] SAFEARRAY(byte) DesignCodeParameters)
	<p>ReinforcementParameters reinforcement parameters</p> <p>DesignCodeParameters pointer to the reinforcement parameters record in SAFEARRAY format, record type depending on used design code. Typecast to a corresponding record to a safearray (see GetReinforcementParameters), see also How to modify load data (SetLoad)</p>

If successful, returns index of the surface, otherwise returns an error code ([errDatabaseNotReady](#), [errIndexOutOfBounds](#), [seConcreteIdIndexOutOfBounds](#), [seRebarSteelGradeIdIndexOutOfBounds](#), [seThicknessMustBePositive](#), [seRebarPosMustBePositive](#), [sePhiMustBePositiveOrZero](#), [seNuMustBePositiveOrZero](#), [seTauaMustBePositiveOrZero](#), [seAggregateSizeMustBePositive](#))

long **SetReinforcementParameters _vb ([i/o] RReinforcementParameters* ReinforcementParameters, [i/o] SAFEARRAY(byte) DesignCodeParameters)**
Visual Basic compatible function of [SetReinforcementParameters](#)

long **SetSurfaceAttr ([i/o] RSurfaceAttr Value)**
Set the properties of the surface. . If successful returns surface index otherwise, returns an error code ([errDatabaseNotReady](#)).

long **SetSurfaceStiffnessFactors ([i/o] RSurfaceStiffnessFactors Value)**
Set the stiffness factors of the surface. If successful returns surface index, otherwise returns an error code ([errDatabaseNotReady](#)).

Properties

EArchitectElemType		ArchitectElemType • Get or set architect element type
double		Area • Get area of the surface element [m^2]
unsigned long		ContourColour • Get or set contour colour. If value is 0xFFFFFFFF then same as assigned material colour
long SAFEARRAY(long)*		ContourColour_vb • Visual Basic compatible property of ContourColour
long		ContourLines Get indexes of surface edges (3 or 4). Array length = PointCount.
unsigned long		DomainId • Get or set domain index if the surface element is part of a mesh otherwise 0
long		MaterialColour • Get or set material colour. If value is 0xFFFFFFFF then same as assigned material colour.
long		MaterialColour_vb • Visual Basic compatible property of MaterialColour
long		PointCount Get number of surface polygon vertices (3 or 4)
ELongBoolean		ReinforcementParametersExists True if finds reinforcement parameters (read only property)
double		StiffnessReduction • Get or set stiffness reduction. Default value is 1. Used only for atLinearStatic and atLinearVibration analysis types . (only valid for STAS and Eurocode [RO] standards)
double		Volume Get volume of the surface element [m^3]
double		Weight Get mass of the surface element [kg]
long		UID Get unique index of the surface which remains static from creation

IAxisVMSurfaceSupports

Surface supports of the model.

Records / structures

```
double RStiffnessesXYZ = (
    x, y, z           elastic foundation stiffness in local x, y, z directions of the surface [kN/m/m2]
)
ELineNonLinearity RNonLinearityXYZ = (
    x, y, z           nonlinear behaviour in local x, y, z directions of the surface
)
double RResistancesXYZ = (
    x, y, z           resistance in local x, y, z directions of the surface [kN/m2]
)
```

Functions

long	AddSurfaceElasticFoundation ([in] long SurfaceId, [i/o] RStiffnessesXYZ StiffnessesXYZ, [i/o] RNonLinearityXYZ NonLinearityXYZ, [i/o] RResistancesXYZ ResistancesXYZ)
	SurfaceId surface index 0 < SurfaceId ≤ AxisVMSurfaces.Count
	StiffnessesXYZ stiffnesses of the elastic foundation
	NonLinearityXYZ nonlinear behaviour of the elastic foundation
	ResistancesXYZ resistances of the elastic foundation
	<i>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, errIndexOutOfBoundsException).</i>
long	Delete ([in] double Index)
	Index surface support to delete
	<i>Deletes a surface support. 1 ≤ Index ≤ Count. If successful, returns Index otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBoundsException).</i>
long	DeleteSelected
	<i>Deletes selected surface supports. If successful, returns the number of deleted surface supports otherwise returns an error code (errDatabaseNotReady).</i>
long	SelectAll ([in] ELongBoolean Select)
	Select selection state
	<i>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). NOTE: Call Refresh function afterwards if not called between functions BeginUpdate and EndUpdate</i>
long	GetSelectedItemIds ([out] SAFEARRAY (long) * ItemIds)
	ItemIds list of selected surface supports
	<i>If successful, returns the number of selected elements otherwise returns an error code(errDatabaseNotReady).</i>

Properties

long AxisVMSurfaceSupport* EBoolean	<p>Count Get number of surface supports in the model</p> <p>Item [long Index] Get surface support interface</p> <p>Selected [long Index] • Get or set the selection status of a surface support</p> <p><i>NOTE:</i> Call Refresh function afterwards if not called between functions BeginUpdate and EndUpdate</p>
long	<p>SelCount Get number of selected surface supports in the model.</p> <p>Negative number is an error code (errDatabaseNotReady).</p>

IAxisVMSurfaceSupport

An AxisVM surface support interface.

Enumerated types

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

Records / structures

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

Functions

-
- long **GetNonLinearityXYZ** ([i/o] [RNonLinearityXYZ](#) Value)
Get the nonlinear behaviour of the surface support. If unsuccessful, returns an error code([errIndexOutOfBounds](#), [errDatabaseNotReady](#)).
-
- long **GetResistancesXYZ** ([i/o] [RResistancesXYZ](#) Value)
Get the resistance components of the surface support. If unsuccessful, returns an error code([errIndexOutOfBounds](#), [errDatabaseNotReady](#)).
-
- long **GetStiffnessesXYZ** ([i/o] [RStiffnessesXYZ](#) Value)
Get the stiffness components of the surface support. If unsuccessful, returns an error code([errIndexOutOfBounds](#), [errDatabaseNotReady](#)).
-
- long **SetNonLinearityXYZ** ([i/o] [RNonLinearityXYZ](#) Value)
Get the nonlinear behaviour of the surface support. If unsuccessful, returns an error code([errIndexOutOfBounds](#), [errDatabaseNotReady](#)).
-
- long **SetResistancesXYZ** ([i/o] [RResistancesXYZ](#) Value)
Get the resistance components of the surface support. If unsuccessful, returns an error code([errIndexOutOfBounds](#), [errDatabaseNotReady](#)).
-
- long **SetStiffnessesXYZ** ([i/o] [RStiffnessesXYZ](#) Value)
Get the stiffness components of the surface support. If unsuccessful, returns an error code([errIndexOutOfBounds](#), [errDatabaseNotReady](#)).
-

Properties

[ESurfaceSupportType](#) **SupportType** Get surface support type

IAxisVMTTask

Various AxisVM tasks

Enumerated types

enum	EDirectObjectDrawType = {	Type of element for direct object draw
	dodt_Column = 0x00,	vertical column
	dodt_BeamHorizontal = 0x01,	horizontal beam
	dodt_Beam = 0x02	general beam
	dodt_Wall = 0x03	vertical wall
	dodt_Slab = 0x04,	horizontal domain
	dodt_SlabVoids = 0x05,	domain with voids
	dodt_Domain = 0x06,	general domain
	dodt_Hole = 0x07 }	opening on domain

Error codes

enum	ETaskError = {	
	teCannotStartTask = -100001	the task can not be launched, another is running
	teCannotChangeMainTab = -100002	Can't change main tab in AxisVM
	}	

Functions

long	AddLoadModal ([in] ELoadType LoadType)	
	LoadType type of load to be added, allowed: <i>ItNodalForce</i> , <i>ItBeamConcentrated</i> , <i>ItBeamDistributed</i>	
	<i>Adds a new load using user interactive modal window. Switches main tab to "Loads". Shows a modal window and waits for user in AxisVM. If successful, returns 1, otherwise an error code (errDatabaseNotReady, errMembersNotAllowed, teCannotStartTask).</i>	
long	AddSupportModal ([in] EPartItemType ElementType)	
	ElementType type of support to be added, allowed: <i>pitNode</i> , <i>pitLine</i>	
	<i>Adds a new support using user interactive modal window. Switches main tab to "Elements". Shows a modal window and waits for user in AxisVM. If successful, returns 1, otherwise an error code (errDatabaseNotReady, errMembersNotAllowed, teCannotStartTask).</i>	
long	DrawObjectsDirectlyModal ([in] EDirectObjectDrawType ObjectType)	
	ObjectType type of object (element) to be added	
	<i>Adds a new element using user interactive modal window. Switches main tab to "Elements". Shows a modal window and waits for user in AxisVM. If successful, returns 1, otherwise an error code (errDatabaseNotReady, teCannotChangeMainTab, teCannotStartTask).</i>	
long	ShowDomainMeshingFormModal ()	
	<i>Switches main tab to "Mesh". Shows a modal window for mesh generation and waits for user in AxisVM. If successful, returns 1, otherwise an error code (teCannotChangeMainTab, teCannotStartTask).</i>	
long	ShowLoadCasesAndGroupsFormModal ()	
	<i>Switches main tab to "Loads". Shows a modal window for load case and load group definition and waits for user in AxisVM. If successful, returns 1, otherwise an error code (teCannotChangeMainTab, teCannotStartTask).</i>	

IAxisVMTimberDesignMembers

Interface for defining timber design members

If property returning this interface is null (nil) then the extension module TD1 is not available.

Enumerated types

enum	ETimberGrain = { tgTopEdge = 0 tgBottomEdge = 1 }	<i>grains parallel with top edge(tapered members)</i> <i>grains parallel with bottom edge(tapered members)</i>
enum	ELoadPositionType = { IptUpper = 0 IptAxis = 1 IptLower = 2 }	<i>load on top of the member</i> <i>load at centre of gravity of the member</i> <i>load on bottom of the member</i>

Error codes

enum	ETimberDesignMemberError = { <tdmelinelistisempty< td=""> = -100001 <tdmenotconnectinglines< td=""> = -100002 <tdmeinvalidnationaldesigncode< td=""> = -100003 }</tdmeinvalidnationaldesigncode<></tdmenotconnectinglines<></tdmelinelistisempty<>	<i>line list is empty</i> <i>lines are not connecting to each other</i> <i>design not valid for current national design code</i>
------	--	--

Records / structures

		RTimberDesignParameters_EC_SIA_ITA = (
double	Ky	<i>see timber beam design in AxisVM manual</i>
double	Kz	<i>see timber beam design in AxisVM manual</i>
double	Klt	<i>see timber beam design in AxisVM manual</i>
ELoadPositionType	LoadPosition	<i>load position (used for lateral torsional buckling)</i>
ETimberGrain	Grain	<i>arrangement of grains in tapered members</i>
double	LayerThickness	<i>thickness of layers of glued laminated timber (Glulam) [m]</i>
)	
		RTimberDesignParameters = (
ELongBoolean	BreakAtElements	<i>See here</i>
RTimberDesignParameters_EC_SI_A_ITA	EC_SIA_ITA	<i>timber design parameters</i>
)	

Functions

long **Add** ([i/o] [RTimberDesignParameters](#)* **TimberDesignParameters**,

[in] SAFEARRAY(long)* **Linelds**)

TimberDesignParameters timber design parameters

Linelds Index array (long) with lineIDs

Defines a new timber design member.

If successful, returns the new timber design member index, otherwise an error code

([errDatabaseNotReady](#), [errIndexOutOfBounds](#), [tdmeInvalidNationalDesignCode](#),

[tdmeNotConnectingLines](#), [tdmeLineListIsEmpty](#)).

long **Add_vb** (Visual Basic compatible function of [Add](#))

long **Clear**

Removes all timber design members. It returns the number removed of timber design members. If it returns a negative number, that is an error code

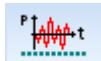
([errDatabaseNotReady](#), [errNotSupportedByNationalDesignCode](#), [tdmeInvalidNationalDesignCode](#)).

long	Delete ([in] long Index)
	<p>Index timber design member index ($0 < \text{Index} \leq \text{IAxisVMSteelDesignMembers.Count}$)</p> <p>Deletes a timber design member. If successful, returns the timber design member index ($1 \leq \text{Index} \leq \text{Count}$), otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBounds).</p>
<hr/>	
long	DeleteSelected
	<p>Returns number of deleted timber design member, otherwise returns an error code (errDatabaseNotReady, errNotSupportedByNationalDesignCode, tdmeInvalidNationalDesignCode).</p>
<hr/>	
long	GetDesignParameters ([in] long Index, [i/o] RTimberDesignParameters* TimberDesignParameters)
	<p>Index timber design member index ($0 < \text{Index} \leq \text{IAxisVMTimberDesignMembers.Count}$)</p> <p>TimberDesignParameters timber design parameters</p> <p>If successful, returns index otherwise an error code (errDatabaseNotReady, errIndexOutOfBounds, tdmeInvalidNationalDesignCode)</p>
<hr/>	
long	GetLineIds ([in] long Index, [out] SAFEARRAY(long)* LineIds)
	<p>Index timber design member index ($0 < \text{Index} \leq \text{TimberDesignMembers.Count}$)</p> <p>LineIds Index array (long) with lineIDs</p> <p>Returns number of lines in timber design member, otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBounds, errNotSupportedByNationalDesignCode, tdmeInvalidNationalDesignCode).</p>
<hr/>	
long	GetSelectedItemIds ([out] SAFEARRAY (long)* ItemIds)
	<p>ItemIds Index list of selected edge connections</p> <p>Returns number of selected timber design member, otherwise returns an error code (errDatabaseNotReady, errNotSupportedByNationalDesignCode, tdmeInvalidNationalDesignCode).</p>
<hr/>	
long	SelectAll ([in] ELongBoolean Select)
	<p>Select selection state</p> <p>If Select is True, selects all timber design members.</p> <p>If Select is False, deselects all timber design members.</p> <p>If successful, returns the number of selected timber design member, otherwise returns an error code (errDatabaseNotReady, errNotSupportedByNationalDesignCode, tdmeInvalidNationalDesignCode).</p> <p>NOTE: Call Refresh function afterwards if not called between functions BeginUpdate and EndUpdate</p>
<hr/>	
long	SetDesignParameters ([in] long Index, [i/o] RTimberDesignParameters* TimberDesignParameters)
	<p>Index timber design member index ($0 < \text{Index} \leq \text{IAxisVMTimberDesignMembers.Count}$)</p> <p>TimberDesignParameters timber design parameters</p> <p>If successful returns timber design memberindex, otherwise an error code (errDatabaseNotReady, errIndexOutOfBounds, tdmeInvalidNationalDesignCode)</p>

Properties

double	Count Get number of timber design members in the model, if 0 then results not calculated or invalid.
long	Length [long Index] Get length of a TimberDesignMember
<u>EBoolean</u>	Selected [long Index] • Get or set the selection status of a TimberDesignMember <i>NOTE:</i> Call Refresh function afterwards if not called between functions BeginUpdate and EndUpdate
long	SelCount Get number of TimberDesignMembers in the model

IAxisVMTTimeIncrementFunctions



Time increment functions, which can be used for dynamic analysis.

If property returning this interface is null (nil) then the extension module DYN is not available.

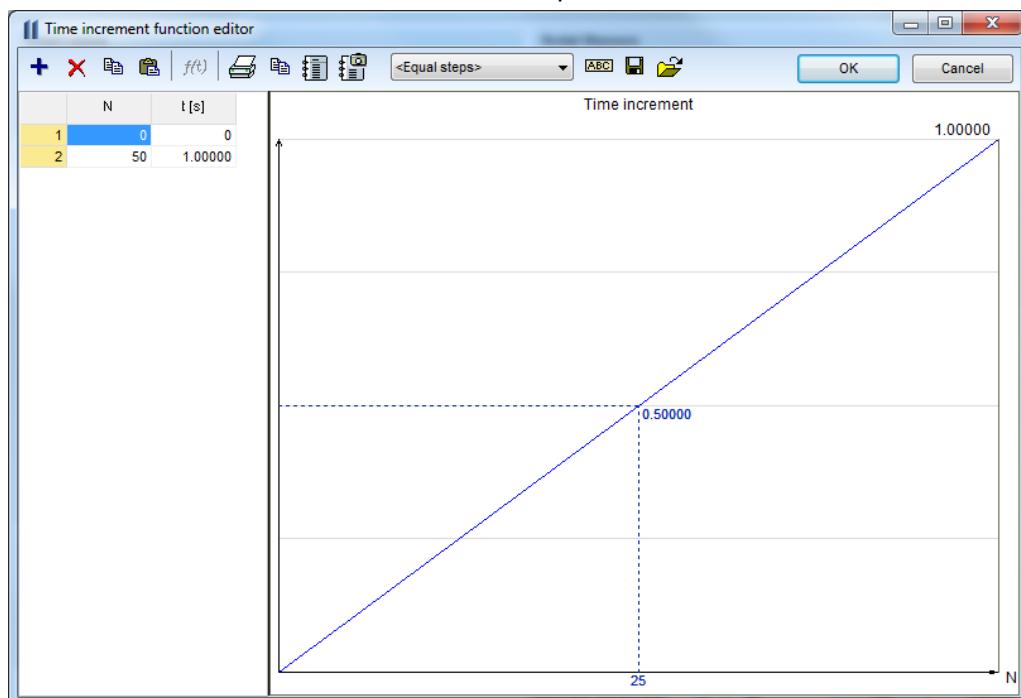
Functions

long **Add** ([in] BSTR Name, [in] SAFEARRAY(RPoint2D) FunctionPoints)

Name name of the new time increment function

FunctionPoints Function points of the time increment function where Coord1 is step N [-] and Coord2 is time [s] Coord1 must integer number. Array length must be 2. First point must be [0,0].

Adds a new time increment function. See example:



If successful, returns index of the new function, otherwise an error code ([fueNameAlreadyExists](#), [fueInvalidFunction](#), [errDatabaseNotReady](#), [errCOMServerInternalError](#)).

long **Add_vb** (Visual Basic compatible function of [Add](#))

long **AddFromFile** ([in] BSTR Name, [in] BSTR FileName)

Name Name of the time increment function

FileName Name of the file containing time increment function

If successful, returns function index, otherwise an error code ([errDatabaseNotReady](#), [fueFailedToAddFromFile](#)).

long **AddPoint** ([in] long index, [i/o] RPoint2d* FunctionPoint)

index time increment function index

FunctionPoint coordinates of the added point in the time increment function

If successful, returns number of points after adding point to the function, otherwise an error code ([errDatabaseNotReady](#) or [errIndexOutOfBounds](#)).

long **Clear**

Deletes all time increment functions.

If successful, returns 1, otherwise an error code ([errDatabaseNotReady](#)).

long	Delete ([in] long index)
	index <i>time increment function index</i>
	<i>If successful, returns number of time increment functions after delete, otherwise an error code (errDatabaseNotReady or errIndexOutOfBounds).</i>
long	DeletePoint ([in] long index , [in] long PointIndex)
	index <i>time increment function index</i>
	PointIndex <i>index of a point in time increment function</i>
	<i>If successful, returns number of points after delete a point from function, otherwise an error code (fuePointIndexOutOfBounds, errDatabaseNotReady or errIndexOutOfBounds).</i>
long	DeletePoints ([in] long index , [in] long StartPointIndex , [in] long EndPointIndex)
	index <i>time increment function index</i>
	StartPointIndex <i>index of the start point in time increment function</i>
	EndPointIndex <i>index of the end point in time increment function</i>
	<i>If successful, returns number of points after delete points from function, otherwise an error code (fuePointIndexOutOfBounds, errDatabaseNotReady or errIndexOutOfBounds).</i>
long	GetPoints ([in] long index , [out] SAFEARRAY(RPoint2d)* FunctionPoints)
	index <i>time increment function index</i>
	FunctionPoints <i>point array of the time increment function</i>
	<i>Deletes function point range starting with point index StartPointIndex (including) and ending with EndPointIndex (including).</i>
	<i>If successful, returns number of points of the function, otherwise an error code (errDatabaseNotReady, errIndexOutOfBounds)</i>
long	IndexOf ([in] BSTR Name)
	Name <i>Name of the time increment function</i>
	<i>If successful, returns function index, otherwise an error code (errDatabaseNotReady).</i>
long	Modify ([in] long index , [in] SAFEARRAY(RPoint2d)* FunctionPoints)
	index <i>time increment function index</i>
	FunctionPoints <i>point array of the time increment function, see function Add for more info about point coordinates.</i>
	<i>If successful, returns number of points of the function, otherwise an error code (errDatabaseNotReady, errIndexOutOfBounds, fueFailedToModifyFunction or errCOMServerInternalError).</i>
long	Modify_vb (Visual Basic compatible function of Modify)
long	SaveToFile ([in] long index , [in] BSTR FileName)
	index <i>time increment function index</i>
	FileName <i>Name of the file used for saving</i>
	<i>Saves function to AxisVM directory \timeinc with file extension: tnc</i>
	<i>If successful, returns function index, otherwise an error code (errDatabaseNotReady, errIndexOutOfBounds or fueFileExists).</i>

Properties

- long **Count** *Get number of time increment functions.*
- BSTR **Name [long Index]** *Get or set name of the time increment function with index Index.*
- long **PointCount [long Index]** *Get number of points of the time increment function with index Index.*

IAxisVMVirtualBeams

Interface for virtual beams and virtual strips of the model.



Note:

There are virtual beams and virtual strips in the software, however, this interface has been developed for handling both. Different functions were defined for creating and modifying virtual beams and strips, respectively, as the name of functions shows their purpose.

Indices of virtual beams and strips are stored in the same list in order related to their create order, thus if one reads Count property, it will return the total number of virtual beams and strips.

Analysis results are integrated internal forces ([RVirtualBeamForceValues](#)) ← see [Virtual Beam and Strip Forces in IAxisVMForces](#).

Enumerated types

enum	EVBDomainsDuplicateMode = {	
	ddmDuplicationError = 0x0,	<i>if a domain (given with domain index in DomainIds array by the use of AddNewforDomains function) already belongs to a virtual beam, the function returns with an error code</i>
	ddmNoDuplication = 0x1,	<i>if a domain (given with domain index in DomainIds array by the use of AddNewforDomains function) already belongs to a virtual beam, the domain will be deleted from the domains of previous virtual beam</i>
	ddmDuplication = 0x2 }	<i>if a domain (given with domain index in DomainIds array by the use of AddNewforDomains function) already belongs to a virtual beam, the domain will belong to more than one virtual beam</i>
enum	EVVirtualBeamType = {	
	vbtVirtualBeam = 0x0,	<i>virtual element for integration is assigned to domains (virtual beam)</i>
	vbtVirtualStrip = 0x1 }	<i>virtual element for integration in the model is given by a strip (virtual strip)</i>
enum	EVBDefinitionType = {	
	vbdtCentroid = 0x0,	<i>virtual beam is defined by centroids</i>
	vbdtStraight = 0x1	<i>virtual beam is defined as a straight line</i>
	vbdt1PAndV = 0x2	<i>virtual beam is defined with the help of a point and a vector</i>
	vbdt2P = 0x3 }	<i>virtual beam is defined with the help of two points</i>
enum	EVSDefinitionType = {	
	vsdtCentroid = 0x0,	<i>virtual strip is defined without eccentricity</i>
	vsdtEccentric = 0x1 }	<i>virtual strip is defined with eccentricity</i>

Error codes

enum	EVirtualBeamError = {	
	vbeDomainIndexOutOfBounds = -100001,	<i>domain index is out of bounds</i>
	vbeDomainIndexIsInvalid = -100002,	<i>domain index is invalid (e.g. it is given twice in the list)</i>
	vbeDomainListIsEmpty = -100003,	<i>domain list is empty</i>
	vbeChainIndexOutOfBounds = -100004,	<i>domain index is out of bounds</i>
	vbeDuplication = -100005,	<i>a given domain already belongs to another virtual beam</i>

```

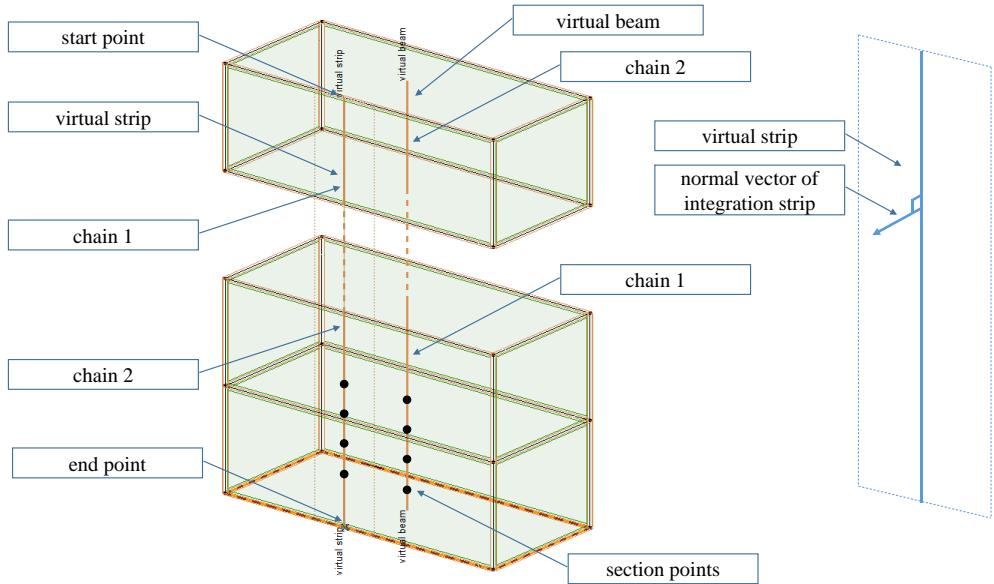
vbeInvalidParameters = -100006,
vbeInvalidReferenceParams = -100007
vbeVirtualBeamNoSection = -100008
vbeInvalidSections = -100009
}

```

*(note: only is EDomainsDuplicateMode = ddmDuplicationError)
 the input parameters or called function is not compatible with the selected virtual beam/strip or few of the parameters may have not valid value
 the type, the length or the parameters of the reference is invalid
 section cannot be created
 cross section is invalid*

Records / structures

	RVirtualBeamParams = (
BSTR	Name	<i>name of the domain virtual beam</i>
long	LocXVid	<i>index of the reference vector (see IAxisVMReferences) being parallel with the local X</i> <i>Note: if it is equal to 0 then the reference vector is automatic</i>
long	LocZVid	<i>index of the reference vector (see IAxisVMReferences) being parallel with the local Z</i> <i>Note: if it is equal to 0 then the reference vector is automatic</i>
<u>RPoint3d</u>	LocXV	<i>local X vector (note: used only for output)</i>
<u>RPoint3d</u>	LocZV	<i>local Z vector (note: used only for output)</i>
long	SectionI	<i>cross section index related to ith end of virtual beam</i>
long	SectionJ	<i>cross section index related to jth end of virtual beam</i>
<u>EVBDefinitionType</u>	DefinitionType	<i>definition type</i>
<u>RPoint3d</u>	P1	<i>first point</i>
<u>RPoint3d</u>	P2	<i>second point</i>
<u>EBoolean</u>	InnerDomains	<i>consider inner domains while creating virtual beam</i>
)	
	RVirtualStripParams = (
BSTR	Name	<i>name of the virtual strip</i>
<u>RPoint3d</u>	StartP	<i>start point of the virtual strip, coordinates in metres.</i>
<u>RPoint3d</u>	EndP	<i>end point of the virtual strip, coordinates in metres.</i>
long	SectionI	
long	SectionJ	
<u>RPoint3d</u>	NormV	<i>normal vector of integration strip</i> <i>note: if it is not perpendicular to the vector from StartP to EndP, its perpendicular component is considered</i>
double	L	<i>integration width on the left hand side [m]</i>
double	R	<i>integration width on the right hand side [m]</i>
<u>EVSDefinitionType</u>	DefinitionType	<i>definition type</i>
double	EccIY	<i>local y eccentricity at start point of virtual strip [m]</i>
double	EccIZ	<i>local z eccentricity at start point of virtual strip [m]</i>
double	EccJY	<i>local y eccentricity at end point of virtual strip [m]</i>
double	EccJZ	<i>local z eccentricity at end point of virtual strip [m]</i>
)	



Functions

long **AddNewVirtualBeam** ([i/o] SAFEARRAY(long)* **DomainIds**, [in] **EVBDomainsDuplicateMode DuplicateMode**, [i/o] **RVirtualBeamParams** **VirtualBeamParams**)

DomainIds array of domain indices considered for the new virtual beam
 $0 < \text{Index} < \text{IAxisVMDomains.Count} + 1$

DuplicateMode mode of handling of duplication

VirtualBeamParams parameters of the new virtual beam

Add a new virtual beam to the domains included in **DomainIds** array. If successful returns the index of the new virtual beam, otherwise an error code ([errDatabaseNotReady](#), [errInvalidName](#), [vbeDomainIndexIsInvalid](#), [vbeDomainIndexOutOfBounds](#), [vbeDomainListIsEmpty](#) or [vbeDuplication](#)).

long **AddNewVirtualStrip** ([i/o] **RVirtualStripParams** **VirtualStripParams**)

VirtualStripParams parameters of the new virtual strip

Add a new virtual strip to the model. If successful returns the index of the new virtual strip, otherwise an error code ([errDatabaseNotReady](#), [errIndexOutOfBounds](#), [errInvalidName](#) or [vbeInvalidParameters](#)).

long **AddNewDomainToVirtualBeam** ([in] long **Index**, [in] long **DomainId**, [in] **EVBDomainsDuplicateMode** **DuplicateMode**)

Index index of the virtual beam

DomainId index of the domain added to the virtual beam

DuplicateMode mode of handling of duplication

Add a domain to the virtual beam identified with **Index**. If successful returns the index of the virtual beam, otherwise an error code ([errDatabaseNotReady](#), [errIndexOutOfBounds](#), [vbeDomainIndexIsInvalid](#), [vbeDomainIndexOutOfBounds](#), [vbeInvalidParameters](#) or [vbeDuplication](#)).

long **Clear**

Removes all virtual beams and strips. It returns the number of virtual beams and strips removed. If it returns a negative number, then it is an error code ([errDatabaseNotReady](#)).

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

Index index of the virtual beam/strip to delete

Deletes a virtual beam/strip. $1 \leq \text{Index} \leq \text{Count}$. If successful, returns the Index, otherwise returns an error code ([errDatabaseNotReady](#), [errIndexOutOfBounds](#)).

long **GetCenterCoordinates** ([in] long **Index**, [in] long **ChainIndex**, [out] SAFEARRAY([RPoint3D](#))* **CenterCoordinates**)

Index *index of the virtual beam/strip*

ChainIndex *index of a chain of the virtual beam/strip*

0 < ChainIndex < IAxisVMVirtualBeams.ChainCount + 1

CenterCoordinates *coordinates of virtual beam's/strip's center points in an array related to a chain*

note: the length of the array is equal to SectionCount

Retrieves with center coordinates of a chain of a virtual beam/strip identified with ChainIndex/Index. If successful returns with the number of coordinates included in the array, otherwise an error code ([errDatabaseNotReady](#), [errIndexOutOfBounds](#) or [vbeChainIndexOutOfBounds](#), [errCOMServerInternalError](#))

long **GetReductionPointCoordinates** ([in] long **Index**, [in] long **ChainIndex**, [out] SAFEARRAY([RPoint3D](#))* **ReductionPointCoordinates**)

Index *index of the virtual beam/strip*

ChainIndex *index of a chain of the virtual beam/strip*

0 < ChainIndex < IAxisVMVirtualBeams.ChainCount + 1

ReductionPointCoordinates *coordinates of virtual beam's/strip's reduction points in an array related to a chain*

note: the length of the array is equal to SectionCount

Retrieves with reduction points' coordinates of a chain of a virtual beam/strip identified with ChainIndex/Index. If successful returns with the number of coordinates included in the array, otherwise an error code ([errDatabaseNotReady](#), [errIndexOutOfBounds](#) or [vbeChainIndexOutOfBounds](#), [errCOMServerInternalError](#))

Note: Reduction points are the points where the internal forces of a virtual beam/strip are integrated.

long **GetVirtualBeamParams** ([in] long **Index**, [i/o] [RVirtualBeamParams](#) **VirtualBeamParams**)

Index *index of the virtual beam*

VirtualBeamParams *virtual beam parameters*

Retrieves with the parameters in a record of the virtual beam identified with Index. If successful returns with the index of virtual beam, otherwise an error code ([errDatabaseNotReady](#), [errIndexOutOfBounds](#))

long **SetVirtualBeamParams** ([in] long **Index**, [i/o] [RVirtualBeamParams](#) **VirtualBeamParams**)

Index *index of the virtual beam*

VirtualBeamParams *virtual beam parameters*

Sets the parameters of a virtual beam identified with Index. If successful returns with the index of virtual beam, otherwise an error code ([errDatabaseNotReady](#), [errIndexOutOfBounds](#), [vbelInvalidParameters](#), [errInvalidName](#) or [ybelInvalidReferenceParams](#))

long **GetVirtualStripParams** ([in] long **Index**, [i/o] [RVirtualStripParams](#) **VirtualStripParams**)

Index *index of the strip virtual beam*

VirtualStripParams *virtual strip parameters*

Retrieves with the parameters in a record of the virtual strip identified with Index. If successful returns with the index of virtual strip, otherwise an error code ([errDatabaseNotReady](#), [errIndexOutOfBounds](#))

long	SetVirtualStripParams ([in] long Index , [i/o] RVirtualStripParams VirtualStripParams)
	Index index of the domain virtual beam
	VirtualStripParams virtual strip parameters
	<i>Sets the parameters of a virtual strip identified with Index. If successful returns with the index of virtual strip, otherwise an error code (errDatabaseNotReady, errIndexOutOfBounds, vbeInvalidParameters, errInvalidName or vbeInvalidReferenceParams)</i>
long	IndexOf ([in] BSTR Name)
	Name name of the virtual beam
	<i>Retrieves the index of the virtual beam/stripe having name equal to Name. If successful, returns the index of the virtual beam/stripe, otherwise returns an error code (errDatabaseNotReady, errNotFound).</i>
long	GetDomains ([in] long Index , [out] SAFEARRAY(long)* DomainIds)
	Index index of the domain virtual beam/stripe
	DomainIds array of domain indices considered for the virtual beam/stripe 0<Index<IAxisVMDomains.Count+1
	<i>Retrieves a list of domains connected to the virtual beam/stripe identified with Index. If successful returns the number of domains of the virtual beam/stripe, otherwise an error code (errDatabaseNotReady, errIndexOutOfBounds or vbeInvalidParameters).</i>
long	GetExtendedDomainList ([in] long Index , [out] SAFEARRAY(long)* DomainIds)
	Index index of the domain virtual beam/stripe
	DomainIds array of domain indices considered for the virtual beam/stripe 0<Index<IAxisVMDomains.Count+1
	<i>Retrieves a list of domains connected to the virtual beam/stripe identified with Index. The list is extended compared to GetDomains with the indices of inner domains. If successful returns the number of domains of the virtual beam/stripe, otherwise an error code (errDatabaseNotReady, errIndexOutOfBounds or vbeInvalidParameters).</i>
long	ModifyDomains ([in] long Index , [i/o] SAFEARRAY(long)* DomainIds , [in] EVBDomainsDuplicateMode DuplicateMode)
	Index index of the domain virtual beam/stripe
	DomainIds array of domain indices considered for the new virtual beam/stripe 0<Index<IAxisVMDomains.Count+1
	DuplicateMode mode of handling of duplication
	<i>Modifies the list of domains connected to virtual beam/stripe identified with Index. If successful returns the index of the virtual beam/stripe, otherwise an error code (errDatabaseNotReady, errIndexOutOfBounds, vbeDomainIndexIsInvalid, vbeDomainIndexOutOfBounds, vbeDomainListIsEmpty, vbeInvalidParameters or vbeDuplication).</i>

Properties

long	Count <i>number of virtual beams and strips in the model</i>
long	ChainCount [<i>long Index</i>] Index <i>index of the virtual beam/strip</i> <i>Get the number of chains related to a virtual beam/strip identified with Index</i>
long	SectionCount [<i>long Index</i> , <i>long ChainIndex</i>] Index <i>index of the virtual beam/strip</i> ChainIndex <i>index of a specific chain of a virtual beam/strip</i> <i>Get the number of section points related to a virtual beam's/strip's chain identified with ChainIndex</i>
double	Length [<i>long Index</i> , <i>long ChainIndex</i>] Index <i>index of the virtual beam/strip</i> ChainIndex <i>index of a specific chain of a virtual beam/strip</i> <i>Get the length of a virtual beam's/strip's chain</i>
BSTR	Name [<i>long Index</i>] • Index <i>index of the virtual beam/strip</i> <i>Get or set the name of a virtual beam/strip</i>
<u>EVirtualBeamType</u>	VirtualBeamType [<i>long Index</i>] Index <i>index of the virtual beam/strip</i> <i>Get the type of a virtual beam/strip</i>
long	IndexOfUID [<i>long UID</i>] <i>Get index of the virtual beam/strip</i> UID <i>unique index of the virtual beam/strip</i>
long	UID [<i>long Index</i>] <i>Get unique index of the virtual beam/strip which remains the same while exists in the model</i> Index <i>index of the virtual beam/strip</i>

IAxisVMWindows

Used for changing display settings of all AxisVM windows

Enumerated types

enum	EDisplayMode = { dmDiagram = 0x00, dmDiagramFilled = 0x07, dmDiagramAvgVals = 0x01, dmSectionLine = 0x02, dmSectionFilled = 0x08, dmlsolines = 0x03 , dmlsosurfaces2D = 0x04 , dmlsosurfaces3D = 0x05 , dmlsosurfacesAvgVals = 0x09, dmNone = 0x06 } <i>Type of display mode.</i>	<i>Display diagram</i> <i>Display filled diagram</i> <i>Display diagram+ average values if possible</i> <i>Display section lines</i> <i>Display filled section lines</i> <i>Display Isolines</i> <i>Display Isosurfaces 2D</i> <i>Display Isosurfaces 3D</i> <i>Display Isosurfaces with average values</i> <i>Display none</i>
enum	EDisplayAnalysisType = { datLinear = 0x00, datNonLinear = 0x01 } <i>Analysis type of displayed results</i>	<i>Display results of linear analysis</i> <i>Display results of nonlinear analysis</i>
enum	EDisplayShape = { dsUndeformed = 0x00, dsDeformed = 0x01 } <i>Show deformation or not.</i>	<i>Model is not deformed</i> <i>Model is deformed</i>
enum	EDisplayedEnvelopes = { de_Current = 0x00, de_Custom = 0x01, de_All = 0x02 } <i>Displayed envelopes</i>	<i>Only the selected envelope</i> <i>Only custom envelopes</i> <i>All envelopes</i>
enum	EActualReinforcementLabelType = { arlRebarsAndReinfValues = 0x00, <i>Rebars + Reinforcement values</i> arlRebarsAndQuantity = 0x01 } <i>Rebars + Quantity x (Length)</i> <i>Type of actual reinforcement labels</i>	
enum	ECriticalCombinationFormula = { ccfAuto = 0x00, ccfCustom = 0x01 } <i>Critical combination formula</i>	<i>Automatically selected combination type</i> <i>Custom</i>
enum	ESmoothing = { smthNone = 0x00, smthSelective = 0x01 , smthAllSurf = 0x02 } <i>Display smoothing of results</i>	<i>None</i> <i>Selective</i> <i>All surfaces</i>
enum	EIntensityReferenceValue = { irvAbsMaxModel = 0x00, irvAbsMaxParts = 0x01 , irvCustomVal = 0x02 } <i>Intensity reference value</i>	<i>Absolute maximum of the entire model</i> <i>Absolute maximum of the current parts</i> <i>Custom value</i>

enum	EWindowSplit = { wsHorizontal = 0x00, wsVertical = 0x01 } <i>Type of window splitting</i>	<i>Split windows Horizontally</i> <i>Split windows vertically</i>
enum	ECombinationMethod = { cmULS = 0x00, cmULSab = 0x01 } <i>Combination method in persistent and transient design situations</i>	<i>Conservative ULS combination formula</i> <i>Newer ULS combination formulas see EN 1990:6.10(a) and (b)</i>
enum	EWindowColourMode = { wcmColour = 0x00, wcmGreyScale = 0x01, wcmBlackAndWhite = 0x02 } <i>Colour mode of the window</i>	<i>Colour</i> <i>Greyscale</i> <i>Black and white</i>

Error codes

enum	EWindowsError = { weLoadCaseIdOutOfBounds = -100001 weLoadCombinationIdOutOfBounds = -100002 weEnvelopeIdOutOfBounds = -100003 }	<i>Load case index is not valid</i> <i>Load combination index is not valid</i> <i>Envelope index is not valid</i>
------	--	---

Constants

Switch constants (1..170)

- 1 Node numbers/names
- 2 Beam numbers/names
- 3 Parts
- 4 Grid
- 5 Beam end releases
- 6 DXF layer detection
- 7 Truss numbers/names
- 8 Architectural (ACH) layer detection
- 9 Beam local system
- 10 Texture
- 11 N/A
- 12 Nodes graphical symbol
- 13 Supports (all types)
- 14 Loads (all types)
- 15 Nodal masses graphical symbol
- 16 Surface numbers/names
- 17 Rib numbers/names
- 18 Rib local system
- 19 View type : hidden line removal
- 20 Surface local system
- 21 Line/surface mesh
- 22 Surface graphical symbol
- 23 Reference numbers/names
- 24 Reference graphical symbol
- 25 View type : rendered
- 26 Sections (all types)
- 27 Cross-section shape
- 28 Surface result values
- 29 N/A
- 30 N/A
- 31 Node result values

- 32 Line element/member result values
- 33 N/A
- 34 Reinforcement parameters
- 35 Support numbers/names
- 36 N/A
- 37 N/A
- 38 DXF layers
- 39 Guidelines
- 40 N/A
- 41 Cross-section numbers
- 42 Material numbers
- 43 Load values (all types)
- 44 Info window
- 45 Color legend window
- 46 Rigid element numbers/names
- 47 N/A
- 48 Cross-section names
- 49 Line element/member length
- 50 N/A
- 51 Support local system
- 52 Material names
- 53 Relative origin symbol
- 54 Surface thickness
- 55 Nodal mass value
- 56 Units (for values)
- 57 Spring local system
- 58 Gap local system
- 59 Spring numbers/names
- 60 Gap numbers/names
- 61 Architectural (ACH) layer
- 62 N/A
- 63 N/A
- 64 N/A
- 65 N/A
- 66 N/A
- 67 Domainr
- 68 N/A
- 69 Design member numbers/names
- 70 Domain local system
- 71 Domain numbers/names
- 72 Result values : min/max only
- 73 AxisVM layer
- 74 AxisVM layer detection
- 75 Non visible parts grayed
- 76 Concentrated loads
- 77 Line loads
- 78 Surface loads
- 79 Temperature loads
- 80 Self weight loads
- 81 Misc/other loads
- 82 Link element local system
- 83 Link elements
- 84 Link element numbers/names
- 85 Center of circle
- 86 Background picture
- 87 Nodal support graphical symbol
- 88 Line/member support graphical symbol
- 89 Surface support graphical symbol
- 90 Edge hinge local system
- 91 Edge hinge numbers/names

- 92** ARBO/CRET elements graphical symbol
- 93** ARBO/CRET element numbers/names
- 94** Load distribution scheme
- 95** Derived loads
- 96** Use finite element numbers
- 97** Diaphragm numbers/names
- 98** All moving load phases simultaneously
- 99** Object contours in 3D
- 100** Actual surface reinforcement
- 101** Rigid elements graphical symbol
- 102** Diaphragm graphical symbol
- 103** Actual reinforcement x-bottom labels
- 104** Actual reinforcement x-top labels
- 105** Actual reinforcement y-bottom labels
- 106** Actual reinforcement y-top labels
- 107** Actual reinforcement labels as Rebars + reinforcement values
- 108** Actual reinforcement labels as Rebars + Quantity x (Length)
- 109** Storey shear center graphical symbol (earthquake)
- 110** Storey center of gravity graphical symbol (earthquake)
- 111** Actual reinforcement x-bottom graphical symbol
- 112** Actual reinforcement x-top graphical symbol
- 113** Actual reinforcement y-bottom graphical symbol
- 114** Actual reinforcement y-top graphical symbol
- 115** N/A
- 116** Bolted joint labels/properties
- 117** Column reinforcement labels/properties
- 118** Actual reinforcement values according to the displayed result component
- 119** Void formers graphical symbol
- 120** N/A
- 121** Void former labels/names
- 122** Domain area labels
- 123** Structural grids graphical symbols
- 124** N/A
- 125** Footing/foundation graphical symbols
- 126** Footing/foundation dimension lines
- 127** Color coding window visibility
- 128** Design member label
- 129** Design optimisation group label
- 130** Load panel graphical symbols
- 131** Load panel labels/names
- 132** Load panels : display the additional symbols for abutting wall or parapet (snow) / edges with return corner (wind)
- 133** Display trusses
- 134** Display beams
- 135** Display ribs
- 136** Isoline labels
- 137** Color legend : round the calculated values
- 138** Display springs
- 139** Display gaps
- 140** Concentrated load values
- 141** Line load values
- 142** Surface load values
- 143** Temperature load values
- 144** Self weight load values
- 145** Misc/other load values
- 146** Load panel local system
- 147** N/A
- 148** Thickness reference points
- 149** N/A
- 150** Display virtual beams
- 151** Display virtual beam strip width

- 152 Virtual beam labels/names
- 153 Virtual beam local system
- 154 Display fire loads
- 155 Fire load values
- 156 N/A
- 157 Transparent load diagrams
- 158 Prevent labels from overlapping
- 159 Transparent labels
- 160 Nodal coordinates
- 161 Detailed dimension lines for footings/foundations
- 162 Display edge hinges
- 163 Core/wall reinforcement labels
- 164 Display core/wall reinforcement
- 165 Masonry wall labels
- 166 Display masonry wall
- 167 Display structural member lateral support
- 168 Punching of wall end/wall corner
- 169 Support characteristics labels
- 170 Support stiffness values

Records / structures

RWriteValuesTo = (

<u>ELongBoolean</u>	Nodes	Show results for nodes
<u>ELongBoolean</u>	Lines	Show results for lines
<u>ELongBoolean</u>	Surfaces	Show results for surfaces
<u>ELongBoolean</u>	MinMaxOnly	Show min and max values only
) Show values next to these element types		

RBasicDisplayParameters = (

Warning! This record has become obsolete, it was superseded by RBasicDisplayParameters_V153

<u>EResultComponent</u>	ResultComponent	Result component
double	Scale	Scale of value on display
<u>EDisplayMode</u>	DisplayStyle	Display mode
<u>EDisplayShape</u>	DisplayShape	Display shape
<u>RWriteValuesTo</u>	WriteValuesTo	Write values to these types of elements
) Basic display parameters		

RBasicDisplayParameters_V153 = (

<u>EResultComponent</u>	ResultComponent	Result component
double	Scale	Scale of value on display
<u>EDisplayMode</u>	DisplayStyle	Display mode
<u>EDisplayShape</u>	DisplayShape	Display shape
<u>RWriteValuesTo</u>	WriteValuesTo	Write values to these types of elements
ELongBoolean	AutoScale	Adjusts the diagram scale automatically
) Basic display parameters		

RExtendedDisplayParameters = (

Warning! This record has become obsolete, it was superseded by RExtendedDisplayParameters_V153

<u>RBasicDisplayParameters</u>	BasicDispParams	Basic display parameters
<u>EDisplayAnalysisType</u>	DisplayAnalysisType	Analysis type of displayed results
<u>EResultType</u>	ResultType	Type of result to display
<u>EMinMaxType</u>	MinMaxType	Min or max value
<u>ECombinationType</u>	CriticalResCombinationType	For ResultType = rtEnvelope and rtCritical set combination type
<u>ELongBoolean</u>	SectPlaneContour	Contour of section plane is displayed if lbTrue
<u>EDisplayedEnvelopes</u>	DisplayedEnvelopes	Displayed envelopes
) Extended display parameters		

RBasicDisplayParameters_V1	RExtendedDisplayParameters_V153 = (
53	BasicDispParams <i>Basic display parameters</i>
EDisplayAnalysisType	DisplayAnalysisType <i>Analysis type of displayed results</i>
EResultType	ResultType <i>Type of result to display</i>
EMinMaxType	MinMaxType <i>Min or max value</i>
ECombinationType	CriticalResCombinationType <i>For ResultType = rtEnvelope and rtCritical set combination type</i>
ELongBoolean	SectPlaneContour <i>Contour of section plane is displayed if lbTrue</i>
EDisplayedEnvelopes	DisplayedEnvelopes <i>Displayed envelopes</i>
) Extended display parameters
	RShowGraphicSymbols = (
ELongBoolean	Mesh <i>Show mesh</i>
ELongBoolean	Node <i>Show node</i>
ELongBoolean	SurfaceCentre <i>Show surface centre</i>
ELongBoolean	CentreOfCircle <i>Show centre of circle</i>
ELongBoolean	Domain <i>Show domains</i>
ELongBoolean	NodalSupport <i>Show nodal supports</i>
ELongBoolean	LineSupport <i>Show line supports</i>
ELongBoolean	SurfaceSupport <i>Show surface support</i>
ELongBoolean	Foundation <i>Show foundations</i>
ELongBoolean	AutoFoundationDimension <i>Show foundation dimensions automatically</i>
ELongBoolean	Links <i>Show links</i>
ELongBoolean	Rigids <i>Show rigids</i>
ELongBoolean	Diaphragm <i>Show diaphragms</i>
ELongBoolean	Reference <i>Show references</i>
ELongBoolean	CrossSectionShape <i>Show cross section shapes</i>
ELongBoolean	EndReleases <i>Show end releases</i>
ELongBoolean	StructuralMembers <i>Show structural members</i>
ELongBoolean	ReinfParams <i>Show reinforcement parameters</i>
ELongBoolean	ReinfDomain <i>Show reinforcement of domains</i>
ELongBoolean	Mass <i>Show mass</i>
ELongBoolean	StoreyCentGrav <i>Show storey's centre of gravity</i>
ELongBoolean	StoreyShearCent <i>Show storey's centre of shear</i>
ELongBoolean	ARBO_CRETElems <i>Show ARBO/CRET elements</i>
ELongBoolean	COBIAXelems <i>Show COBIAX elements</i>
ELongBoolean	Trusses <i>Show truss elements</i>
ELongBoolean	Beams <i>Show beam elements</i>
ELongBoolean	Ribs <i>Show rib elements</i>
ELongBoolean	Springs <i>Show spring elements</i>
ELongBoolean	IsolineLabels <i>Show labels on isolines</i>
ELongBoolean	RoundIsoValues <i>Round values of isolines and isoareas</i>
ELongBoolean	Gaps <i>Show gap elements</i>
ELongBoolean	StructuralGrids <i>Show structural grids</i>
) Shown graphical symbol settings
	RShowLocalSystems = (
ELongBoolean	Beam <i>Show local coordinate systems of beams</i>
ELongBoolean	Rib <i>Show local coordinate systems of ribs</i>
ELongBoolean	Surface <i>Show local coordinate systems of surfaces</i>
ELongBoolean	Domain <i>Show local coordinate systems of domains</i>
ELongBoolean	Support <i>Show local coordinate systems of supports</i>
ELongBoolean	Spring <i>Show local coordinate systems of springs</i>
ELongBoolean	Gap <i>Show local coordinate systems of gaps</i>
ELongBoolean	Link <i>Show local coordinate systems of links</i>
ELongBoolean	EdgeHinge <i>Show local coordinate systems of edge hinges</i>
)
	Show local coordination systems by element types
	RShowLoads = (
ELongBoolean	Concentrated <i>Show concentrated loads</i>
ELongBoolean	Line <i>Show line loads</i>
ELongBoolean	Surface <i>Show surface loads</i>
ELongBoolean	Temperature <i>Show temperature loads</i>
ELongBoolean	SelfWeight <i>Show self weight</i>
ELongBoolean	Miscel <i>Show miscelaneous (other) loads</i>
ELongBoolean	LoadDistrScheme <i>Show load distribution scheme</i>
ELongBoolean	DerivedBeamLoad <i>Show derived beam loads</i>
ELongBoolean	MovingLoadPhases <i>Show moving load phases</i>
) Show loads by types
	RShowSymbols = (
RShowGraphicSymbols	ShowGraphicSymbols <i>Show these graphic symbols</i>
	584

<u>RShowLocalSystems</u>	ShowLocalSystems	<i>Show these local coordination systems</i>
<u>RShowLoads</u>	ShowLoads	<i>Show these loads</i>
<u>EBoolean</u>	ObjectContours3D	<i>Show 3D object contours</i>
)Shown symbols settings	

RShowNumbering = (

<u>EBoolean</u>	Show node numbers
<u>EBoolean</u>	Show truss numbers
<u>EBoolean</u>	Show beam numbers
<u>EBoolean</u>	Show rib numbers
<u>EBoolean</u>	Show surface numbers
<u>EBoolean</u>	Show domain numbers
<u>EBoolean</u>	Show support numbers
<u>EBoolean</u>	Show link numbers
<u>EBoolean</u>	Show rigid numbers
<u>EBoolean</u>	Show diaphragm numbers
<u>EBoolean</u>	Show spring numbers
<u>EBoolean</u>	Show gap numbers
<u>EBoolean</u>	Show material numbers
<u>EBoolean</u>	Show cross section numbers
<u>EBoolean</u>	Show reference numbers
<u>ARBO_CRETelems</u>	Show arbo/cret element numbers
<u>EBoolean</u>	Show design group numbers
<u>EBoolean</u>	Show optimisation group numbers

)Shown numbers by element types

RShowProperties = (

<u>EBoolean</u>	Show material name
<u>EBoolean</u>	Show cross section name
<u>EBoolean</u>	Show bolted joints
<u>EBoolean</u>	Show column reinforcement
<u>EBoolean</u>	Show beam lengths
<u>EBoolean</u>	Show domain(surface) thickness
<u>EBoolean</u>	Show domain area
<u>EBoolean</u>	Show COBIAX labels
<u>EBoolean</u>	Show load value
<u>EBoolean</u>	Show mass value
<u>EBoolean</u>	Show units
<u>EBoolean</u>	Show value of concentrated loads
<u>EBoolean</u>	Show value of line loads
<u>EBoolean</u>	Show value of surface loads
<u>EBoolean</u>	Show value of temperature loads
<u>EBoolean</u>	Show value of self weight
<u>EBoolean</u>	Show value of all other types of loads

)

Shown properties

RShowActualReinforcement = (

<u>EBoolean</u>	Show Symbol of actual bottom reinforcement in x direction
<u>EBoolean</u>	Show Symbol of actual bottom reinforcement in y direction
<u>EBoolean</u>	Show Symbol of actual top reinforcement in x direction
<u>EBoolean</u>	Show Symbol of actual top reinforcement in y direction
<u>EBoolean</u>	Show Symbol of actual bottom reinforcement in x direction
<u>EBoolean</u>	Show Symbol of actual bottom reinforcement in y direction
<u>EBoolean</u>	Show Symbol of actual top reinforcement in x direction
<u>EBoolean</u>	Show Symbol of actual top reinforcement in y direction
<u>EBoolean</u>	Type of actual reinforcement label

EActualReinforcementLabelTypeEBooleanAccordResComponent

)

Show symbol/label of actual reinforcement

RShowLabels = (

<u>RShowNumbering</u>	Shown numbers by element types
<u>RShowProperties</u>	Shown properties
<u>RShowActualReinforcement</u>	Show symbol/label of actual reinforcement
<u>EBoolean</u>	Use finite element numbers. More info here .
<u>EBoolean</u>	Labels on lines seen from axis direction

)

Show numbering and properties of various elements

RInfoWindowSwitch = (
ELongBoolean	Show coordinates window
ELongBoolean	Show info window
ELongBoolean	Show colour coding window
ELongBoolean	Show colour legend window
)	
Show or hide windows	
RDisplaySwitch = (
ELongBoolean	Show Parts
ELongBoolean	Show NonVisiblePartsGreyed
ELongBoolean	Show Guidelines
ELongBoolean	Show StructuralGrid
)	
Display switch	
RShowSwitches = (
RInfoWindowSwitch	Show or hide windows
RDisplaySwitch	Display switch
)	
Show windows and display switch	
RCommonCriticalResultsSettings = (
ELongBoolean	Investigate all combinations
ECriticalCombinationFormula	Critical combination formula
ECombinationMethod	Combination method in persistent and transient des. situations
)	
Display setting for critical results	
RMiscellaneousSettings = (
ESmoothing	<i>Smoothing</i>
double	Maximum angle allowed between local z axes in degrees
double	Maximum angle allowed between local x axes in degrees
EIntensityReferenceValue	Intensity reference value
double	Used when IntensityRefVal= irvCustomVal
ELongBoolean	Show average results of supports per member
)	
Miscellaneous settings	
RCommonDisplayParameters = (
RCommonCriticalResultsSettings	Critical results settings where <i>ExtendedDisplayParameters.ResultType</i> = rtCritical
ELongBoolean	Cut moment peaks
ELongBoolean	Draw in plane
RMiscellaneousSettings	Miscelsettings
)	
Display parameters, which are common for all AxisVM tabs.	
RWorldRectangle = (
double	<i>Left</i> distance of left side of a rectangle from the origin
double	<i>Right</i> distance of right side of a rectangle from the origin
double	<i>Top</i> distance of top side of a rectangle from the origin
double	<i>Bottom</i> distance of bottom side of a rectangle from the origin
)	
This record may be used to specify a rectangle in order to set the model view.	

Functions

long	Duplicate ([in] long Index , [in] EWindowSplit WindowSplit , [in] double SplitPos)
	Index <i>Index of the window 0 < Index < IAxisVMWindows.Count</i>
	WindowSplit <i>Type of window splitting</i>
	SplitPos <i>Split position (ratio) of the window, valid number: 0,1 - 0,9</i>
	<i>Splits window to two. If successful returns IAxisVMWindows.Count, otherwise an error code (errDatabaseNotReady or errIndexOutOfBounds).</i>
long	GetCommonDisplayParameters ([i/o] RCommonDisplayParameters CommonDisplayParameters)
	CommonDisplayParameters <i>Display parameters which are common for all AxisVM tabs.</i>
	<i>Get common display parameters. If successful returns 1, otherwise an error code (errDatabaseNotReady).</i>
long	GetBucklingDisplayParameters ([in] long Index , [i/o] RExtendedDisplayParameters ExtendedDisplayParameters , [out] long LoadCaseOrCombinationId , [out] long ModeShapeId , [out] SAFEARRAY(long)* SectionIds)
	Warning! This function has become obsolete, was superseded by GetBucklingDisplayParameters_V153
	Index <i>Index of the window 0 < Index < IAxisVMWindows.Count</i>
	ExtendedDisplayParameters <i>Extended display parameters</i>
	LoadCaseOrCombinationId <i>Load case or combination index depending on ExtendedDisplayParameters.ResultType (only rtLoadCase and rtLoadCombination are valid types)</i>
	ModeShapeId <i>Index of the mode shape</i>
	SectionIds <i>Index of displayed sections</i>
	<i>Get display parameters of buckling results. If successful returns 1, otherwise an error code (errDatabaseNotReady or errIndexOutOfBounds).</i>
long	GetBucklingDisplayParameters_V153 ([in] long Index , [i/o] RExtendedDisplayParameters_V153 ExtendedDisplayParameters , [out] long LoadCaseOrCombinationId , [out] long ModeShapeId , [out] SAFEARRAY(long)* SectionIds)
	Index <i>Index of the window 0 < Index < IAxisVMWindows.Count</i>
	ExtendedDisplayParameters <i>Extended display parameters</i>
	LoadCaseOrCombinationId <i>Load case or combination index depending on ExtendedDisplayParameters.ResultType (only rtLoadCase and rtLoadCombination are valid types)</i>
	ModeShapeId <i>Index of the mode shape</i>
	SectionIds <i>Index of displayed sections</i>
	<i>Get display parameters of buckling results. If successful returns 1, otherwise an error code (errDatabaseNotReady or errIndexOutOfBounds).</i>
long	GetDisplayOptions ([in] long Index , [i/o] RShowSymbols ShowSymbols , [i/o] RShowLabels ShowLabels , [i/o] RShowSwitches ShowSwitches)
	Index <i>Index of the window 0 < Index < IAxisVMWindows.Count</i>
	ShowSymbols <i>Shown symbols settings</i>
	ShowLabels <i>Show numbering and properties of various elements</i>
	ShowSwitches <i>Show windows and display switch</i>
	<i>Get display options. If successful returns 1, otherwise an error code (errDatabaseNotReady or errIndexOutOfBounds). For a full low level access, or querying directly only a few switches, use the more universal Switch property instead.</i>

long	GetDefaultDisplayOptions ([i/o] RShowSymbols ShowSymbols , [i/o] RShowLabels ShowLabels , [i/o] RShowSwitches ShowSwitches)	
	ShowSymbols <i>Shown symbols settings</i>	
	ShowLabels <i>Show numbering and properties of various elements</i>	
	ShowSwitches <i>Show windows and display switch</i>	
	<i>Get default display options of AxisVM. If successful returns 1, otherwise an error code (errDatabaseNotReady).</i>	
long	GetDetectedLayerIDs ([in] long Index , [out] SAFEARRAY(long)* DetectedLayerIDs)	
	Index <i>Index of the window 0 < Index < IAxisVMWindows.Count</i>	
	DetectedLayerIDs <i>Indexes of layers which can be detected with mouse</i>	
	<i>Get indexes of detected layers. If successful returns Index, otherwise an error code (errDatabaseNotReady or errIndexOutOfBounds).</i>	
long	GetDynamicDisplayParameters ([in] long Index , [i/o] RExtendedDisplayParameters ExtendedDisplayParameters , [out] long DynLoadCaseOrEnvelopeId , [out] long TimeStepId , [out] SAFEARRAY(long)* SectionIds)	
	Warning! This function has become obsolete, was superseded by GetDynamicDisplayParameters_V153	
	Index <i>Index of the window 0 < Index < IAxisVMWindows.Count</i>	
	ExtendedDisplayParameters <i>Extended display parameters</i>	
	DynLoadCaseOrEnvelopeId <i>Dynamic load case index or envelope index depending on the ExtendedDisplayParameters.ResultType</i>	
	TimeStepId <i>Index of the time step</i>	
	SectionIds <i>Index of displayed sections</i>	
	<i>Get display parameters of dynamic results. If successful returns 1, otherwise an error code (errDatabaseNotReady or errIndexOutOfBounds).</i>	
long	GetDynamicDisplayParameters_V153 ([in] long Index , [i/o] RExtendedDisplayParameters_V153 ExtendedDisplayParameters , [out] long DynLoadCaseOrEnvelopeId , [out] long TimeStepId , [out] SAFEARRAY(long)* SectionIds)	
	Index <i>Index of the window 0 < Index < IAxisVMWindows.Count</i>	
	ExtendedDisplayParameters <i>Extended display parameters</i>	
	DynLoadCaseOrEnvelopeId <i>Dynamic load case index or envelope index depending on the ExtendedDisplayParameters.ResultType</i>	
	TimeStepId <i>Index of the time step</i>	
	SectionIds <i>Index of displayed sections</i>	
	<i>Get display parameters of dynamic results. If successful returns 1, otherwise an error code (errDatabaseNotReady or errIndexOutOfBounds).</i>	
long	GetLockedLayerIDs ([in] long Index , [out] SAFEARRAY(long)* LockedLayerIDs)	
	Index <i>Index of the window 0 < Index < IAxisVMWindows.Count</i>	
	LockedLayerIDs <i>Indexes of layers which are locked</i>	
	<i>Get indexes of locked layers. If successful returns Index, otherwise an error code (errDatabaseNotReady or errIndexOutOfBounds).</i>	

long	GetRCDesignDisplayParameters ([in] long Index, [i/o] RExtendedDisplayParameters ExtendedDisplayParameters, [out] long LoadCaseOrCombinationOrEnvelopeld, [out] SAFEARRAY(long)* SectionIds)										
Warning! This function has become obsolete, was superseded by GetRCDesignDisplayParameters_V153											
	<table border="0"> <tr> <td style="vertical-align: top; padding-right: 10px;">Index</td><td><i>Index of the window 0<Index<IAxisVMWindows.Count</i></td></tr> <tr> <td style="vertical-align: top; padding-right: 10px;">ExtendedDisplayParameters</td><td><i>Extended display parameters</i></td></tr> <tr> <td style="vertical-align: top; padding-right: 10px;">LoadCaseOrCombinationOrEnvelopeld</td><td><i>Load case,load combination or envelope index depending on the ExtendedDisplayParameters.ResultType</i></td></tr> <tr> <td style="vertical-align: top; padding-right: 10px;">SectionIds</td><td><i>Index of displayed sections</i></td></tr> <tr> <td style="vertical-align: top; padding-right: 10px;">ExtendedDisplayParameters</td><td><i>Extended display parameters</i></td></tr> </table>	Index	<i>Index of the window 0<Index<IAxisVMWindows.Count</i>	ExtendedDisplayParameters	<i>Extended display parameters</i>	LoadCaseOrCombinationOrEnvelopeld	<i>Load case,load combination or envelope index depending on the ExtendedDisplayParameters.ResultType</i>	SectionIds	<i>Index of displayed sections</i>	ExtendedDisplayParameters	<i>Extended display parameters</i>
Index	<i>Index of the window 0<Index<IAxisVMWindows.Count</i>										
ExtendedDisplayParameters	<i>Extended display parameters</i>										
LoadCaseOrCombinationOrEnvelopeld	<i>Load case,load combination or envelope index depending on the ExtendedDisplayParameters.ResultType</i>										
SectionIds	<i>Index of displayed sections</i>										
ExtendedDisplayParameters	<i>Extended display parameters</i>										
	<i>Get display parameters of RC design results. If successful returns 1, otherwise an error code (errDatabaseNotReady or errIndexOutOfBounds).</i>										
<hr/>											
long	GetRCDesignDisplayParameters_V153 ([in] long Index, [i/o] RExtendedDisplayParameters_V153 ExtendedDisplayParameters, [out] long LoadCaseOrCombinationOrEnvelopeld, [out] SAFEARRAY(long)* SectionIds)										
	<table border="0"> <tr> <td style="vertical-align: top; padding-right: 10px;">Index</td><td><i>Index of the window 0<Index<IAxisVMWindows.Count</i></td></tr> <tr> <td style="vertical-align: top; padding-right: 10px;">ExtendedDisplayParameters</td><td><i>Extended display parameters</i></td></tr> <tr> <td style="vertical-align: top; padding-right: 10px;">LoadCaseOrCombinationOrEnvelopeld</td><td><i>Load case,load combination or envelope index depending on the ExtendedDisplayParameters.ResultType</i></td></tr> <tr> <td style="vertical-align: top; padding-right: 10px;">SectionIds</td><td><i>Index of displayed sections</i></td></tr> <tr> <td style="vertical-align: top; padding-right: 10px;">ExtendedDisplayParameters</td><td><i>Extended display parameters</i></td></tr> </table>	Index	<i>Index of the window 0<Index<IAxisVMWindows.Count</i>	ExtendedDisplayParameters	<i>Extended display parameters</i>	LoadCaseOrCombinationOrEnvelopeld	<i>Load case,load combination or envelope index depending on the ExtendedDisplayParameters.ResultType</i>	SectionIds	<i>Index of displayed sections</i>	ExtendedDisplayParameters	<i>Extended display parameters</i>
Index	<i>Index of the window 0<Index<IAxisVMWindows.Count</i>										
ExtendedDisplayParameters	<i>Extended display parameters</i>										
LoadCaseOrCombinationOrEnvelopeld	<i>Load case,load combination or envelope index depending on the ExtendedDisplayParameters.ResultType</i>										
SectionIds	<i>Index of displayed sections</i>										
ExtendedDisplayParameters	<i>Extended display parameters</i>										
	<i>Get display parameters of RC design results. If successful returns 1, otherwise an error code (errDatabaseNotReady or errIndexOutOfBounds).</i>										
<hr/>											
long	GetStaticDisplayParameters ([in] long Index, [i/o] RExtendedDisplayParameters ExtendedDisplayParameters, [out] long LoadCaseOrCombinationOrEnvelopeld, [out] SAFEARRAY(long)* SectionIds)										
	Warning! This function has become obsolete, was superseded by GetRCDesignDisplayParameters_V153										
	<table border="0"> <tr> <td style="vertical-align: top; padding-right: 10px;">Index</td><td><i>Index of the window 0<Index<IAxisVMWindows.Count</i></td></tr> <tr> <td style="vertical-align: top; padding-right: 10px;">ExtendedDisplayParameters</td><td><i>Extended display parameters</i></td></tr> <tr> <td style="vertical-align: top; padding-right: 10px;">LoadCaseOrCombinationOrEnvelopeld</td><td><i>Load case,load combination or envelope index depending on the ExtendedDisplayParameters.ResultType</i></td></tr> <tr> <td style="vertical-align: top; padding-right: 10px;">SectionIds</td><td><i>Index of displayed sections</i></td></tr> </table>	Index	<i>Index of the window 0<Index<IAxisVMWindows.Count</i>	ExtendedDisplayParameters	<i>Extended display parameters</i>	LoadCaseOrCombinationOrEnvelopeld	<i>Load case,load combination or envelope index depending on the ExtendedDisplayParameters.ResultType</i>	SectionIds	<i>Index of displayed sections</i>		
Index	<i>Index of the window 0<Index<IAxisVMWindows.Count</i>										
ExtendedDisplayParameters	<i>Extended display parameters</i>										
LoadCaseOrCombinationOrEnvelopeld	<i>Load case,load combination or envelope index depending on the ExtendedDisplayParameters.ResultType</i>										
SectionIds	<i>Index of displayed sections</i>										
	<i>Get display parameters of (static) analysis results. If successful returns 1, otherwise an error code (errDatabaseNotReady or errIndexOutOfBounds).</i>										
<hr/>											
long	GetStaticDisplayParameters_V153 ([in] long Index, [i/o] RExtendedDisplayParameters_V153 ExtendedDisplayParameters, [out] long LoadCaseOrCombinationOrEnvelopeld, [out] SAFEARRAY(long)* SectionIds)										
	<table border="0"> <tr> <td style="vertical-align: top; padding-right: 10px;">Index</td><td><i>Index of the window 0<Index<IAxisVMWindows.Count</i></td></tr> <tr> <td style="vertical-align: top; padding-right: 10px;">ExtendedDisplayParameters</td><td><i>Extended display parameters</i></td></tr> <tr> <td style="vertical-align: top; padding-right: 10px;">LoadCaseOrCombinationOrEnvelopeld</td><td><i>Load case,load combination or envelope index depending on the ExtendedDisplayParameters.ResultType</i></td></tr> <tr> <td style="vertical-align: top; padding-right: 10px;">SectionIds</td><td><i>Index of displayed sections</i></td></tr> </table>	Index	<i>Index of the window 0<Index<IAxisVMWindows.Count</i>	ExtendedDisplayParameters	<i>Extended display parameters</i>	LoadCaseOrCombinationOrEnvelopeld	<i>Load case,load combination or envelope index depending on the ExtendedDisplayParameters.ResultType</i>	SectionIds	<i>Index of displayed sections</i>		
Index	<i>Index of the window 0<Index<IAxisVMWindows.Count</i>										
ExtendedDisplayParameters	<i>Extended display parameters</i>										
LoadCaseOrCombinationOrEnvelopeld	<i>Load case,load combination or envelope index depending on the ExtendedDisplayParameters.ResultType</i>										
SectionIds	<i>Index of displayed sections</i>										
	<i>Get display parameters of (static) analysis results. If successful returns 1, otherwise an error code (errDatabaseNotReady or errIndexOutOfBounds).</i>										

long **GetSteelDesignDisplayParameters** ([in] long **Index**, [i/o] **RExtendedDisplayParameters**
ExtendedDisplayParameters, [out] long **LoadCaseOrCombinationOrEnvelopeld**,
[out] SAFEARRAY(long)* **SectionIds**)

Warning! This function has become obsolete, was superseded by
GetSteelDesignDisplayParameters_V153

Index Index of the window

0<Index<IAxisVMWindows.Count

ExtendedDisplayParameters Extended display parameters

LoadCaseOrCombinationOrEnvelopeld Load case,load combination or envelope index
depending on the

ExtendedDisplayParameters.ResultType

SectionIds Index of displayed sections

Get display parameters of steel design results. If successful returns 1, otherwise an error code
(*errDatabaseNotReady* or *errIndexOutOfBounds*).

long **GetSteelDesignDisplayParameters_V153** ([in] long **Index**, [i/o]
RExtendedDisplayParameters_V153 **ExtendedDisplayParameters**, [out] long
LoadCaseOrCombinationOrEnvelopeld,
[out] SAFEARRAY(long)* **SectionIds**)

Index Index of the window

0<Index<IAxisVMWindows.Count

ExtendedDisplayParameters Extended display parameters

LoadCaseOrCombinationOrEnvelopeld Load case,load combination or envelope index
depending on the

ExtendedDisplayParameters.ResultType

SectionIds Index of displayed sections

Get display parameters of steel design results. If successful returns 1, otherwise an error code
(*errDatabaseNotReady* or *errIndexOutOfBounds*).

long **GetTimberDesignDisplayParameters** ([in] long **Index**, [i/o] **RExtendedDisplayParameters**
ExtendedDisplayParameters, [out] long **LoadCaseOrCombinationOrEnvelopeld**,
[out] SAFEARRAY(long)* **SectionIds**)

Warning! This function has become obsolete, was superseded by
GetTimberDesignDisplayParameters_V153

Index Index of the window

0<Index<IAxisVMWindows.Count

ExtendedDisplayParameters Extended display parameters

LoadCaseOrCombinationOrEnvelopeld Load case,load combination or envelope index
depending on the

ExtendedDisplayParameters.ResultType

SectionIds Index of displayed sections

Get display parameters of timber design results. If successful returns 1, otherwise an error code
(*errDatabaseNotReady* or *errIndexOutOfBounds*).

long **GetTimberDesignDisplayParameters_V153** ([in] long **Index**, [i/o]
RExtendedDisplayParameters_V153 **ExtendedDisplayParameters**, [out] long
LoadCaseOrCombinationOrEnvelopeld,
[out] SAFEARRAY(long)* **SectionIds**)

Index Index of the window

0<Index<IAxisVMWindows.Count

ExtendedDisplayParameters Extended display parameters

LoadCaseOrCombinationOrEnvelopeld Load case,load combination or envelope index
depending on the

ExtendedDisplayParameters.ResultType

SectionIds Index of displayed sections

Get display parameters of timber design results. If successful returns 1, otherwise an error code
(*errDatabaseNotReady* or *errIndexOutOfBounds*).

long **GetVibrationDisplayParameters** ([in] long **Index**, [i/o] **RExtendedDisplayParameters**
ExtendedDisplayParameters, [out] long **LoadCaseOrCombinationId**, [out] long **ModeShapeId**,
[out] SAFEARRAY(long)* **SectionIds**)

Warning! This function has become obsolete, was superseded by
GetVibrationDisplayParameters_V153

Index Index of the window

0<Index<IAxisVMWindows.Count

ExtendedDisplayParameters Extended display parameters

LoadCaseOrCombinationId Load case or combination index depending on the
[ExtendedDisplayParameters.ResultType](#)

ModeShapeId Index of the mode shape

SectionIds Index of displayed sections

Get display parameters of vibration results. If successful returns 1, otherwise an error code
([errDatabaseNotReady](#) or [errIndexOutOfBounds](#)).

long **GetVibrationDisplayParameters_V153** ([in] long **Index**, [i/o] **RExtendedDisplayParameters_V153**
ExtendedDisplayParameters, [out] long **LoadCaseOrCombinationId**, [out] long **ModeShapeId**,
[out] SAFEARRAY(long)* **SectionIds**)

Index Index of the window

0<Index<IAxisVMWindows.Count

ExtendedDisplayParameters Extended display parameters

LoadCaseOrCombinationId Load case or combination index depending on the
[ExtendedDisplayParameters.ResultType](#)

ModeShapeId Index of the mode shape

SectionIds Index of displayed sections

Get display parameters of vibration results. If successful returns 1, otherwise an error code
([errDatabaseNotReady](#) or [errIndexOutOfBounds](#)).

long **GetVisibleLayerIDs** ([in] long **Index**, [out] SAFEARRAY(long)* **VisibleLayerIDs**)

Index Index of the window. 0<Index<IAxisVMWindows.Count

VisibleLayerIDs Indexes of visible layers

Get indexes of visible layers. If successful number of visible AxisVM layers, otherwise an error code
([errDatabaseNotReady](#) or [errIndexOutOfBounds](#)).

long **GetVisibleStructuralGridIDs** ([in] long **Index**, [out] SAFEARRAY(long)* **VisibleStructuralGridIDs**)

Index Index of the window. 0<Index<IAxisVMWindows.Count

VisibleStructuralGridIDs Indexes of visible structural grids

Get indexes of visible structural grids. If successful returns number of enabled structural grids,
otherwise an error code ([errDatabaseNotReady](#) or [errIndexOutOfBounds](#)).

long **GetVisibleSectionIDs** ([in] long **Index**, [out] SAFEARRAY(long)* **SectionIDs**)

Index Index of the window. 0<Index<IAxisVMWindows.Count

SectionIDs Indexes of visible sections ([IAxisVMSections](#))

Get indexes of visible sections. If successful returns number of visible sections, otherwise an error
code ([errDatabaseNotReady](#) or [errIndexOutOfBounds](#)).

long **GetWindowDisplayPartUIDs** ([in] long **Index**, [out] SAFEARRAY(long)* **PartUIDs**)

Index Index of the window. 0<Index<IAxisVMWindows.Count

PartUIDs Unique indexes of displayed parts

Get unique indexes of displayed parts. If successful returns number of enabled parts, otherwise an
error code ([errDatabaseNotReady](#) or [errIndexOutOfBounds](#)).

long	GetWorldRectangle ([in] long Index , [i/o] RWorldRectangle WorldRectangle)	Index <i>Index of the window. 0<Index<IAxisVMWindows.Count</i>
		WorldRectangle <i>distance of sides of a rectangle from the origin used for model view</i>
<i>Get the distance of sides of a rectangle from the origin used for model view. If successful returns window index, otherwise an error code (errDatabaseNotReady or errIndexOutOfBounds).</i>		
long	ReDraw	<i>Redraws all windows in AxisVM. If successful returns number of windows, otherwise an error code (errDatabaseNotReady or errIndexOutOfBounds).</i>
<hr/>		
long	Remove ([in] long Index)	Index <i>Index of the window 0<Index<IAxisVMWindows.Count</i>
		<i>Deletes window. If successful returns Index, otherwise an error code (errDatabaseNotReady or errIndexOutOfBounds).</i>
<hr/>		
long	SaveWindowToBitmap ([in] long Index , [in] EWindowColourMode WindowColourMode , [in] BSTR FileName)	Index <i>Index of the window. 0<Index<IAxisVMWindows.Count</i>
		WindowColourMode <i>Colour mode</i>
		FileName <i>file name with full path including extension</i>
		<i>Save window to a bitmap file. If successful returns Index, otherwise an error code (errIndexOutOfBounds).</i>
<hr/>		
long	SaveWindowsToBitmap ([in] EWindowColourMode WindowColourMode , [in] BSTR FileName)	WindowColourMode <i>Colour mode</i>
		FileName <i>file name with full path including extension</i>
		<i>Save the whole desktop of AxisVM to a bitmap file. If successful returns 1.</i>
<hr/>		
long	SaveWindowToClipboard ([in] long Index , [in] EWindowColourMode WindowColourMode)	Index <i>Index of the window 0<Index<IAxisVMWindows.Count</i>
		WindowColourMode <i>Colour mode</i>
		<i>Save window to the clipboard. If successful returns Index, otherwise an error code (errIndexOutOfBounds).</i>
<hr/>		
long	SaveWindowsToClipboard ([in] EWindowColourMode WindowColourMode)	WindowColourMode <i>Colour mode</i>
		<i>Save the whole desktop of AxisVM to the clipboard. If successful returns 1.</i>
<hr/>		
long	SaveWindowToMetafile ([in] long Index , [in] BSTR FileName)	Index <i>Index of the window 0<Index<IAxisVMWindows.Count</i>
		FileName <i>file name with full path including extension</i>
		<i>Save window to a meta file. If successful returns Index, otherwise an error code (errIndexOutOfBounds).</i>
<hr/>		
long	SaveWindowsToMetafile ([in] BSTR FileName)	FileName <i>file name with full path including extension</i>
		<i>Save the whole desktop of AxisVM to a meta file. If successful returns 1.</i>
<hr/>		
long	SetCommonDisplayParameters ([i/o] RCommonDisplayParameters CommonDisplayParameters)	CommonDisplayParameters <i>Display parameters which are common for all AxisVM tabs.</i>
		<i>Set common display parameters. If successful returns 1, otherwise an error code (errDatabaseNotReady).</i>

long **SetBucklingDisplayParameters** ([in] long **Index**, [i/o] **RExtendedDisplayParameters** **ExtendedDisplayParameters**, [out] long **LoadCaseOrCombinationId**, [out] long **ModeShapeId**, [out] SAFEARRAY(long)* **SectionIds**)

Warning! This function has become obsolete, was superseded by **SetBucklingDisplayParameters_V153**

Index *Index of the window*

0 < Index < IAxisVMWindows.Count

ExtendedDisplayParameters *Extended display parameters*

LoadCaseOrCombinationId *Load case or combination index depending on [ExtendedDisplayParameters.ResultType](#)
(only rtLoadCase and rtLoadCombination are valid types)*

ModeShapeId *Index of the mode shape*

SectionIds *Index of displayed sections*

Set display parameters of buckling results. If successful returns 1, otherwise an error code ([errDatabaseNotReady](#), [weLoadCaseIdOutOfBounds](#), [weLoadCombinationIdOutOfBounds](#) or [errIndexOutOfBounds](#)).

long **SetBucklingDisplayParameters_V153** ([in] long **Index**, [i/o] **RExtendedDisplayParameters_V153** **ExtendedDisplayParameters**, [out] long **LoadCaseOrCombinationId**, [out] long **ModeShapeId**, [out] SAFEARRAY(long)* **SectionIds**)

Index *Index of the window*

0 < Index < IAxisVMWindows.Count

ExtendedDisplayParameters *Extended display parameters*

LoadCaseOrCombinationId *Load case or combination index depending on [ExtendedDisplayParameters.ResultType](#)
(only rtLoadCase and rtLoadCombination are valid types)*

ModeShapeId *Index of the mode shape*

SectionIds *Index of displayed sections*

Set display parameters of buckling results. If successful returns 1, otherwise an error code ([errDatabaseNotReady](#), [weLoadCaseIdOutOfBounds](#), [weLoadCombinationIdOutOfBounds](#) or [errIndexOutOfBounds](#)).

long **SetDefaultDisplayOptions** ([i/o] **RShowSymbols** **ShowSymbols**, [i/o] **RShowLabels** **ShowLabels**, [i/o] **RShowSwitches** **ShowSwitches**)

ShowSymbols *Shown symbols settings*

ShowLabels *Show numbering and properties of various elements*

ShowSwitches *Show windows and display switch*

Set default display options of AxisVM. If successful returns 1, otherwise an error code ([errDatabaseNotReady](#)).

long **SetDetectedLayerIDs** ([in] long **Index**, [i/o] SAFEARRAY(long)* **DetectedLayerIDs**)

Index *Index of the window*

0 < Index < IAxisVMWindows.Count

DetectedLayerIDs *Indexes of layers which can be detected with mouse*

Set indexes of detected layers. If successful returns Index, otherwise an error code ([errDatabaseNotReady](#) or [errIndexOutOfBounds](#)).

long	SetDisplayOptions ([in] long Index , [i/o] RShowSymbols ShowSymbols , [i/o] RShowLabels ShowLabels , [i/o] RShowSwitches ShowSwitches)
	Index <i>Index of the window 0<Index<IAxisVMWindows.Count</i>
	ShowSymbols <i>Shown symbols settings</i>
	ShowLabels <i>Show numbering and properties of various elements</i>
	ShowSwitches <i>Show windows and display switch</i>
	<i>Set display options. If successful returns 1, otherwise an error code (errDatabaseNotReady or errIndexOutOfBounds). For a full low level access, or modifying directly only a few switches, use the more universal Switch property instead.</i>
long	SetDynamicDisplayParameters ([in] long Index , [i/o] RExtendedDisplayParameters ExtendedDisplayParameters , [in] long DynLoadCaseOrEnvelopeId , [in] long TimeStepId , [in] SAFEARRAY(long)* SectionIds)
	Warning! This function has become obsolete, was superseded by SetDynamicDisplayParameters_V153
	Index <i>Index of the window 0<Index<IAxisVMWindows.Count</i>
	ExtendedDisplayParameters <i>Extended display parameters</i>
	DynLoadCaseOrEnvelopeId <i>Dynamic load case index or envelope index depending on the ExtendedDisplayParameters.ResultType (only rtLoadCase and rtEnvelope are valid types)</i>
	TimeStepId <i>Index of the time step</i>
	SectionIds <i>Index of displayed sections</i>
	<i>Set display parameters of dynamic results. If successful returns 1, otherwise an error code (errDatabaseNotReady, weLoadCaseIdOutOfBounds, weEnvelopeIdOutOfBounds or errIndexOutOfBounds).</i>
long	SetDynamicDisplayParameters_V153 ([in] long Index , [i/o] RExtendedDisplayParameters_V153 ExtendedDisplayParameters , [in] long DynLoadCaseOrEnvelopeId , [in] long TimeStepId , [in] SAFEARRAY(long)* SectionIds)
	Index <i>Index of the window 0<Index<IAxisVMWindows.Count</i>
	ExtendedDisplayParameters <i>Extended display parameters</i>
	DynLoadCaseOrEnvelopeId <i>Dynamic load case index or envelope index depending on the ExtendedDisplayParameters.ResultType (only rtLoadCase and rtEnvelope are valid types)</i>
	TimeStepId <i>Index of the time step</i>
	SectionIds <i>Index of displayed sections</i>
	<i>Set display parameters of dynamic results. If successful returns 1, otherwise an error code (errDatabaseNotReady, weLoadCaseIdOutOfBounds, weEnvelopeIdOutOfBounds or errIndexOutOfBounds).</i>
long	SetLockedLayerIDs ([in] long Index , [i/o] SAFEARRAY(long)* LockedLayerIDs)
	Index <i>Index of the window 0<Index<IAxisVMWindows.Count</i>
	LockedLayerIDs <i>Indexes of layers which are locked</i>
	<i>Set indexes of locked layers. If successful returns Index, otherwise an error code (errDatabaseNotReady or errIndexOutOfBounds).</i>

long	SetRCDesignDisplayParameters ([in] long Index , [i/o] RExtendedDisplayParameters ExtendedDisplayParameters, [in] long LoadCaseOrCombinationOrEnvelopeld , [in] SAFEARRAY(long)* SectionIds)								
	Warning! This function has become obsolete, was superseded by SetRCDesignDisplayParameters_V153								
	<table border="0"> <tr> <td style="vertical-align: top; padding-right: 10px;">Index</td><td><i>Index of the window</i> <i>0 < Index < IAxisVMWindows.Count</i></td></tr> <tr> <td style="vertical-align: top; padding-right: 10px;">ExtendedDisplayParameters</td><td><i>Extended display parameters</i></td></tr> <tr> <td style="vertical-align: top; padding-right: 10px;">LoadCaseOrCombinationOrEnvelopeld</td><td><i>Load case, load combination or envelope index depending on the</i> <i>ExtendedDisplayParameters.ResultType</i> <i>0 for rtCritical</i></td></tr> <tr> <td style="vertical-align: top; padding-right: 10px;">SectionIds</td><td><i>Index of displayed sections</i></td></tr> </table> <p><i>Set display parameters of RC design results. If successful returns 1, otherwise an error code (errDatabaseNotReady, weLoadCaseIdOutOfBounds, weLoadCombinationIdOutOfBounds, weEnvelopeIdOutOfBounds or errIndexOutOfBounds).</i></p>	Index	<i>Index of the window</i> <i>0 < Index < IAxisVMWindows.Count</i>	ExtendedDisplayParameters	<i>Extended display parameters</i>	LoadCaseOrCombinationOrEnvelopeld	<i>Load case, load combination or envelope index depending on the</i> <i>ExtendedDisplayParameters.ResultType</i> <i>0 for rtCritical</i>	SectionIds	<i>Index of displayed sections</i>
Index	<i>Index of the window</i> <i>0 < Index < IAxisVMWindows.Count</i>								
ExtendedDisplayParameters	<i>Extended display parameters</i>								
LoadCaseOrCombinationOrEnvelopeld	<i>Load case, load combination or envelope index depending on the</i> <i>ExtendedDisplayParameters.ResultType</i> <i>0 for rtCritical</i>								
SectionIds	<i>Index of displayed sections</i>								
long	SetRCDesignDisplayParameters_V153 ([in] long Index , [i/o] RExtendedDisplayParameters_V153 ExtendedDisplayParameters, [in] long LoadCaseOrCombinationOrEnvelopeld , [in] SAFEARRAY(long)* SectionIds)								
	<table border="0"> <tr> <td style="vertical-align: top; padding-right: 10px;">Index</td><td><i>Index of the window</i> <i>0 < Index < IAxisVMWindows.Count</i></td></tr> <tr> <td style="vertical-align: top; padding-right: 10px;">ExtendedDisplayParameters</td><td><i>Extended display parameters</i></td></tr> <tr> <td style="vertical-align: top; padding-right: 10px;">LoadCaseOrCombinationOrEnvelopeld</td><td><i>Load case, load combination or envelope index depending on the</i> <i>ExtendedDisplayParameters.ResultType</i> <i>0 for rtCritical</i></td></tr> <tr> <td style="vertical-align: top; padding-right: 10px;">SectionIds</td><td><i>Index of displayed sections</i></td></tr> </table> <p><i>Set display parameters of RC design results. If successful returns 1, otherwise an error code (errDatabaseNotReady, weLoadCaseIdOutOfBounds, weLoadCombinationIdOutOfBounds, weEnvelopeIdOutOfBounds or errIndexOutOfBounds).</i></p>	Index	<i>Index of the window</i> <i>0 < Index < IAxisVMWindows.Count</i>	ExtendedDisplayParameters	<i>Extended display parameters</i>	LoadCaseOrCombinationOrEnvelopeld	<i>Load case, load combination or envelope index depending on the</i> <i>ExtendedDisplayParameters.ResultType</i> <i>0 for rtCritical</i>	SectionIds	<i>Index of displayed sections</i>
Index	<i>Index of the window</i> <i>0 < Index < IAxisVMWindows.Count</i>								
ExtendedDisplayParameters	<i>Extended display parameters</i>								
LoadCaseOrCombinationOrEnvelopeld	<i>Load case, load combination or envelope index depending on the</i> <i>ExtendedDisplayParameters.ResultType</i> <i>0 for rtCritical</i>								
SectionIds	<i>Index of displayed sections</i>								
long	SetStaticDisplayParameters ([in] long Index , [i/o] RExtendedDisplayParameters ExtendedDisplayParameters, [in] long LoadCaseOrCombinationOrEnvelopeld , [in] SAFEARRAY(long)* SectionIds)								
	Warning! This function has become obsolete, was superseded by SetStaticDisplayParameters_V153								
	<table border="0"> <tr> <td style="vertical-align: top; padding-right: 10px;">Index</td><td><i>Index of the window</i> <i>0 < Index < IAxisVMWindows.Count</i></td></tr> <tr> <td style="vertical-align: top; padding-right: 10px;">ExtendedDisplayParameters</td><td><i>Extended display parameters</i></td></tr> <tr> <td style="vertical-align: top; padding-right: 10px;">LoadCaseOrCombinationOrEnvelopeld</td><td><i>Load case, load combination or envelope index depending on the</i> <i>ExtendedDisplayParameters.ResultType</i></td></tr> <tr> <td style="vertical-align: top; padding-right: 10px;">SectionIds</td><td><i>Index of displayed sections</i></td></tr> </table> <p><i>Set display parameters of (static) analysis results. If successful returns 1, otherwise an error code (errDatabaseNotReady, weLoadCaseIdOutOfBounds, weLoadCombinationIdOutOfBounds, weEnvelopeIdOutOfBounds or errIndexOutOfBounds).</i></p>	Index	<i>Index of the window</i> <i>0 < Index < IAxisVMWindows.Count</i>	ExtendedDisplayParameters	<i>Extended display parameters</i>	LoadCaseOrCombinationOrEnvelopeld	<i>Load case, load combination or envelope index depending on the</i> <i>ExtendedDisplayParameters.ResultType</i>	SectionIds	<i>Index of displayed sections</i>
Index	<i>Index of the window</i> <i>0 < Index < IAxisVMWindows.Count</i>								
ExtendedDisplayParameters	<i>Extended display parameters</i>								
LoadCaseOrCombinationOrEnvelopeld	<i>Load case, load combination or envelope index depending on the</i> <i>ExtendedDisplayParameters.ResultType</i>								
SectionIds	<i>Index of displayed sections</i>								

long **SetStaticDisplayParameters_V153** ([in] long **Index**, [i/o] [RExtendedDisplayParameters_V153](#)
ExtendedDisplayParameters, [in] long **LoadCaseOrCombinationOrEnvelopeld**,
[in] SAFEARRAY(long)* **SectionIds**)

Index *Index of the window*

0<Index<IAxisVMWindows.Count

Extended display parameters

LoadCaseOrCombinationOrEnvelopeld *Load case,load combination or envelope index*

depending on the

[ExtendedDisplayParameters.ResultType](#)

SectionIds *Index of displayed sections*

*Set display parameters of (static) analysis results. If successful returns 1, otherwise an error code
([errDatabaseNotReady](#), [weLoadCaseldOutOfBounds](#), [weLoadCombinationIdOutOfBounds](#),
[weEnvelopeldOutOfBounds](#) or [errIndexOutOfBounds](#)).*

long **SetSteelDesignDisplayParameters** ([in] long **Index**, [i/o] [RExtendedDisplayParameters](#)
ExtendedDisplayParameters, [in] long **LoadCaseOrCombinationOrEnvelopeld** ,
[in] SAFEARRAY(long)* **SectionIds**)

Warning! This function has become obsolete, was superseded by
[SetSteelDesignDisplayParameters_V153](#)

Index *Index of the window*

0<Index<IAxisVMWindows.Count

Extended display parameters

LoadCaseOrCombinationOrEnvelopeld *Load case,load combination or envelope index*

depending on the

[ExtendedDisplayParameters.ResultType](#)

SectionIds *Index of displayed sections*

*Set display parameters of steel design results. If successful returns 1, otherwise an error code
([errDatabaseNotReady](#), [weLoadCaseldOutOfBounds](#), [weLoadCombinationIdOutOfBounds](#),
[weEnvelopeldOutOfBounds](#) or [errIndexOutOfBounds](#)).*

long **SetSteelDesignDisplayParameters_V153** ([in] long **Index**, [i/o] [RExtendedDisplayParameters_V153](#)
ExtendedDisplayParameters, [in] long **LoadCaseOrCombinationOrEnvelopeld** ,
[in] SAFEARRAY(long)* **SectionIds**)

Index *Index of the window*

0<Index<IAxisVMWindows.Count

Extended display parameters

LoadCaseOrCombinationOrEnvelopeld *Load case,load combination or envelope index*

depending on the

[ExtendedDisplayParameters.ResultType](#)

SectionIds *Index of displayed sections*

*Set display parameters of steel design results. If successful returns 1, otherwise an error code
([errDatabaseNotReady](#), [weLoadCaseldOutOfBounds](#), [weLoadCombinationIdOutOfBounds](#),
[weEnvelopeldOutOfBounds](#) or [errIndexOutOfBounds](#)).*

long **SetTimberDesignDisplayParameters** ([in] long **Index**, [i/o] [RExtendedDisplayParameters](#)
ExtendedDisplayParameters, [in] long **LoadCaseOrCombinationOrEnvelopeld** ,
[in] SAFEARRAY(long)* **SectionIds**)

Warning! This function has become obsolete, was superseded by
[SetTimberDesignDisplayParameters_V153](#)

Index *Index of the window*

0<Index<IAxisVMWindows.Count

Extended display parameters

LoadCaseOrCombinationOrEnvelopeld *Load case,load combination or envelope index*

depending on the

[ExtendedDisplayParameters.ResultType](#)

SectionIds *Index of displayed sections*

*Set display parameters of timber design results. If successful returns 1, otherwise an error code
([errDatabaseNotReady](#), [weLoadCaseldOutOfBounds](#), [weLoadCombinationIdOutOfBounds](#),
[weEnvelopeldOutOfBounds](#) or [errIndexOutOfBounds](#)).*

long	SetTimberDesignDisplayParameters_V153 ([in] long Index , [i/o] RExtendedDisplayParameters_V153 ExtendedDisplayParameters , [in] long LoadCaseOrCombinationOrEnvelopeld , [in] SAFEARRAY(long)* SectionIds)
	<p>Index <i>Index of the window 0<Index<IAxisVMWindows.Count</i></p> <p>ExtendedDisplayParameters <i>Extended display parameters</i></p> <p>LoadCaseOrCombinationOrEnvelopeld <i>Load case,load combination or envelope index depending on the ExtendedDisplayParameters.ResultType</i></p> <p>SectionIds <i>Index of displayed sections</i></p> <p><i>Set display parameters of timber design results. If successful returns 1, otherwise an error code (errDatabaseNotReady, weLoadCaseIdOutOfBounds, weLoadCombinationIdOutOfBounds, weEnvelopeIdOutOfBounds or errIndexOutOfBounds).</i></p>
long	SetVibrationDisplayParameters ([in] long Index , [i/o] RExtendedDisplayParameters ExtendedDisplayParameters , [in] long LoadCaseOrCombinationId , [in] long ModeShapeld , [in] SAFEARRAY(long)* SectionIds)
	<p>Warning! This function has become obsolete, was superseded by SetVibrationDisplayParameters_V153</p> <p>Index <i>Index of the window 0<Index<IAxisVMWindows.Count</i></p> <p>ExtendedDisplayParameters <i>Extended display parameters</i></p> <p>LoadCaseOrCombinationId <i>Load case or combination index depending on the ExtendedDisplayParameters.ResultType (only rtLoadCase and rtLoadCombination are valid)</i></p> <p>ModeShapeld <i>Index of the mode shape</i></p> <p>SectionIds <i>Index of displayed sections</i></p> <p><i>Set display parameters of vibration results. If successful returns 1, otherwise an error code (errDatabaseNotReady, weLoadCaseIdOutOfBounds, weLoadCombinationIdOutOfBounds or errIndexOutOfBounds).</i></p>
long	SetVibrationDisplayParameters_V153 ([in] long Index , [i/o] RExtendedDisplayParameters_V153 ExtendedDisplayParameters , [in] long LoadCaseOrCombinationId , [in] long ModeShapeld , [in] SAFEARRAY(long)* SectionIds)
	<p>Index <i>Index of the window 0<Index<IAxisVMWindows.Count</i></p> <p>ExtendedDisplayParameters <i>Extended display parameters</i></p> <p>LoadCaseOrCombinationId <i>Load case or combination index depending on the ExtendedDisplayParameters.ResultType (only rtLoadCase and rtLoadCombination are valid)</i></p> <p>ModeShapeld <i>Index of the mode shape</i></p> <p>SectionIds <i>Index of displayed sections</i></p> <p><i>Set display parameters of vibration results. If successful returns 1, otherwise an error code (errDatabaseNotReady, weLoadCaseIdOutOfBounds, weLoadCombinationIdOutOfBounds or errIndexOutOfBounds).</i></p>
long	SetVisibleLayerIDs ([in] long Index , [i/o] SAFEARRAY(long)* VisibleLayerIDs)
	<p>Index <i>Index of the window 0<Index<IAxisVMWindows.Count</i></p> <p>VisibleLayerIDs <i>Indexes of visible layers</i></p> <p><i>Set indexes of visible layers. If successful returns Index, otherwise an error code (errDatabaseNotReady or errIndexOutOfBounds).</i></p>

long **SetVisibleSectionIDs** ([in] long **Index**, [i/o] SAFEARRAY(long)* **SectionIDs**)
 Index *Index of the window*
 0 < Index < IAxisVMWindows.Count
 SectionIDs *Indexes of visible sections ([IAxisVMSections](#))*

Set indexes of visible sections. If successful returns Index, otherwise an error code ([errDatabaseNotReady](#) or [errIndexOutOfBounds](#)).

long **SetVisibleStructuralGridIDs** ([in] long **Index**, [i/o] SAFEARRAY(long)* **VisibleStructuralGridIDs**)
 Index *Index of the window*
 0 < Index < IAxisVMWindows.Count
 VisibleStructuralGridIDs *Indexes of visible structural grids*
Set indexes of visible structural grids. If successful returns window's index, otherwise an error code ([errDatabaseNotReady](#) or [errIndexOutOfBounds](#)).

long **SetWindowDisplayPartUIDs** ([in] long **Index**, [out] SAFEARRAY(long)* **PartUIDs**)
 Index *Index of the window*
 0 < Index < IAxisVMWindows.Count
 PartUIDs *Unique indexes of displayed parts*
Set unique indexes of displayed parts. If successful returns number of enabled parts, otherwise an error code ([errDatabaseNotReady](#) or [errIndexOutOfBounds](#)).

long **SetWorldRectangle** ([in] long **Index**, [i/o] [RWorldRectangle](#) **WorldRectangle**)
 Index *Index of the window*
 0 < Index < IAxisVMWindows.Count
 WorldRectangle *distance of sides of a rectangle from the origin used for model view*
Set the distance of sides of a rectangle from the origin used for model view. If successful returns window index, otherwise an error code ([errDatabaseNotReady](#) or [errIndexOutOfBounds](#)).

Properties

long **ActiveStoryIndex** • [long **Index**]
 Index *index of the window*
 Get or set actual story index in the window, 0 if storeys are disabled
[IAxisVMWindow](#) **ActiveWindow** • *Get the active window.*
 long **ActiveWindowIndex** • *Get or set index of the active window.*
 long **Count** *Get number of displayed windows.*
 long **LoadCaseIndex** • *Get or set displayed load case in all windows*
[EDisplay](#) **Display** • [long **Index**]
 Index *index of the window*
 Get or set display mode of the model in the window.
[IAxisVMWindow](#) **Item** [long **Index**]
 Index *index of the window*
 Get window interface by its index.
[EView](#) **View** • [long **Index**]
 Index *index of the window*
 Get or set view mode of the window.
 long **StoryIndex** • [long **Index**]
 Index *index of the window*
 Get or set actual story index of the actual work plane in the window.
 long **Switch** • [long **WindowIndex**, long **SwitchIndex**]
 WindowIndex *index of the window*
 SwitchIndex *index of the switch, see the [list of indexes](#)*

Get or set the value of a switch in the window. It provides full low level access to the underlying AxisVM structure. Can be seen as a replacement of the [GetDisplayOptions](#) and [SetDisplayOptions](#) functions.

long **WorkPlaneIndex** • [long **Index**]

Index *index of the window*

Get or set actual work plane index the window.

IAxisVMWindow

Used for changing display settings of one AxisVM window at a time

Functions

long **Duplicate** ([in] [EWindowSplit](#) WindowSplit, [in] double SplitPos)

See [Duplicate](#) of [IAxisVMWindows](#)

long **GetBucklingDisplayParameters** ([i/o] [RExtendedDisplayParameters](#) ExtendedDisplayParameters, [out] long LoadCaseOrCombinationId, [out] long ModeShapeId, [out] SAFEARRAY(long)* SectionIds)

Warning! This function has become obsolete, was superseded by
[GetBucklingDisplayParameters_V153](#)

See [GetBucklingDisplayParameters](#) of [IAxisVMWindows](#)

long **GetBucklingDisplayParameters_V153** ([i/o] [RExtendedDisplayParameters_V153](#) ExtendedDisplayParameters, [out] long LoadCaseOrCombinationId, [out] long ModeShapeId, [out] SAFEARRAY(long)* SectionIds)

See [GetBucklingDisplayParameters](#) of [IAxisVMWindows](#)

long **GetDisplayOptions** ([i/o] [RShowSymbols](#) ShowSymbols, [i/o] [RShowLabels](#) ShowLabels, [i/o] [RShowSwitches](#) ShowSwitches)

See [GetDisplayOptions](#) of [IAxisVMWindows](#)

long **GetDetectedLayerIDs** ([out] SAFEARRAY(long)* DetectedLayerIDs)

DetectedLayerIDs Indexes of layers which can be detected with mouse

Get indexes of detected layers. If successful returns window index, otherwise an error code ([errDatabaseNotReady](#)).

long **GetDynamicDisplayParameters** ([i/o] [RExtendedDisplayParameters](#) ExtendedDisplayParameters, [out] long DynLoadCaseOrEnvelopId, [out] long TimeStepId, [out] SAFEARRAY(long)* SectionIds)

Warning! This function has become obsolete, was superseded by
[GetDynamicDisplayParameters_V153](#)

See [SetDynamicDisplayParameters](#) of [IAxisVMWindows](#)

long **GetDynamicDisplayParameters_V153** ([i/o] [RExtendedDisplayParameters_V153](#) ExtendedDisplayParameters, [out] long DynLoadCaseOrEnvelopId, [out] long TimeStepId, [out] SAFEARRAY(long)* SectionIds)

See [SetDynamicDisplayParameters](#) of [IAxisVMWindows](#)

long **GetLockedLayerIDs** ([out] SAFEARRAY(long)* LockedLayerIDs)

LockedLayerIDs Indexes of locked layers

Get indexes of locked layers. If successful returns window index, otherwise an error code ([errDatabaseNotReady](#)).

long **GetRCDesignDisplayParameters** ([i/o] [RExtendedDisplayParameters](#) ExtendedDisplayParameters,

[out] long LoadCaseOrCombinationOrEnvelopId, [out] SAFEARRAY(long)* SectionIds)

Warning! This function has become obsolete, was superseded by
[GetRCDesignDisplayParameters_V153](#)

See [GetRCDesignDisplayParameters](#) of [IAxisVMWindows](#)

long **GetRCDesignDisplayParameters_V153** ([i/o] [RExtendedDisplayParameters_V153](#) ExtendedDisplayParameters,

[out] long LoadCaseOrCombinationOrEnvelopId, [out] SAFEARRAY(long)* SectionIds)

See [GetRCDesignDisplayParameters](#) of [IAxisVMWindows](#)

long	GetStaticDisplayParameters ([i/o] RExtendedDisplayParameters ExtendedDisplayParameters, [out] long LoadCaseOrCombinationOrEnvelopeld, [out] SAFEARRAY(long)* SectionIds) Warning! This function has become obsolete, was superseded by GetStaticDisplayParameters_V153 See GetStaticDisplayParameters of IAxisVMWindows
long	GetStaticDisplayParameters_V153 ([i/o] RExtendedDisplayParameters_V153 ExtendedDisplayParameters, [out] long LoadCaseOrCombinationOrEnvelopeld, [out] SAFEARRAY(long)* SectionIds) See GetStaticDisplayParameters of IAxisVMWindows
long	GetSteelDesignDisplayParameters ([i/o] RExtendedDisplayParameters ExtendedDisplayParameters, [out] long LoadCaseOrCombinationOrEnvelopeld, [out] SAFEARRAY(long)* SectionIds) Warning! This function has become obsolete, was superseded by GetSteelDesignDisplayParameters_V153 See GetSteelDesignDisplayParameters of IAxisVMWindows
long	GetSteelDesignDisplayParameters_V153 ([i/o] RExtendedDisplayParameters_V153 ExtendedDisplayParameters, [out] long LoadCaseOrCombinationOrEnvelopeld, [out] SAFEARRAY(long)* SectionIds) See GetSteelDesignDisplayParameters of IAxisVMWindows
long	GetTimberDesignDisplayParameters ([i/o] RExtendedDisplayParameters ExtendedDisplayParameters, [out] long LoadCaseOrCombinationOrEnvelopeld, [out] SAFEARRAY(long)* SectionIds) Warning! This function has become obsolete, was superseded by GetTimberDesignDisplayParameters_V153 See GetTimberDesignDisplayParameters of IAxisVMWindows
long	GetTimberDesignDisplayParameters_V153 ([i/o] RExtendedDisplayParameters_V153 ExtendedDisplayParameters, [out] long LoadCaseOrCombinationOrEnvelopeld, [out] SAFEARRAY(long)* SectionIds) See GetTimberDesignDisplayParameters of IAxisVMWindows
long	GetVibrationDisplayParameters ([i/o] RExtendedDisplayParameters ExtendedDisplayParameters, [out] long LoadCaseOrCombinationId, [out] long ModeShapeId, [out] SAFEARRAY(long)* SectionIds) Warning! This function has become obsolete, was superseded by GetTimberDesignDisplayParameters_V153 See SetVibrationDisplayParameters of IAxisVMWindows
long	GetVibrationDisplayParameters_V153 ([i/o] RExtendedDisplayParameters_V153 ExtendedDisplayParameters, [out] long LoadCaseOrCombinationId, [out] long ModeShapeId, [out] SAFEARRAY(long)* SectionIds) See SetVibrationDisplayParameters of IAxisVMWindows
long	GetVisibleLayerIDs ([out] SAFEARRAY(long)* VisibleLayerIDs) VisibleLayerIDs Indexes of visible layers Get indexes of visible layers. If successful returns window index, otherwise an error code (errDatabaseNotReady).
long	GetVisibleStructuralGridIDs ([out] SAFEARRAY(long)* VisibleStructuralGridIDs) VisibleStructuralGridIDs Indexes of visible structural grids Get indexes of visible structural grids. If successful returns window's index, otherwise an error code (errDatabaseNotReady).

long	GetPixelPosition ([i/o] RPoint3d p , [out] long Left , [out] long Top)	p global point coordinates Left left pixel coordinate Top top pixel coordinate	Get pixel position on the window from global point coordinates.Returns 1.
long	GetWindowDisplayPartUIDs ([out] SAFEARRAY(long)* PartUIDs)	PartUIDs Unique indexes of displayed parts	Get unique indexes of displayed parts. If successful returns number of enabled parts, otherwise an error code (errDatabaseNotReady or errIndexOutOfBounds).
long	GetWorldRectangle ([i/o] RWorldRectangle WorldRectangle)	WorldRectangle distance of sides of a rectangle from the origin used for model view	Get the distance of sides of a rectangle from the origin used for model view. If successful returns window index, otherwise an error code (errDatabaseNotReady or errIndexOutOfBounds).
long	PanToCoord ([i/o] RPoint3d Coord , [in] ElongBoolean MoveCursor)	Coord global point coordinates for paning (centering) the view MoveCursor move also the cursor to the coordinates	Pan (center) the view in window to global coordinates.Returns 1.
long	ReDraw	It redraws the window. If successful returns window index, otherwise an error code (errDatabaseNotReady or errIndexOutOfBounds).	
long	Remove	See Remove of IAxisVMWindows	
long	SaveWindowToBitmap ([in] EWindowColourMode WindowColourMode , [in] BSTR FileName)	WindowColourMode Colour mode FileName file name with full path including extension	Save window to a bitmap file. If successful returns Index, otherwise an error code (errIndexOutOfBounds).
long	SaveWindowToClipboard ([in] EWindowColourMode WindowColourMode)	WindowColourMode Colour mode	Save window to the clipboard. If successful returns Index, otherwise an error code (errIndexOutOfBounds).
long	SaveWindowToMetafile ([in] BSTR FileName)	FileName file name with full path including extension	Save window to a meta file. If successful returns Index, otherwise an error code (errIndexOutOfBounds).
long	SetBucklingDisplayParameters ([i/o] RExtendedDisplayParameters ExtendedDisplayParameters , [out] long LoadCaseOrCombinationId , [out] long ModeShapeId , [out] SAFEARRAY(long)* SectionIds)	Warning! This function has become obsolete, was superseded by SetBucklingDisplayParameters_V153	See SetBucklingDisplayParameters of IAxisVMWindows

long	SetBucklingDisplayParameters_V153 ([i/o] RExtendedDisplayParameters_V153 ExtendedDisplayParameters, [out] long LoadCaseOrCombinationId, [out] long ModeShapeId, [out] SAFEARRAY(long)* SectionIds) See SetBucklingDisplayParameters of IAxisVMWindows
long	SetDetectedLayerIDs ([i/o] SAFEARRAY(long)* DetectedLayerIDs) DetectedLayerIDs <i>Indexes of layers which can be detected by mouse</i> Set indexes of detected layers. If successful returns window index, otherwise an error code (errDatabaseNotReady). See SetDetectedLayerIDs of IAxisVMWindows
long	SetDisplayOptions ([i/o] RShowSymbols ShowSymbols, [i/o] RShowLabels ShowLabels, [i/o] RShowSwitches ShowSwitches) See SetDisplayOptions of IAxisVMWindows
long	SetDynamicDisplayParameters ([i/o] RExtendedDisplayParameters ExtendedDisplayParameters, [in] long DynLoadCaseOrEnvelopId, [in] long TimeStepId, [in] SAFEARRAY(long)* SectionIds) Warning! This function has become obsolete, was superseded by SetDynamicDisplayParameters_V153 See SetDynamicDisplayParameters of IAxisVMWindows
long	SetDynamicDisplayParameters_V153 ([i/o] RExtendedDisplayParameters_V153 ExtendedDisplayParameters, [in] long DynLoadCaseOrEnvelopId, [in] long TimeStepId, [in] SAFEARRAY(long)* SectionIds) See SetDynamicDisplayParameters of IAxisVMWindows
long	SetLockedLayerIDs ([i/o] SAFEARRAY(long)* LockedLayerIDs) LockedLayerIDs <i>Indexes of locked layers</i> Set indexes of locked layers. If successful returns window index, otherwise an error code (errDatabaseNotReady). See SetLockedLayerIDs of IAxisVMWindows
long	SetRCDesignDisplayParameters ([i/o] RExtendedDisplayParameters ExtendedDisplayParameters, [in] long LoadCaseOrCombinationOrEnvelopId, [in] SAFEARRAY(long)* SectionIds) Warning! This function has become obsolete, was superseded by SetRCDesignDisplayParameters_V153 See SetRCDesignDisplayParameters of IAxisVMWindows
long	SetRCDesignDisplayParameters_V153 ([i/o] RExtendedDisplayParameters_V153 ExtendedDisplayParameters, [in] long LoadCaseOrCombinationOrEnvelopId, [in] SAFEARRAY(long)* SectionIds) See SetRCDesignDisplayParameters of IAxisVMWindows
long	SetStaticDisplayParameters ([i/o] RExtendedDisplayParameters ExtendedDisplayParameters, [in] long LoadCaseOrCombinationOrEnvelopId, [in] SAFEARRAY(long)* SectionIds) Warning! This function has become obsolete, was superseded by SetStaticDisplayParameters_V153 See SetStaticDisplayParameters of IAxisVMWindows
long	SetStaticDisplayParameters_V153 ([i/o] RExtendedDisplayParameters_V153 ExtendedDisplayParameters, [in] long LoadCaseOrCombinationOrEnvelopId, [in] SAFEARRAY(long)* SectionIds) See SetStaticDisplayParameters of IAxisVMWindows

long	SetSteelDesignDisplayParameters ([i/o] RExtendedDisplayParameters ExtendedDisplayParameters, [in] long LoadCaseOrCombinationOrEnvelopeld, [in] SAFEARRAY(long)* SectionIds) Warning! This function has become obsolete, was superseded by SetSteelDesignDisplayParameters_V153 See SetSteelDesignDisplayParameters of IAxisVMWindows
long	SetSteelDesignDisplayParameters_V153 ([i/o] RExtendedDisplayParameters_V153 ExtendedDisplayParameters, [in] long LoadCaseOrCombinationOrEnvelopeld, [in] SAFEARRAY(long)* SectionIds) See SetSteelDesignDisplayParameters of IAxisVMWindows
long	SetTimberDesignDisplayParameters ([i/o] RExtendedDisplayParameters ExtendedDisplayParameters, [in] long LoadCaseOrCombinationOrEnvelopeld, [in] SAFEARRAY(long)* SectionIds) Warning! This function has become obsolete, was superseded by SetTimberDesignDisplayParameters_V153 See SetTimberDesignDisplayParameters of IAxisVMWindows
long	SetTimberDesignDisplayParameters_V153 ([i/o] RExtendedDisplayParameters_V153 ExtendedDisplayParameters, [in] long LoadCaseOrCombinationOrEnvelopeld, [in] SAFEARRAY(long)* SectionIds) See SetTimberDesignDisplayParameters of IAxisVMWindows
long	SetVibrationDisplayParameters ([i/o] RExtendedDisplayParameters ExtendedDisplayParameters, [in] long LoadCaseOrCombinationId, [in] long ModeShapeId, [in] SAFEARRAY(long)* SectionIds) Warning! This function has become obsolete, was superseded by SetVibrationDisplayParameters_V153 See SetVibrationDisplayParameters of IAxisVMWindows
long	SetVibrationDisplayParameters_V153 ([i/o] RExtendedDisplayParameters_V153 ExtendedDisplayParameters, [in] long LoadCaseOrCombinationId, [in] long ModeShapeId, [in] SAFEARRAY(long)* SectionIds) See SetVibrationDisplayParameters of IAxisVMWindows
long	SetVisibleLayerIDs ([i/o] SAFEARRAY(long)* VisibleLayerIDs) VisibleLayerIDs Indexes of visible layers Set indexes of visible layers. If successful returns window index, otherwise an error code (errDatabaseNotReady).
long	SetVisibleStructuralGridIDs ([i/o] SAFEARRAY(long)* VisibleStructuralGridIDs) VisibleStructuralGridIDs Indexes of visible structural grids Set indexes of visible structural grids. If successful returns window index, otherwise an error code (errDatabaseNotReady).
long	SetWindowDisplayPartUIDs ([out] SAFEARRAY(long)* PartUIDs) PartUIDs Unique indexes of displayed parts Set unique indexes of displayed parts. If successful returns number of enabled parts, otherwise an error code (errDatabaseNotReady or errIndexOutOfBounds).

long **SetWorldRectangle** ([*i/o*] [RWorldRectangle](#) **WorldRectangle**)

WorldRectangle *distance of sides of a rectangle from the origin used for model view*

Set the distance of sides of a rectangle from the origin used for model view. If successful returns window index, otherwise an error code ([errDatabaseNotReady](#) or [errIndexOutOfBounds](#)).

Properties

long **ActiveStoryIndex** • *Get or set actual story index in the window, 0 if storeys are disabled*

[EDisplay](#) **Display** • *Get or set display mode of the model in the window.*

long **Height** *Get height of the window.*

[EView](#) **View** • *Get or set view mode of the model in the window.*

long **Width** *Get width of the window.*

long **WorkplaneIndex** • *Get or set actual work plane index the window.*

long **StoryIndex** • *Get or set actual story index of the actual work plane in the window.*

[ElongBoolean](#) **ShowOnlySelected** • *Get or set whether selected elements are shown or not.*

NOTE: If already True the shown elements will be updated.

IAxisVMWorkplanes

Workplanes of the model

Enumerated types

enum	EGlobalWorkplaneType = {	
	gwptXY = 0x00,	<i>Global XY plane</i>
	gwptXZ = 0x01,	<i>Global XZ plane</i>
	gwptYZ = 0x02 }	<i>Global YZ plane</i>
<i>Global plane of the workplane</i>		
enum	ESmartWorkplaneElementType = {	
	swetMember = 0x00,	<i>XY plane made of member's local x and y axis</i>
	swetSurface = 0x01,	<i>XY plane made of surface's local x and y axis</i>
	swetDomain = 0x02 }	<i>XY plane made of domain's local x and y axis</i>
<i>Plane of an existing element (member, surface, domain)</i>		
enum	EWorkplaneType = {	
	wptGlobal = 0x00,	<i>Global work plane</i>
	wptSmart = 0x01,	<i>Smart work plane</i>
	wptGeneral = 0x02 }	<i>General work plane</i>
<i>Type of the work plane</i>		

Error codes

enum	EWorkplanesError = {	
	wpeInvalidName = -100001	<i>Name of the work plane is not valid</i>
	wpeNameAlreadyExists = -100002	<i>Work plane with the same name already exists in the model</i>
	wpeInvalidWorkPlaneParameters = -100003	<i>One or more parameters of the work plane are not valid</i>
	wpeWorkplanesNotGlobal = -100004	<i>Type of the subject workpane is not global</i>
	wpeWorkplanesNotSmart = -100005	<i>Type of the subject workpane is not smart</i>
	wpeWorkplanesNotGeneral = -100006	<i>Type of the subject workpane is not general</i>
	}	

Functions

long **AddGeneral** ([in] BSTR Name, [i/o] Rpoint3D Origin, [i/o] Rpoint3D LocalX,
[i/o] Rpoint3D LocalY, [in] Elongboolean HideNotInPlane,
[in] Elongboolean ShowGrayedNotInPlane)

Name	<i>Name of the new work plane</i>
Origin	<i>Coordinates of the origin</i>
LocalX	<i>Vector for work plane's x axis</i>
LocalY	<i>Vector for work plane's y axis</i>
HideNotInPlane	<i>Hide elements which are not in the work plane</i>
ShowGrayedNotInPlane	<i>Show elements which are not in the work plane grayed</i>

Adds new general type of work plane. If successful, returns the work plane index otherwise an error code ([errDatabaseNotReady](#), [wpeInvalidWorkPlaneParameters](#), [wpeNameAlreadyExists](#), [wpeInvalidName](#))

long **AddGlobal** ([in] BSTR Name, [in] EGlobalWorkplaneType GlobalWorkplaneType, [in] double PlaneOffset, [in] Elongboolean HideNotInPlane, [in] Elongboolean ShowGrayedNotInPlane)

Name	<i>Name of the new workplane</i>
GlobalWorkplaneType	<i>Global plane of the new work plane</i>
PlaneOffset	<i>Distance of the work planes's origin from global plane</i>
HideNotInPlane	<i>Hide elements which are not in the work plane</i>
ShowGrayedNotInPlane	<i>Show elements which are not in the work plane grayed</i>

Adds new global work plane. If successful, returns the work plane index otherwise an error code ([errDatabaseNotReady](#), [wpeInvalidWorkPlaneParameters](#), [wpeNameAlreadyExists](#), [wpeInvalidName](#))

long **AddSmart** ([in] BSTR Name, [in] ESmartWorkplaneElementType SmartWorkplaneElementType,
[in] long ElementIndex, [in] Elongboolean HideNotInPlane, [in] Elongboolean ShowGrayedNotInPlane)

Name	<i>Name of the new work plane</i>
SmartWorkplaneElementType	<i>Type of the element used for the work plane</i>
ElementIndex	<i>index of the element</i>
HideNotInPlane	<i>Hide elements which are not in the work plane</i>
ShowGrayedNotInPlane	<i>Show elements which are not in the work plane grayed</i>

Adds new global work plane.

If successful, returns the work plane index otherwise an error code ([errDatabaseNotReady](#), [wpeInvalidWorkPlaneParameters](#), [wpeNameAlreadyExists](#), [wpeInvalidName](#))

long **GetGeneralParameters** ([in] long Index, [i/o] Rpoint3D Origin, [i/o] Rpoint3D LocalX,
[i/o] Rpoint3D LocalY)

Index	<i>Index of the general work plane</i>
Origin	<i>Coordinates of the origin</i>
LocalX	<i>Vector for work plane's x axis</i>
LocalY	<i>Vector for work plane's y axis</i>

Get parameters of the general work plane. If successful, returns the work plane index otherwise an error code ([errDatabaseNotReady](#), [errIndexOutOfBounds](#), [wpeWorkplanesNotGeneral](#))

long **GetGlobalParameters** ([in] long Index, [out] EGlobalWorkplaneType GlobalWorkplaneType,
[out] double PlaneOffset)

Index	<i>Index of the global work plane</i>
GlobalWorkplaneType	<i>Global plane of the new work plane</i>
PlaneOffset	<i>Distance of the work planes's origin from global plane</i>

Get parameters of the global work plane. If successful, returns the work plane index otherwise an error code ([errDatabaseNotReady](#), [errIndexOutOfBounds](#), [wpeWorkplanesNotGlobal](#))

long	GetSmartParameters ([in] long Index , [out] ESmartWorkplaneElementType SmartWorkplaneElementType , [out] long ElementIndex)
	Index <i>Index of the global work plane</i>
	SmartWorkplaneElementType <i>Type of the element used for the work plane</i>
	ElementIndex <i>index of the element</i>
	<i>Get parameters of the smart work plane. If successful, returns the work plane index otherwise an error code (errDatabaseNotReady, errIndexOutOfBounds, wpeWorkplanesNotSmart)</i>
long	Delete ([in] long Index)
	Index <i>Index of the global work plane</i>
	<i>Delete the work plane. If successful, returns 1, otherwise an error code (errDatabaseNotReady, errIndexOutOfBounds)</i>
long	SetGeneralParameters ([in] long Index , [i/o] Rpoint3D Origin , [i/o] Rpoint3D LocalX , [i/o] Rpoint3D LocalY)
	Index <i>Index of the general work plane</i>
	Origin <i>Coordinates of the origin</i>
	LocalX <i>Vector for work plane's x axis</i>
	LocalY <i>Vector for work plane's y axis</i>
	<i>Set parameters of the general work plane. If successful, returns the work plane index otherwise an error code (errDatabaseNotReady, errIndexOutOfBounds, wpeWorkplanesNotGeneral)</i>
long	SetGlobalParameters ([in] long Index , [in] EGlobalWorkplaneType GlobalWorkplaneType , [in] double PlaneOffset)
	Index <i>Index of the global work plane</i>
	GlobalWorkplaneType <i>Global plane of the new work plane</i>
	PlaneOffset <i>Distance of the work planes's origin from global plane</i>
	<i>Get parameters of the global work plane. If successful, returns the work plane index otherwise an error code (errDatabaseNotReady, errIndexOutOfBounds, wpeInvalidWorkPlaneParameters, wpeWorkplanesNotGlobal)</i>
long	SetSmartParameters ([in] long Index , [in] ESmartWorkplaneElementType SmartWorkplaneElementType , [in] long ElementIndex)
	Index <i>Index of the global work plane</i>
	SmartWorkplaneElementType <i>Type of the element used for the work plane</i>
	ElementIndex <i>index of the element</i>
	<i>Set parameters of the smart work plane. If successful, returns the work plane index otherwise an error code (errDatabaseNotReady, errIndexOutOfBounds, wpeWorkplanesNotSmart, wpeInvalidWorkPlaneParameters)</i>

Properties

long	Count <i>Get number of work planes</i>
ELongBoolean	HideNotInPlane • [long Index]
	Index <i>index of the work plane</i>
	<i>Get or set whether elements which are not in the work plane are hidden or not</i>
long	Name [long Index]
	Index <i>index of the work plane</i>
	<i>Get name of the work plane.</i>
ELongBoolean	ShowGrayedNotInPlane • [long Index]
	Index <i>index of the work plane</i>
	<i>Get or set whether elements which are not in the work plane are shown greyed or not</i>
EWorkplaneType	WorkplaneType [long Index]
	Index <i>index of the work plane</i>
	<i>Get type of the work plane.</i>

IAxisVXMLAMpanels

XLAM panels used in the model

Error codes

```
enum EXLAMpanelsError = {
    xpeInvalidLayers = -100001           Must contain odd number of layers but minimum 3
    xpeThicknessesMustBePositive = -100002  Thicknesses should be symmetric
}
```

Functions

long	Add ([in] BSTR Name, [i/o] SAFEARRAY(double)* LayerThicknesses)	Name Name of the new XLAM panel LayerThicknesses Layer thicknesses, total number of thicknesses must be an odd number Adds new XLAM panel. If successful, returns the XLAM panel index otherwise an error code (see EGeneralErrors or EXLAMpanelsError).
long	AddFromCatalog ([in] BSTR Manufacturer, [in] BSTR Name)	Manufacturer Name of the XLAM panel manufacturer Name Name of the new XLAM panel Adds new XLAM panel from catalog. If successful, returns the XLAM panel index otherwise an error code (see EGeneralErrors or EXLAMpanelsError).
long	GetLayerThicknesses ([in] BSTR Index, [out] SAFEARRAY(double)* LayerThicknesses)	Index XLAM panel index LayerThicknesses Layer thicknesses If successful, returns the number of layers in the XLAM panel, otherwise an error code (see EGeneralErrors or EXLAMpanelsError).
long	Delete ([in] long Index)	Index Index of the XLAM panel Delete the XLAM panel. If successful, returns index, otherwise an error code (see EGeneralErrors or EXLAMpanelsError).
long	IndexOf ([in] BSTR Name)	Name Name of the XLAM panel Get index of the XLAM panel by name. If successful, returns index, otherwise an error code (see EGeneralErrors or EXLAMpanelsError).
long	ReplaceFromCatalog ([in] long Index), [in] BSTR Manufacturer, [in] BSTR Name)	Index Index of the XLAM panel Manufacturer Name of the XLAM panel manufacturer Name Name of the XLAM panel Replace XLAM panel from catalog. If successful, returns the XLAM panel index otherwise an error code (see EGeneralErrors or EXLAMpanelsError).

Properties

long	Count	Get number of XLAM panels
long	Name [long Index] Index	index of the XLAM panel Get name of the XLAM panel.

IAxisVMOBJECTCREATOR

Used for creating various interfaces, which then can be assigned to some properties of [IAxisVMModel](#) or used in parameters of some [IAxisVMCrossSections](#) interface functions *AddCustom*.

Functions

[IAxisVMLine2d](#) **NewLine2d**
Create new *IAxisVMLine2d* interface of the interface

[IAxisVMNewLines3d](#) **NewLines3d**
Create new *IAxisVMLines3d* interface of the interface

[IAxisVMMovingLoadOnBeam](#) **NewMovingLoadOnBeam**
Create new *IAxisVMMovingLoadOnBeam* interface of the interface

[IAxisVMMovingLoadOnDomain](#) **NewMovingLoadOnDomain**
Create new *IAxisVMMovingLoadOnDomain* interface of the interface

[IAxisVMPolygon2d](#) **NewPolygon2d**
Create new *IAxisVMPolygon2d* interface of the interface

[IAxisVMPolygon2dList](#) **NewPolygon2dList**
Create new *IAxisVMPolygon2dList* interface of the interface

IAxisVMLINE2D

A 2D line segment (straight line or arc) with orientation. This interface can be created with [ObjectCreator](#).

Enumerated types

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

enum **ELine2dPointIndex** = {
 piStart = 0x00, *startpoint*
 piEnd = 0x01 } *endpoint*
Point status.

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

Records / structures

double **RPoint2d** = (
 Coord1, **Coord2** *2D coordinates, units depend on type of usage*
)

Functions

void **GetCircleArcCenter** ([i/o] [RPoint2d](#) **Value**)

Value *circle arc center coordinates in metres.*

Obtains circle arc center coordinates. Value coordinates in metres.

void **GetLinePoints** ([i/o] [RPoint2d](#) **StartPoint**, [i/o] [RPoint2d](#) **EndPoint**)

StartPoint *startpoint of the line, coordinates in metres.*

EndPoint *endpoint of the line, coordinates in metres.*

Obtains line endpoints.

void	GetPoint ([in] ELine2dPointIndex Index , [i/o] RPoint2d Value)
	Index specifies the startpoint or the endpoint
	Value point coordinates in metres.
	<i>Obtains startpoint or endpoint coordinates.</i>
void	SetCircleArcCenter ([i/o] RPoint2d Value)
	Value circle arc center coordinates in metres.
	<i>Set circle arc center coordinates.</i>
void	SetLinePoints ([i/o] RPoint2d StartPoint , [i/o] RPoint2d EndPoint)
	StartPoint startpoint of the line, coordinates in metres.
	EndPoint endpoint of the line, coordinates in metres.
	<i>Set line endpoints</i>
void	SetPoint ([in] ELine2dPointIndex Index , [i/o] RPoint2d Value)
	Index specifies the startpoint or the endpoint
	EndPoint point coordinates in metres.
	<i>Set startpoint or endpoint coordinates.</i>

Properties

EArcAngleOrientation	CircleArcOrientation • (if LineType = ItCircleArc) Get or set orientation of the arc
ELine2dType	LineType • Get or set line type

IAxisVMLines3d

A list of three-dimensional line segments. IAxisVMLines3d can represent mesh-independent load polygons or polylines. This interface can be created with [ObjectCreator](#).

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 = {
    ltStraightLine3d = 0x00,     straight line
    ltCircleArc3d = 0x01 }       arc
    Line geometry.
```

Records / structures

```
RLine3d = (
    ELine3dType
    RPoint3d
    RPoint3d
    RPoint3d
    EArcAngleOrientation
    RPoint3d
)
RPoint3d = (
    double x, y, z
)
```

line geometry
startpoint
endpoint
(if LineType = ltCircleArc3d) center of the circle
(if LineType = ltCircleArc3d) orientation of the arc
(if LineType = ltCircleArc3d) arc plane normal

x, y, z coordinates of a a 3D point or components of a 3D vector [m]

Functions

long **Add ([i/o] 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.

EBoolean

Delete ([in] long Index)

Index index of the line to delete

Deletes a line from the list.

If successful, returns True, otherwise False.

void **GetItem ([in] long Index, [i/o] RLine3d Value)**

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

Obtains a line by index. If unsuccessful returns an error code
([errIndexOutOfBounds](#))

void **SetItem ([in] long Index, [i/o] RLine3d Value)**

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

Set a line by index. If unsuccessful returns an error code
([errIndexOutOfBounds](#))

Properties

long **Count**

Get number of 3D lines in the list

IAxisVMMovingLoadOnBeam

Moving loads on beams. This interface can be created with [ObjectCreator](#) then added with [IAxisVMMovingLoads](#) interface.

Enumerated types

enum	ERunningMode = {	
	rmOneWay = 0x00,	<i>Moving one way</i>
	rmRoundTrip = 0x01 }	<i>Moving around</i>
enum	EStructureMode = {	
	smChainrunway =	<i>Load group is applied at start and end</i>
	0x00,	
	smBridge = 0x01 }	<i>Load group is not applied at start and end</i>
		

Records / structures

ESystem	RConcentratedMovingLoadOnBeam = (
double	SystemGL	<i>coordinate system of load components (global or local only)</i>
	Position	<i>position of the point where load is applied, only the relative distance between positions in record matters, see AxisVM manual Moving loads for more info</i>
double	Fx, Fy, Fz	<i>x, y, z force components [kN]</i>
double	Mx, My, Mz	<i>moment components about the x, y, z axis [kNm]</i>
)	
EBeamRibDistributionType	RDistributedMovingLoadOnBeam = (
	SystemGL	<i>coordinate system of load components (global or local only)</i>
	DistributionType	<i>distributed by length or projected</i>
double	Position1	<i>position of the start point where the distributed load is applied, , see AxisVM manual Moving loads for more info</i>
double	Fx1, Fy1, Fz1	<i>x, y, z force components at the start point of the distributed load [kN]</i>
double	Position2	<i>position of the end point where the distributed load is applied, , see AxisVM manual Moving loads for more info</i>
double	Fx2, Fy2, Fz2	<i>x, y, z force components at the end point of the distributed load [kN]</i>
)	
ELoadType	RMovingLoadOnBeamItem = (
RConcentratedMovingLoadOnBeam	ItemType	<i>ItBeamConcentrated or ItBeamDistributed</i>
RDistributedMovingLoadOnBeam	Concentrated	<i>Parameters of concentrated moving load on beam</i>
	Distributed	<i>Parameters of distributed moving load on beam</i>
)	<i>Record is used for defining moving load on beam item</i>

Functions

long **AddItem** ([i/o] [RMovingLoadOnBeamItem](#) **MovingLoadOnBeamItem**)
MovingLoadOnBeamItem *Moving load on beam parameters*
Add moving load to the beam. If successful, returns moving load on beam index, otherwise an error code ([mleInvalidSystemValue](#), [mleInvalidItemType](#), [errDatabaseNotReady](#)).

long **DeleteItem** ([in] long **ItemIndex**)
ItemIndex *Index of the moving load on beam*
Delete moving load from the beam. If successful, returns item index, otherwise an error code ([errIndexOutOfBounds](#), [errDatabaseNotReady](#)).

long **GetItem** ([in] long **ItemIndex**, [i/o] **RMovingLoadOnBeamItem** **MovingLoadOnBeamItem**)

ItemIndex Index of the moving load on beam

MovingLoadOnBeamItem Moving load on beam parameters

Get moving load parameters of load item with ItemIndex. If successful, returns item index, otherwise an error code ([errIndexOutOfBounds](#), [errDatabaseNotReady](#), [mleInvalidMovingLoadType](#)).

long **GetPath** ([out] SAFEARRAY (long) * **Path**, [out] long **StartNode**, [out] long **EndNode**)

Path Line indexes

StartNode Start node index

EndNode End node index

Get path of the moving load on beam. If successful, returns number of lines of the path, otherwise an error code ([errDatabaseNotReady](#)).

long **SetItem** ([in] long **ItemIndex**, [i/o] **RMovingLoadOnBeamItem** **MovingLoadOnBeamItem**)

ItemIndex Index of the moving load on beam

MovingLoadOnBeamItem Moving load on beam parameters

Set moving load parameters on load with index ItemIndex. If successful, returns item index, otherwise an error code ([errIndexOutOfBounds](#), [errDatabaseNotReady](#)).

long **SetPath** ([in] SAFEARRAY (long) * **Path**, [in] long **StartNode**, [in] long **EndNode**)

Path Line indexes

StartNode Start node index

EndNode End node index

Set path of the moving load on beam. If successful, returns 1, otherwise an error code ([errDatabaseNotReady](#), [mleInvalidPathOrNodes](#)).

long **SetPath_vb** (Visual Basic compatible function of **SetPath**)

Properties

long **ItemCount** Get number of moving load items

[**ELoadType**](#) **ItemType** [long **Index**] Get type of moving load on beam

long **LoadCaseId** • Get or set loadcase index of moving load on beam

[**ERunningMode**](#) **RunningMode** • Get or set mode of run of moving load on beam

long **Steps** • Get or set number steps of moving loads on beams on path one way (minimum two steps). Total number of steps=steps x 2 for round trip

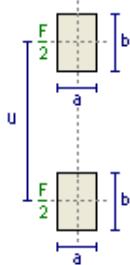
[**EStructureMode**](#) **StructureMode** • Get or set mode of structure of moving load on beam

IAxisVMMovingLoadOnDomain

Moving loads on beams. This interface can be created with [ObjectCreator](#) then added with [IAxisVMMovingLoads](#) interface.

Records / structures

RMovingLoadOnDomainItem = (
ESystem	double	coordinate system of load components
SystemGL		position of the point where load is applied, only the relative distance between positions of load items matters, see AxisVM manual Moving loads for more info
Position		
	double a	Width of the wheel [m]
	double b	depth of the wheel [m]
	double u	Vehicle gauge [m]
	double Fx, Fy, Fz	x, y, z load components of force F [kN]. Force is distributed equally onto two wheels
) Record is used for defining moving load on domain item



Functions

long **AddItem** ([i/o] [RMovingLoadOnDomainItem](#) **MovingLoadOnDomainItem**)

MovingLoadOnDomainItem Moving load on domain parameters

Add moving load to a domain acting at two points. If successful, returns load index, otherwise an error code ([mleInvalidSystemValue](#), [errDatabaseNotReady](#)).

long **DeleteItem** ([in] long **ItemIndex**)

ItemIndex Index of the moving load on domain

Delete a moving load item with index **ItemIndex**. If successful, returns item index, otherwise an error code ([errIndexOutOfBounds](#), [errDatabaseNotReady](#)).

long **GetItem** ([in] long **ItemIndex**, [i/o] [RMovingLoadOnDomainItem](#) **MovingLoadOnDomainItem**)

ItemIndex Index of the moving load on beam

MovingLoadOnDomainItem Moving load on domain parameters

Get parameters of the load item with **ItemIndex**. If successful, returns item index, otherwise an error code ([errIndexOutOfBounds](#), [errDatabaseNotReady](#), [mleInvalidMovingLoadType](#)).

long **GetPath** ([out] SAFEARRAY ([RPoint3D](#)) * **Path**, [out] SAFEARRAY ([RPoint3D](#)) * **NormV**)

Path Global coordinates of the path defined by points, (has always one more items than NormV array)

NormV Normal vectors of each segment, path segment is the line between two coordinates of the path

Get path of the moving load on a domain. If successful, returns number of path's coordinates, otherwise an error code ([errDatabaseNotReady](#), [errCOMServerInternalError](#)).

long	SetItem ([in] long ItemIndex , [i/o] RMovingLoadOnDomainItem MovingLoadOnDomainItem)
	ItemIndex <i>Index of the moving load on a domain</i>
	MovingLoadOnDomainItem <i>Moving load on a domain parameters</i>
	<i>Set moving load parameters of a load item with index ItemIndex. If successful, returns load item index, otherwise an error code (errIndexOutOfBounds, errDatabaseNotReady, mleInvalidSystemValue).</i>
long	SetPath ([in] SAFEARRAY (RPoint3D) * Path , [in] SAFEARRAY (RPoint3D) * NormV)
	Path <i>Global coordinates of the path defined by points, (has always one more items than NormV array)</i>
	NormV <i>Normal vectors of each path segment, path segment is the line between two points of the path</i>
	<i>Set path of the moving load acting at two points. If successful, returns 1, otherwise an error code (errDatabaseNotReady, mleInvalidPathOrNormV, mleInvalidNormVLength).</i>
long	SetPath_vb (Visual Basic compatible function of SetPath)

Properties

ELongBoolean	ConcentratedLoad <i>Get or set number of moving load is acting as point load at centre of wheel (No by default)</i>
long	ItemCount <i>Get number of moving load items</i>
long	LoadCaseId <i>Get or set loadcase index of moving load on domain</i>
ERunningMode	RunningMode <i>Get or set mode of run of moving load on domain</i>
long	Steps <i>Get or set number steps of moving load on path one way (minimum two steps). Total number of steps=steps x 2 for round trip</i>
EStructureMode	StructureMode <i>Get or set mode of structure of moving load on domain</i>

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's winding or its **Hole** property determines if it is a hole (opening) or a boundary. Polygon is closed, when the startpoint of the first line is the endpoint of the last line and the successive lines are connected. Line indexes run from 1 to N. This interface can be created with [ObjectCreator](#).

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.

[ELongBoolean](#)

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

Index index of the line to delete

Deletes a line from the polygon.

If successful, returns True, otherwise False.

Properties

[ELongBoolean](#)

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. Polygons which points are in counter clockwise order are denoting the outer shape and polygon's with points in clockwise order are denoting the opening (hole). Both polygons must be always closed. This interface can be created with [ObjectCreator](#).

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.

[ELongBoolean](#)

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

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.

Important notes:

Since several events might occur when model is changed, the client should only update it's own structure when [ModelChanged](#) or [FileChanged](#) events are invoked. Other events should be used as indicators about changes in the model.

When any of the Changed or Deleted event is invoked, the data structure of the model is changing, elements (nodes, lines, domains,...) are re-indexed so the COM client should not access the COM server during this process.

The list of AxisVM COM event interfaces:

IAxisVMAccelerationEvents

See [IAxisVMCrossSectionEditorEvents](#)

IAxisVMActualReinforcementEvents

See [IAxisVMCrossSectionsEvents](#)

IAxisVMAplicationEvents

Events:

void **ClientAliveTestCall**

When AxisVM is checking if client application is responding or not

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.

void **MainFormActivated**

When AxisVM application's window is activated

void **MainFormDeactivated**

When AxisVM application's window is deactivated

void **ModelClosed ([in] long Index, [in] BSTR FileName)**

Index *index of the model (always set to 1)*

FileName *File name*

Called when model is closed

void **ModelLoaded ([in] long Index)**

Index *index of the model (always set to 1)*

When AxisVM model is loaded

void **ModelSaved ([in] long Index, [in] BSTR OldFileName, [in] BSTR NewFileName))**

Index *index of the model (always set to 1)*

OldFileName *Old file name*

NewFileName *New file name*

Called when model is saved

```
void NewModel ([in] long Index)
    Index index of the list element
    New model has been created.
```

IAxisVMAttachmentsEvents

See [IAxisVMCrossSectionEditorEvents](#)

IAxisVMAtributesEvents

See [IAxisVMCrossSectionEditorEvents](#)

IAxisVMColumnRebarsEvents

See [IAxisVMCrossSectionEditorEvents](#)

IAxisVMCalculatedReinforcementEvents

See [IAxisVMCrossSectionEditorEvents](#)

IAxisVMCalculationEvents

Enumerated types:

```
enum ECalculatioFinishedType {
    cft_OK= 0
    cft_Canceled= 1
    cft_Warning= 2
    cft_Error= 3}
```

calculation finished without problems
calculation finished due to cancellation
calculation finished with warning(s)
calculation was interrupted due to error(s)

Events:

```
void MainProgress ([in] double Progress; [i/o] ELongBoolean Abort)
    Progress indicates the progress as a number between 0 and 1
    Abort if set to lbTrue, the calculation is aborted
```

*Called during the calculation to allow the user to display his own progress bar or abort the calculation. [CallMainProgress](#) property must be set to *lbTrue*.*

```
void Error ([in] long Index, [in] long ErrorCode)
    Index Not used
    ErrorCode the error code
```

Called in case of an error.

```
void Finished ([in] long Index, [in] EAnalysisType AnalysisType, [in] SAFEARRAY(VARIANT)*
    FinishTypes, [in] SAFEARRAY(VARIANT)* Messages, [in] SAFEARRAY(VARIANT)*
    LoadCombinationIds)
    Index Not used
    AnalysisType analysis type
    FinishTypes finish types /codes (long array)
    Note: for values see ECalculatioFinishedType
    Messages messages of warnings/errors if occurred (string array)
    LoadCombinationIds load combination or load case indexes where error/ warning occurred
    (long array)
    Called after the calculation (analysis) has finished.
```

IAxisVMCrackWidthEvents

See [IAxisVMCrossSectionEditorEvents](#)

IAxisVMCriticalGroupCombinationsEvents

Events:

void **Cleared**

All elements of the list have been cleared.

void **Changed**

An element of the list has been changed.

void **Error ([in] long Index, [in] long ErrorCode)**

Index *Not used*

ErrorCode *the error code*

Called in case of an error.

IAxisVMCrossSectionsEvents

Events:

void **Cleared**

All elements of the list have 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 Index, [in] long ErrorCode)**

Index *Not used*

ErrorCode *the error code*

Called in case of an error.

IAxisVMCrossSectionEditorEvents

Events:

void **Error ([in] long Index, [in] long ErrorCode)**

Index *Not used*

ErrorCode *the error code*

Called in case of an error.

IAxisVMCustomPartsEvents

See [IAxisVMCrossSectionEditorEvents](#)

IAxisVMCustomPartFolderEvents

See [IAxisVMCrossSectionEditorEvents](#)

IAxisVMDisplacementsEvents

See [IAxisVMCrossSectionEditorEvents](#)

IAxisVMDiaphragmEvents

See [IAxisVMCrossSectionEditorEvents](#)

IAxisVMDimensionsEvents

See [IAxisVMCrossSectionsEvents](#)

IAxisVMDomainsEvents

See [IAxisVMCrossSectionsEvents](#)

IAxisVMDomainSupportsEvents

See [IAxisVMCrossSectionsEvents](#)

IAxisVMDrawingsLibraryEvents

See [IAxisVMWindowsEvents](#)

IAxisVMDynamicLoadFunctionsEvents

See [IAxisVMCrossSectionsEvents](#)

IAxisVMEdgeConnectionsEvents

See [IAxisVMCrossSectionsEvents](#)

IAxisVMEvelopesEvents

See [IAxisVMCrossSectionEditorEvents](#)

IAxisVMForcesEvents

See [IAxisVMCrossSectionEditorEvents](#)

IAxisVMIcrementFunctionsEvents

See [IAxisVMCrossSectionsEvents](#)

IAxisVMLayersEvents

See [IAxisVMCrossSectionsEvents](#)

IAxisVMLinesEvents

Events:

void **AfterDeleted** ([in] SAFEARRAY(VARIANT)* **UIDs**)

An element of the list has been deleted. It is called after deletion and not in the meanwhile deletion like Deleted event. It is recommended to use it instead of Deleted.

void **Cleared**

All elements of the list have 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 Index, [in] long ErrorCode)
```

Index Not used

ErrorCode the error code

Called in case of an error.

IAxisVMLineSupportsEvents

See [IAxisVMCrossSectionsEvents](#)

IAxisVMLinkElementsEvents

See [IAxisVMCrossSectionsEvents](#)

IAxisVMLoadsEvents

See [IAxisVMCrossSectionEditorEvents](#)

IAxisVMLoadCasesEvents

See [IAxisVMCrossSectionsEvents](#)

IAxisVMLoadCombinationsEvents

See [IAxisVMCrossSectionsEvents](#)

IAxisVMLoadGroupsEvents

See [IAxisVMCrossSectionsEvents](#)

IAxisVMLogicalPartsEvents

See [IAxisVMCrossSectionEditorEvents](#)

IAxisVMMaterialsEvents

See [IAxisVMCrossSectionsEvents](#)

IAxisVMMathTextsEvents

```
void FillMathText ([in] BSTR MathTextUID, [out] BSTR API_Name,  
[out] BSTR MathTextTitle, [out] BSTR MathText)
```

MathTextUID unique index of the MathText

API_Name Name of the API (COM) client using MathTexts, must be the same string as used when the MathText was created. Must be assigned when handled!

MathTextTitle title shown in the report

MathText The math text shown in the report. This text overwrites math text used in [IAxisVMMathTexts.New](#) function.

Called whenever MathText is needed. When preview, export or print task is launched in AxisVM report.

Important: Check MathTextUID and handle only the MathTextUID created by your API (COM) client. Always set / assign the API_Name parameter.

void **ValidMathText** ([in] BSTR MathTextUID, [out] BSTR API_Name,
[out] BSTR MathTextTitle, [i/o] ELongBoolean Valid)
 MathTextUID unique index of the MathText
 API_Name Name of the API (COM) client using MathTexts, must be the same string
 as used when the MathText was created. Must be assigned when handled!
 MathTextTitle Title shown in the report. This text overwrites the title used in New function.
 Valid If lbFalse then the Title is shown red in the report
Called whenever MathText title is needed. When AxisVM report is opened and content is displayed.
Important: Check MathTextUID and handle only the MathTextUID created by your API (COM) client.

IAxisVMMembersEvents

Events:

void **AfterDeleted** ([in] SAFEARRAY(VARIANT)* UIDs)
An element of the list has been deleted. It is called after deletion and not during the deletion process like Deleted event. It is recommended to use this event rather than Deleted. See important notes in this section.

void **Cleared**
All elements of the list have 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 Index, [in] long ErrorCode)
 Index Not used
 ErrorCode the error code
Called in case of an error.

IAxisVMMODELSEvents

Events:

void **AfterMessageDisplay** ([in] BSTR Title, [in] BSTR Description, [in] EMessageDialogType MessageDialogType, [in] EMessageDialogButton ClickedButton)
 Title title of the displayed message
 Description displayed message
 MessageDialogType type of the message dialog
 ClickedButton code of clicked button
Called after showing a message (dialog box)

void **BeforeMessageDisplay** ([in] BSTR Title, [in] BSTR Description, [in] EMessageDialogType MessageDialogType, [in] long Buttons)
 Title title of the displayed message
 Description displayed message
 MessageDialogType type of the message dialog
 Buttons sum of button codes shown on dialog
Called before showing a message (dialog box)

void	Cleared	<i>All elements of the list have been cleared.</i>
void	Cleared	<p>Index <i>index of the list element to delete</i> <i>All elements of the list have been cleared.</i></p>
void	DisplayedErrors ([in] BSTR Title, [in] BSTR Description, [in] BSTR Errors)	<p>Title <i>Title of the error</i> Description <i>Description of the error</i> Errors <i>Error messages(s), contain a multiline strings (separated with CR+LF)</i> <i>Called before showing a form with errors</i></p>
void	Deleted ([in] long Index)	<p>Index <i>index of the list element to delete</i> <i>An element of the list has been deleted.</i></p>
void	Error ([in] long Index, [in] long ErrorCode)	<p>Index <i>index of the model (always set to 1)</i> ErrorCode <i>the error code</i> <i>Called in case of an error.</i></p>
void	MainProgress ([in] double Progress, [i/o] ELongBoolean Abort)	<p>Progress <i>indicates the progress as a number between 0 and 1</i> Abort <i>if set to lbTrue, the progres is aborted (does not work all the time)</i> <i>Called during any progress bar movement in status bar of AxisVM main form to allow the COM client to display own progress indicator. CallProgress property must be set to lbTrue.</i></p>
void	ModelClosed ([in] long Index, [in] BSTR FileName)	<p>Index <i>index of the model (always set to 1)</i> FileName <i>File name</i> <i>Called when model is closed</i></p>
void	ModelLoaded ([in] long Index, [in] ELongBoolean NewStatus)	<p>Index <i>index of the model (always set to 1)</i> NewStatus <i>True if new model</i> <i>Called when model is loaded</i></p>
void	ModelChanged ([in] long Index)	<p>Index <i>index of the model (always set to 1)</i> <i>Called after model has been changed, not called when only the view of the model has changed</i> <i>The FileChanged event is not invoked when this event is invoked.</i></p>
void	FileChanged ([in] long Index)	<p>Index <i>index of the model (always set to 1)</i> <i>Called after the file has changed, also when view has changed. Invoked after all operation affecting the model file, e.g.: If new cross-section or material is created, but not associated to any element, the file is changed but not the model.</i> <i>The ModelChanged event is not invoked when this event is invoked.</i></p>
void	ModelSaved ([in] long Index, [in] BSTR OldFileName, [in] BSTR NewFileName)	<p>Index <i>index of the model (always set to 1)</i> OldFileName <i>Old file name</i> NewFileName <i>New file name</i> <i>Called when model is saved</i></p>

void	NewModel ([in] long Index)	
	Index	<i>index of the list element</i>
		<i>New model has been created.</i>
void	SelectionProcessingChanged ([in] long Index , [in] ELongBoolean NewStatus)	
	Index	<i>index of the model (always set to 1)</i>
	NewStatus	<i>if True, the user is selecting elements if False, the user cannot select elements</i>
		<i>This procedure is called when selection toolbar appears and the user can select elements (NewStatus = lbTrue) or when the user ends selection and the toolbar disappears (NewStatus = lbFalse).</i>

IAxisVMMovingLoadsEvents

See [IAxisVMCrossSectionEditorEvents](#)

IAxisVMMovingLoadOnBeamEvents

See [IAxisVMCrossSectionEditorEvents](#)

IAxisVMMovingLoadOnDomainEvents

See [IAxisVMCrossSectionEditorEvents](#)

IAxisVMNodalSupportsEvents

See [IAxisVMCrossSectionsEvents](#)

IAxisVMNodesEvents

See [IAxisVMCrossSectionsEvents](#)

IAxisVMPushoverHingeFunctionsEvents

See [IAxisVMCrossSectionsEvents](#)

IAxisVMRCBeamDesignEvents

See [IAxisVMCrossSectionEditorEvents](#)

IAxisVMRCColumnCheckingEvents

See [IAxisVMCrossSectionEditorEvents](#)

IAxisVMRigidBodiesEvents

See [IAxisVMCrossSectionEditorEvents](#)

IAxisVMRebarSteelGradesEvents

See [IAxisVMCrossSectionsEvents](#)

IAxisVMReferencesEvents

See [IAxisVMCrossSectionsEvents](#)

IAxisVMReportsEvents

See [IAxisVMCrossSectionsEvents](#)

IAxisVMReinforcementCheckEvents

See [IAxisVMCrossSectionEditorEvents](#)

IAxisVMResultsEvents

See [IAxisVMCrossSectionEditorEvents](#)

IAxisVMSectionsEvents

See [IAxisVMCrossSectionsEvents](#)

IAxisVMSeismicStoreysEvents

See [IAxisVMCrossSectionsEvents](#)

IAxisVMSettingsEvents

Events:

void **Error** ([in] long **ErrorCode**)

ErrorCode the error code

Called in case of an erroneous setting.

IAxisVMShearCapacityEvents

See [IAxisVMCrossSectionEditorEvents](#)

IAxisVMSpectrumEvents

See [IAxisVMCrossSectionEditorEvents](#)

IAxisVMSpringParamsEvent

Events:

void **Error** ([in] long **Index**, [in] long **ErrorCode**)

Index Index of the item (0 if not applicable)

ErrorCode the error code

Called in case of an error.

IAxisVMSpringParamsEvents

Events:

void **Cleared**

All elements of the list have 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 **Index**, [in] long **ErrorCode**)

Index index of the item (0 if not applicable)

ErrorCode the error code

Called in case of an error.

IAxisVMSteelCrossSectionOptimizationEvents

Events:

void **Cleared**

All elements of the list have 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 Index, [in] long ErrorCode)**

Index *index of the model (always set to 1)*

ErrorCode *the error code*

Called in case of an error.

void **MainProgress ([in] double Progress; [i/o] ELongBoolean Abort)**

Progress *indicates the optimization progress as a number between 0 and 1*

Abort *if set to lbTrue, the optimization is aborted*

Called during the optimization progress to allow the user to display his own progress bar or abort the optimization. CallMainProgress property must be set to lbTrue.

IAxisVMSteelDesignMembersEvents

See [IAxisVMCrossSectionsEvents](#)

IAxisVMSteelDesignResultsEvents

See [IAxisVMCrossSectionEditorEvents](#)

IAxisVMStoreysEvents

See [IAxisVMCrossSectionsEvents](#)

IAxisVMSignificancesEvents

See [IAxisVMCrossSectionEditorEvents](#)

IAxisVMStructuralGridsEvents

See [IAxisVMCrossSectionsEvents](#)

IAxisVMSurfacesEvents

See [IAxisVMCrossSectionsEvents](#)

IAxisVMSurfaceSupportsEvents

See [IAxisVMCrossSectionsEvents](#)

IAxisVMTimberDesignMembersEvents

See [IAxisVMCrossSectionsEvents](#)

IAxisVMTimberDesignResultsEvents

See [IAxisVMCrossSectionEditorEvents](#)

IAxisVMTIMEINCREMENTFUNCTIONSEVENTS

See [IAxisVMCrossSectionsEvents](#)

IAxisVMVELOCITYEVENTS

See [IAxisVMCrossSectionEditorEvents](#)

IAxisVMVIRTUALBEAMSEVENTS

See [IAxisVMCrossSectionsEvents](#)

IAxisVMWINDOWSEVENTS

Events:

void **New** ([in] long **Index**, [in] [EWindowSplit](#) **WindowSplit**, [in] double **SplitPosition**)

Index new index of the window

WindowSplit type of window split

SplitPosition position of split

Called after a new window was created.

void **Deleted** ([in] long **Index**)

Index index of the window

Called after a window was deleted.

```
void Error ([in] long Index, [in] long ErrorCode)
    Index Not used
    ErrorCode the error code
    Called in case of an error.
```

IAxisVMWorkplanesEvents

Events:

```
void Created
    Called after a workplane was created.
void Deleted ([in] long Index)
    Index index of the workplane
    Called after a workplane was deleted.


---


void Error ([in] long Index, [in] long ErrorCode)
    Index Not used
    ErrorCode the error code
    Called in case of an error.
```

IAxisVMLAMpanelsEvents

See [IAxisVMWindowsEvents](#)

CustomFunction parameters

This function was created to allow for additional functions and features which would require version update of the COM server. Since version update would require in some cases rebuilding of applications developed by 3rd party developers, this function allows adding new functions which will be incorporated in the next version of the COM server.

The input string is in JSON format. Input JSON string parameter must have at least two fields: InterfaceName and FunctionName. These fields are used to identify the function.

Input JSON string example:

```
{"InterfaceName":"IAxisVMLoads","FunctionName":"ReApplyAllLoads"}
```

This input JSON parameter refers to a new to be fully implemented function *ReApplyAllLoads* which will be added to *IAxisVMLoads* interface.

The following interface and function names are handled in the latest release of AxisVM x4:

InterfaceName	FunctionName
---------------	--------------

IAxisVMAplication	GetHandles
--------------------------	-------------------

Returns two long type fields: MainFormHandle, ApplicationHandle

IAxisVMWindows	SetSteelDesignDisplayParameters
-----------------------	--

Same parameters as per function SetSteelDesignDisplayParameters with support for additional ResultComponent types in record RExtendedDisplayParameters.RBasicDisplayParameters:

10090: rc_sd_utilULS

10091: rc_sd_utilSLS

SetTimberDesignDisplayParameters

Same parameters as per function SetTimberDesignDisplayParameters with support for additional ResultComponent types in record RExtendedDisplayParameters.RBasicDisplayParameters:

10190: rc_td_UtilULS

10191: rc_td_UtilSLS

IAxisVMLoads	CheckAllLoads
---------------------	----------------------

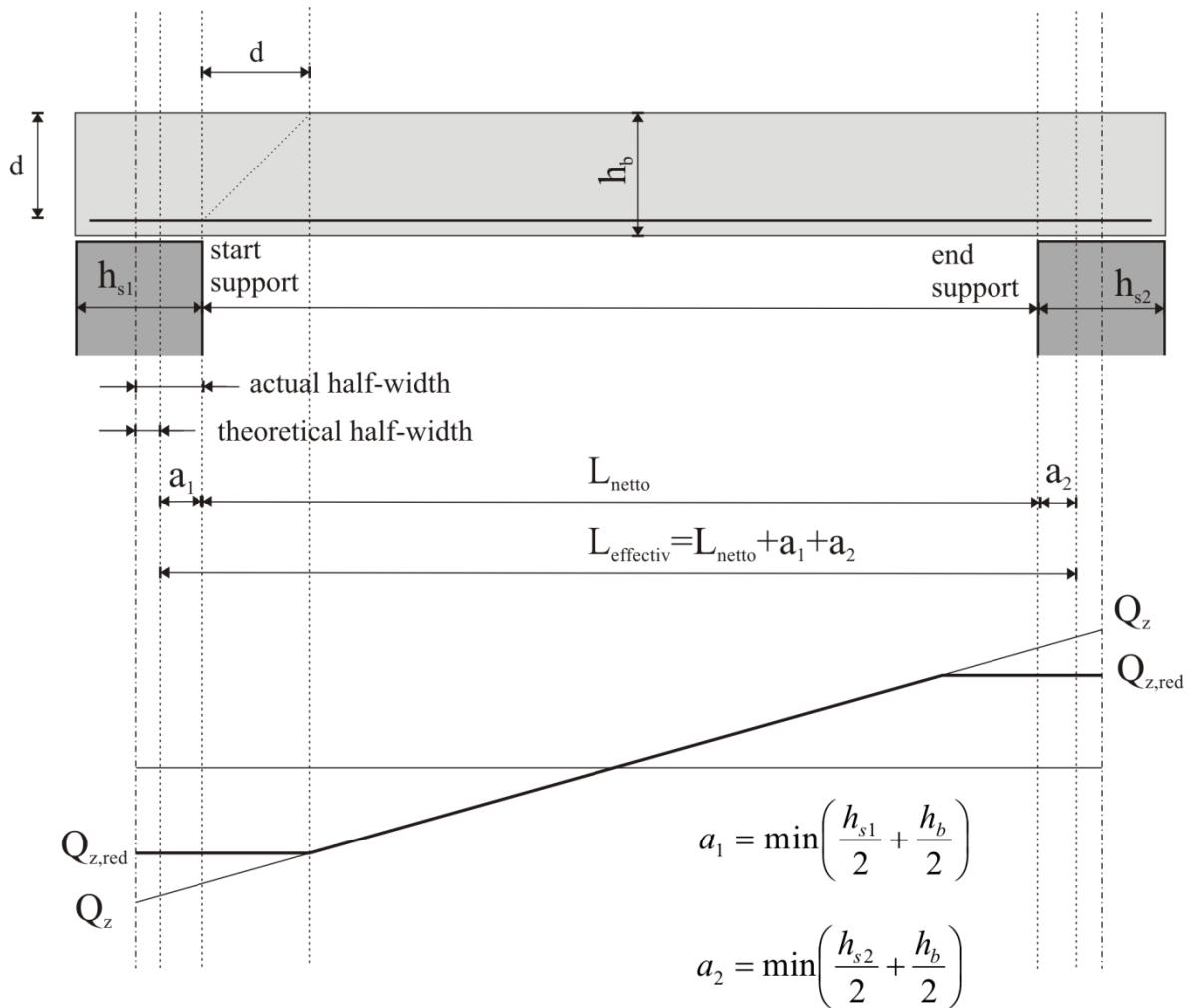
Check all loads in the model and delete the invalid or some of not applicable loads

ReApplyAllLoads

Regenerate and reapply associated loads, like loads applied on load panels etc.

Shear force reduction

The $Q_{z,red}$ is the shear force Q_z at distance d from face of the support. If the load is applied at the top of the beam, then part of the load between axis of the support and d distance from the face of the support is directly distributed to the support.



Eccentricity and loads on ribs

This section explains the implication and effects on the ribs due to difference of axis of ribs and midplane axis of the attached line (e.g. domain edge).

In case of eccentric ribs, the actual axis of the rib is not the same as the axis that appears in the finite element model: the rib can be assigned to a line (domain edge) of a domain midplane or to an independent line considering the eccentricity set by the user.

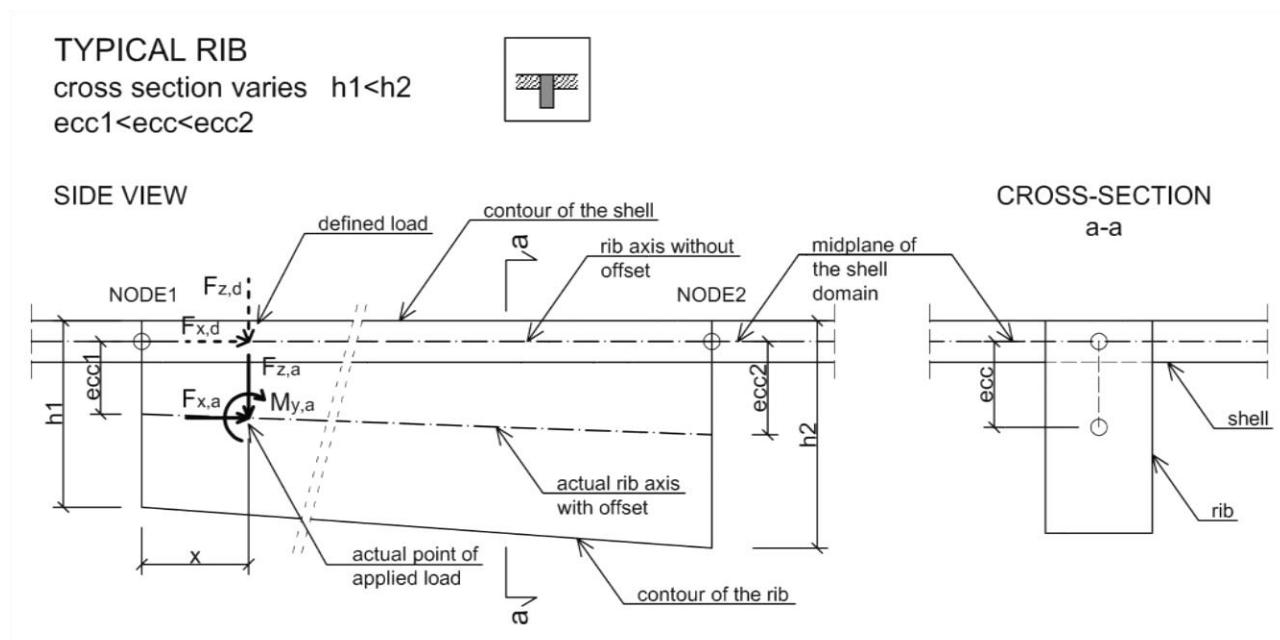
Nodal load can only be assigned to this line element (e.g domain edge), not to the actual axis of the rib.

The given load is interpreted according to this functioning, the bending moment of eccentric loads are also considered in the calculation.

The following figure shows two examples (typical and customized rib), where the derived bending moment of $F_{x,d}$ component is also taken into account using the formula $M_{y,a} = F_{x,d} \cdot e$, where "e" is the eccentricity of the rib. Index "d" indicates the load is defined by the user, index "a" refers to the load's actual point of application.

Typical rib:

In this case, the top of the rib is aligned to the attached slab. Furthermore the depth of cross-section varies at the ends of the rib (NODE1 and NODE2).



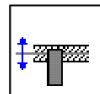
Customized rib:

In this case, the top of the rib is NOT aligned to anything, but the depth of cross-section varies at the ends.. Furthermore the defined eccentricity also varies on two ends of the rib (NODE1 and NODE2).

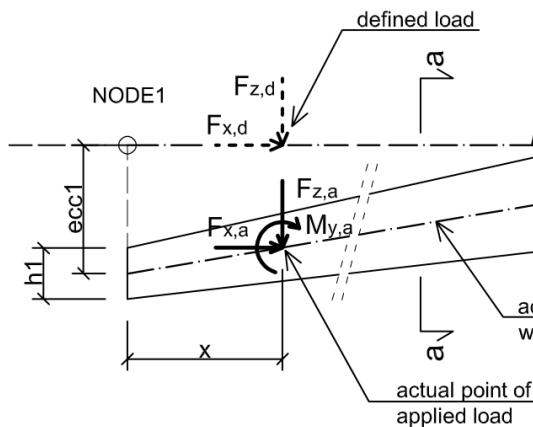
CUSTOMIZED RIB

cross section varies $h_1 < h_2$

$ecc_1 > ecc > ecc_2$

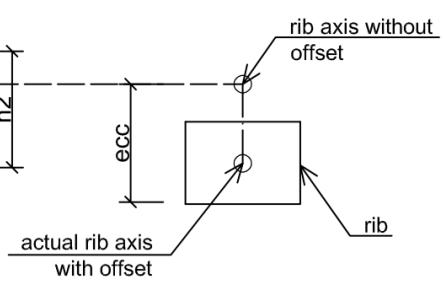


SIDE VIEW



CROSS-SECTION

a-a



Changes between versions 3.x and 5.0

[1] All in/out SAFEARRAY (VARIANT) parameters have been replaced with an equivalent SAFEARRAY (record type) parameter. (for .NET users)

[2] The pointer type SAFEARRAY (long) in functions GetLoad and SetLoad was changed to SAFEARRAY (byte).

Compatibility issues fixed.

[3] Some names of functions and properties have been renamed for consistency between AxisVM and COM server.

- Story -> Storey
- Stories -> Storeys
- Color -> Colour
- IAxisVMSeismicStories -> IAxisVMSeismicStoreys

[4] COM server now runs in Single Instance mode that means each COM client has its own COM server instance (only one COM client is connected to the same COM server at a time). Before COM server 5.0, each COM client connected to the same COM server instance so it was running in Multi Instance mode. It is possible to run AxisVM.exe in Multi Instance mode if client starts the AxisVM.exe itself first with '/MULTIINSTANCECOMCLIENTS' command line switch and after that starts the COM server instance by creating a COM object with the IAxisVMAApplication interface.

[5] Just like in previous versions COM clients must start AxisVM COM server by creating a COM object with the IAxisVMAApplication interface. Because of the Single Instance mode if COM clients creates additional COM objects with interfaces managed by AxisVM COM server then each COM object will start its separate COM server instances. To avoid this problem a new interface was implemented so COM server creates (without starting new COM server instance) these COM objects for the client. The new interface is the IAxisVMOBJECTCREATOR (a new IAxisVMAApplication property) and it can create COM objects with IAxisVMLINE2D, IAxisVMPOLYGON2D, IAxisVMPOLYGON2DLIST, IAxisVMLINES3D interfaces.

[6] Win32 Plugins:

- v1 plugins **are not** loaded
- v2 see [AxisVM COM plugin 2.0](#)
- Only one instance of AxisVM can run if the user wants to use a plugins. When multiple instances of AxisVM are running then COM client plugins are connected to first AxisVM instance (This applies to v1 plugins as well)

[7] .NET Plugins see new version [2.0](#)

Renamed for clarity and consistency:

- IAxisVMCrossSection interface:
 - ECrossSectionType -> ECrossSectionShape
 - CrossSectionType --> CrossSectionShape
 - AddRegularPolygon: v-> t
 - ReplaceWithRegularPolygon: v-> t
 - t -> tf
 - t2 -> tf2
 - v -> tw
 - v2 -> tw2

- IAxisVMCrossSections interface:
 - CrossSectionType -> CrossSectionShape
 - AddPipe : v-> t
 - ReplaceWithPipe: v-> t
 - AddRectangular: v-> h
 - ReplaceWithRectangular: v-> h
 - AddRegularPolygon: v-> t
 - ReplaceWithRegularPolygon: v-> t
- IAxisVMSurfaces interface:
 - GetSurfacesCoorinates -> GetAllCoordinatesOfSurfaces
- RLoadBeamInfluence record:
 - Dx,Dy,Dz -> ex,ey,ez
 - Rxx,Ryy,Rzz -> fx,fy,fz
- RLoadBeamStress record:
 - Force -> P
- IAxisVMMaterials and IAxisVMMaterial:
 - FillColor -> FillColour
 - ContourColor -> ContourColour
- IAxisVMSurfaces:
 - GetSurfacesCoorinates -> GetAllCoordinatesOfSurfaces
- IAxisVMDomains, IAxisVMLines, IAxisVMMembers, IAxisVMNodes
 - DeleteNameOfSelected**** ---> DeleteNameOfAll*****

New interfaces:

- IAxisVMDomainSupport
- IAxisVMDomainSupports
- IAxisVMStoreys
- IAxisVMSeismicStoreys
- IAxisVMTimberDesignMembers
- IAxisVMDynamicLoadFunctions
- IAxisVMTimeIncrementFunctions
- IAxisVMIcrementFunctions
- IAxisVMVelocity
- IAxisVMAcceleration
- IAxisVMRigidBodies
- IAxisVMDiaphragm
- IAxisVMOBJECTCreator

New functions:

- IAxisVMAplication
 - ChangeUnitSystem
- IAxisVMMaterials
 - AddFromDialog
- IAxisVMCrossSections
 - AddRectangular
 - ReplaceWithRectangular
 - AddFromDialog
 - AddFromEditor
 - Edit
- IAxisVMNodes
 - Check
 - SetDefaultNameForSelectedNodes

- IAxisVMLines
 - SetDefaultNameForSelectedNodes
- IAxisVMMembers
 - GenerateMeshWithParamsOnSelectedItems
 - SetDefaultNameForSelectedBeams
 - SetDefaultNameForSelectedRibs
- IAxisVMMember
 - GenerateMeshWithParams
 - CreateMeshWidthCoordinates
- IAxisVMDomains
 - GenerateMeshOnSelectedDomainsWithOriginalParams
 - SetDefaultNameForSelectedDomains
- IAxisVMDomain
 - GenerateCustomMesh
 - ModifyHole
 - SetElasticFoundation
 - GetElasticFoundation
 - DeleteElasticFoundation
- IAxisVMSurface
 - GetReinforcementParameters
- IAxisVMSteelDesignMembers
 - Add2
 - GetDesignParameters2
 - SetDesignParameters2
- IAxisVMLoads
 - AddDynamic
 - CreateStandardPushOverLoads
- IAxisVMLoadCases
 - CreatePushOverCases
- IAxisVMCalculation
 - DynamicAnalysis
 - PushOverAnalysis
- IAxisVMResults
 - GetModeActive
 - SetModeActive
 - GetSeismicEqCoeff
 - GetCapacityCurve

New properties:

- IAxisVMLine
 - MemberId

- IAxisVMDomain
 - ElasticFoundationExists
- IAxisVMModel
 - DomainSupports
 - Stories
 - SeismicStories
 - TimberDesignMembers
 - DynamicLoadFunctions
 - TimeIncrementFunctions
 - IncrementFunctions
 - RigidBodies
 - Diaphragm
 - DocumentLanguage
- IAxisVMResults
 - Velocity
 - Acceleration
 - TimeStepCount
- IAxisVMApplication
 - ObjectCreator
- IAxisVMSettings
 - ReportLanguage

Deleted functions:

- IAxisVMMember
 - GenerateMesh
- IAxisVMMembers
 - GenerateMeshOnSelectedItems

Changes between versions 5.0 and 5.1

Some functions have been renamed for clarity and consistency:

IAxisVMLoads interface:

- AddArea -> AddDomainPolyArea
- AddDomainArea -> AddDomainLinear
- AddDomainDistributed -> AddDomainConstant
- AddDomainPoly -> AddDomainPolyLine

Some records (structs) have been renamed for clarity and consistency:

- RLoadArea -> RLoadDomainPolyArea
- RLoadDomainArea -> RLoadDomainLinear
- RLoadDomainDistributed -> RLoadDomainConstant
- RLoadDomainPoly -> RLoadDomainPolyLine

Some fields in records (structs) have been renamed for clarity and consistency:

RLoadDomainLinear , RLoadDomainPolyArea:

- Coord11 -> x1
- Coord12 -> y1
- Coord13 -> z1
- Coord21 -> x2
- Coord22 -> y2
- Coord23 -> z2
- Coord31 -> x3
- Coord32 -> y3
- Coord33 -> z3

RLoadSupportDisplacement

- Fx, Fy, Fz -> ex, ey, ez
- Mx, My, Mz -> fx, fy, fz

RLoadTrussStress:

- Force-> P

RLoadSurfaceThermal , RLoadRibThermal, RLoadBeamThermal:

- Tsup -> Ttop
- Tinf -> Tbot

RLoadSurfaceEdge:

- qqx1 -> qx1
- qqx2 -> qx2
- qqx3 -> qx3
- qqx4 -> qx4
- qqy1 -> qy1
- qqy2 -> qy2
- qqy3 -> qy3
- qqy4 -> qy4
- qqz1 -> qz1
- qqz2 -> qz2
- qqz3 -> qz3
- qqz4 -> qz4

RLoadDomainConcentrated, RLoadSurfaceConcentrated:

- Fxc -> Fx
- Fyc -> Fy
- Fzc -> Fz
- Mxc -> Mx
- Myc -> My
- Mzc -> Mz
- Positionx -> x
- Positiony -> y
- Positionz -> z

Some properties have been renamed for clarity and consistency:

IAxisVMLoadGroup:

- GammaA -> GammaInf
- GammaF -> GammaSup

New interfaces:

- IAxisVMCrossSectionEditor
- IAxisVMRebarSteelGrades

New functions:

- IAxisVMSurface
 - SetReinforcementParameters
- IAxisVMCrossSections
 - AddCustomWithUserParams
 - ReplaceWithCustomAndUserParams
 - AddCustomWithUserParamsAsArray
 - ReplaceWithCustomAndUserParamsAsArray
- IAxisVMCrossSection
 - GetStressPointParams
 - SetStressPointParams
 - GetUserParams
 - SetUserParams
 - GetUserParamsAsArray
 - SetUserParamsAsArray
- IAxisVMLoadCombinations
 - GenerateAutoCombinations
- IAxisVMCatalog
 - GetRebarSteelGradeNames
 - GetMaterialNamesByType
- IAxisVMSpectrum
 - GetSpectrumData
 - SetSpectrumData

New properties:

- IAxisVMMaterial
 - RebarSteelGrades
- IAxisVMSpectrum
 - Parametric

Changes between versions 5.1 and 5.3

Some properties have been renamed for clarity and consistency:

IAxisVMCrossSection:

- Cy -> Ys
- Cz -> Zs

New interfaces:

- IAxisVMMovingLoads
- IAxisVMMovingLoadOnBeam
- IAxisVMMovingLoadOnDomain
- IAxisVMTimberDesignResults
- IAxisVMRCBeamDesign
- IAxisVMColumnRebars
- IAxisVMRCColumnChecking

New functions:

- IAxisVMOBJECTCREATOR
 - NewMovingLoadOnBeam
 - NewMovingLoadOnDomain
- IAxisVMLoads
 - AddBeamMovingLoad
 - AddDomainMovingLoad
- IAxisVMLINE
 - DeleteColumnReinforcementParameters
 - GetColumnReinforcementParameters
 - SetColumnReinforcementParameters
- IAxisVMMEMBER
 - DeleteColumnReinforcementParameters
 - GetColumnReinforcementParameters
 - SetColumnReinforcementParameters
- IAxisVMLoadCases
 - CreatePreStressCases

New properties:

- IAxisVMMODEL
 - MovingLoads
 - ColumnRebars
 - RCBeamDesign
 - RCColumnChecking
- IAxisVMRESULTS
 - TimberDesignResults
- IAxisVMLINE
 - ColumnReinforcementParametersExists
 - RigidBodyId
- IAxisVMMEMBER
 - ColumnReinforcementParametersExists
- IAxisVMLoadCases
 - LoadDurationClass
- IAxisVMAPLICATION:
 - ClientAliveTest
 - ClientAliveTestIntervalSec

Changes between versions 5.3 and 6.0

All records, enums and interfaces have new GUIDs!!!

Enumeration redefined:

- [ECombinationType](#) enum has changed completely!

Records redefined:

- RRCBeamReinforcementParameters –
 - **TopPos** and **BottomPos** sets concrete cover or rebar position depending on used design code!!!!

New functions:

- IAxisVMLoads:
 - AddRibMemberConcentrated
 - AddBeamMemberConcentrated
 - AddRibMemberDistributed
 - AddBeamMemberDistributed
- IAxisVMLoadCases
 - CreateImperfectionCase
 - GetPushOverParams
 - SetPushOverParams
 - GetImperfectionParams
 - SetImperfectionParams
- IAxisVMCrossSections
 - ReplaceFromCatalog
 - ReplaceFromCatalogFile

New properties:

- IAxisVMLine, IAxisVMMember, IAxisVMDomain & IAxisVMSurface
 - ArchitectElemType
 - ContourColor
 - MaterialColor
- IAxisVMNodes, IAxisVMMember, IAxisVMMembers, IAxisVMSurface, AxisVMSurfaces, AxisVMDomain & AxisVMDomains
 - UID

Changes between versions 6.0 and 6.1

Important changes in several functions parameters!

- ECriticalType enum replaced with ECombinationType
- ECriticalTypeBits replaced with ECombinationTypeBits
- **RResultBlock:**
 - *CriticalType replaced with CombinationType (ECriticalType enum)*
- RRCBeamReinforcementParameters –
 - **TopPos** and **BottomPos** sets concrete cover or rebar position depending on used design code!!!!

Property renamed:

IAxisVMMember

LineType -> MemberType

Enum value renamed:

IngGreek -> IngBrasilianPortuguese

New functions:

- IAxisVMMember
 - DefineAsTruss
 - DefineAsTimberTruss
 - GetTrussData
 - GetTimberTrussData
- IAxisVMMembers
 - DeleteNameOfAllTrusses
 - RenameSelectedTrusses

New properties:

- IAxisVMRebarSteelGrades
 - UID
- IAxisVMSurfaces, IAxisVMDomains, IAxisVMLoadCombinations, IAxisVMLoadCases, IAxisVMMembers, IAxisVMNodes, IAxisVMCrossSections, IAxisVMMaterials, IAxisVMRebarSteelGrades
 - IndexOfUID

New Visual Basic and VBA compatible functions:

- AxisVMActualReinforcement
 - AddDomainReinforcement_vb
 - AddPolygonReinforcement_vb
- AxisVMAApplication
 - MessageDlg_vb
- ColumnRebars:
 - Add_vb
 - SetRebars_vb
- AxisVMCrossSection
 - SetUserParamsAsArray_vb
 - SetUserParamsAsByteArray_vb
- AxisVMCrossSectionEditor
 - AddCustomWithUserParamsAsArray_vb
 - AddCustomWithUserParamsAsByteArray_vb

- AxisVMCrossSections
 - AddFromDialog_vb
 - AddCustomWithUserParamsAsArray_vb
 - ReplaceWithCustomAndUserParamsAsArray_vb
 - AddCustomWithUserParamsAsByteArray_vb
 - ReplaceWithCustomAndUserParamsAsByteArray_vb
- AxisVMDiaphragm
 - Add_vb
 - RemoveLinesFromDiaphragm_vb
- COMAxisVMDomain
 - AddHole_vb
 - Modify_vb
 - SetContourLines_vb
 - ModifyHole_vb
 - Get_MaterialColour_vb
 - Set_MaterialColour_vb
 - Get_ContourColour_vb
 - Set_ContourColour_vb
- AxisVMDomains
 - Add_vb
- DynamicLoadFunctions
 - Add_vb
 - Modify_vb
- IncrementFunctions
 - Add_vb
 - Modify_vb
- COMAxisVMLine
 - MaterialColour_vb
 - ContourColour_vb
- COMAxisVMLines
 - CrossLines_vb
- AxisVMLoadCombinations
 - Add_vb
 - SetCombination_vb
- COMAxisVMLoads
 - SetLoad_vb
 - AddSurfaceToBeam_vb
 - AddSurfaceToBeamAssoc_vb
 - SetLines_vb
- AxisVMMaterial
 - MaterialColour_vb
 - ContourColour_vb

- COMAxisVMMaterials
 - AddFromDialog_vb
 - AddSteel_Hungarian_MSZ_vb
 - AddSteel_EuroCode_vb
 - AddSteel_Romanian_STAS_vb
 - AddSteel_Dutch_NEN_vb
 - AddSteel_German_DIN1045_1_vb
 - AddSteel_Swiss_SIA26x_vb
 - AddSteel_Italian_vb
 - AddConcrete_Hungarian_MSZ_vb
 - AddConcrete_EuroCode_vb
 - AddConcrete_Romanian_STAS_vb
 - AddConcrete_Dutch_NEN_vb
 - AddConcrete_German_DIN1045_1_vb
 - AddConcrete_Swiss_SIA26x_vb
 - AddConcrete_Italian_vb
 - AddTimber_vb
 - AddAluminium_vb
 - AddBrick_vb
- COMAxisVMMember
 - CreateMeshWithCoordinates_vb
 - MaterialColour_vb
 - ContourColour_vb
- COMAxisVMMembers
 - Add_vb
 - IndexOf_vb
- COMAxisVMMovingLoadOnBeam
 - SetPath_vb
- COMAxisVMMovingLoadOnDomain
 - SetPath_vb
- COMAxisVMRCBeamDesign
 - SetLines_vb
 - SetSectionParameters_vb
 - SetSupports_vb
- COMAxisVMRCColumnChecking
 - CheckLines_vb
 - CheckMembers_vb
 - CheckByParameters_vb
- COMAxisVMResults
 - SetEnvelope_vb
- COMAxisVMRigidBodies
 - Add_vb
 - RemoveLinesFromRigidBodies_vb
- COMAxisVMSteelDesignMembers
 - Add_vb
 - Add2_vb

- COMAxisVMSurface
 - Modify_vb
 - SetContourLines_vb
 - MaterialColour_vb
 - ContourColour_vb
- COMAxisVMSurfaces
 - Add_vb
 - GetAllCoordinatesOfSurfaces_vb
- COMAxisVMTimberDesignMembers
 - Add_vb
- COMAxisVMTimeIncrementFunctions
 - Add_vb
 - Modify_vb

Changes between versions 6.1 and 6.2

Replaced fields in records and enums

- [**RSurfaceForceValues**](#)
 - Deleted fields: sfvMxD, sfvMyD
 - New fields: sfvMxDp, sfvMxDm, sfvMyDp, sfvMyDm
- [**EResultComponent**](#)
 - Deleted fields: rc_sfMxD, rc_sfMyD
 - New fields: rc_sfMxDp, rc_sfMxDm, rc_sfMyDp, rc_sfMyDm
- [**ESurfaceForce**](#)
 - Deleted fields: sfMxD, sfMyD
 - New fields: sfMxDp, sfMxDm, sfMyDp, sfMyDm

Deleted records

- RSurfaceReinforcementParams
- RRCBeamReinforcementParameters (merged with [RRCBeamDesignParameters](#))

New GUID for this record

- [RRCBeamDesignParameters](#)

Renamed records

- RSurfaceReinforcement -> [RActualReinforcement](#)
- RRCBeamSections -> [RRCBeamCrossSections](#)

Deleted fields in records

- [**RReinforcementParameters**](#):
 - RebarPos

New fields in records

- [**RReinforcementParameters_EC_DIN_SIA_ITA_only**](#)
 - AutoCalc; ExpClass_T; ExpClass_B; StructClass; dxt; dxb; dyt; dyb; AggregateSize; MainDirectionTop; MainDirectionBottom; ct; cb
- [**RReinforcementParameters_MSZ_only**](#), [**RReinforcementParameters_STAS_only**](#)
 - RebarPos
- [**RRCBeamDesignParameters**](#)
 - SLSCheck, SLSMaxCrackOpening_Bottom, SLSMaxCrackOpening_Top, Ks, RebarMaterial, StirrupMaterial, StirrupDiameter, StirrupLegs, BottomPos, TopPos, ds_top, ds_bottom, EC_ITA, DIN_SIA, STAS
- [**RColumnReinforcementParameters**](#)
 - Ks

Renamed fields in records

- [**RRCBeamSupport**](#)
 - ActualWidth -> ActualHalfWidth
 - TheoreticalWidth -> TheoreticalHalfWidth
- [**RRCBeamSection**](#)
 - b -> bw
 - t -> hf
 - b1 -> beff

- **RActualReinforcement** (former RSurfaceReinforcement)
 - $f_i \rightarrow ds$
 - $d \rightarrow \text{spacing}$

Parameters of functions modified:

- **IAxisVMRCBeamDesign**
 - Calculate
- **IAxisVMActualReinforcement**
 - AddDomainReinforcement
 - AddPolygonReinforcement
 - GetReinforcement

Deleted functions:

- **IAxisVMRCBeamDesign**
 - SetLines
 - SetLines_vb
 - GetSectionParameters
 - SetSectionParameters
 - SetSectionParameters_vb
 - GetSupports
 - SetSupports
 - SetSupports_vb
 - GetReinforcementParameters
 - SetReinforcementParameters

New functions:

- **IAxisVMRCBeamDesign**
 - AddMembers
 - AddMembers_vb
 - AddLines
 - AddLines_vb
 - GetPartialRCBeamDesignParameters
- **IAxisVMSections**
 - GetSegmentChainCount
 - GetSegmentChainCoords
 - GetSegmentChainData
 - GetSegmentChainDisplacementsByLoadCaseId
 - GetSegmentChainDisplacementsByLoadCombinationId
 - GetEnvelopeSegmentChainDisplacements
 - GetCriticalSegmentChainDisplacements
 - GetSegmentChainVelocitiesByLoadCaseId
 - GetEnvelopeSegmentChainVelocities
 - GetSegmentChainAccelerationsByLoadCaseId
 - GetEnvelopeSegmentChainAccelerations
 - GetSegmentChainSurfaceForcesByLoadCaseId
 - GetSegmentChainSurfaceForcesByLoadCombinationId
 - GetEnvelopeSegmentChainSurfaceForces
 - GetCriticalSegmentChainSurfaceForces
 - GetSegmentChainCalculatedReinforcementsByLoadCaseId
 - GetSegmentChainCalculatedReinforcementsByLoadCombinationId
 - GetEnvelopeSegmentChainCalculatedReinforcements
 - GetCriticalSegmentChainCalculatedReinforcements
 - GetSegmentChainSurfaceSupportForcesByLoadCaseId
 - GetSegmentChainSurfaceSupportForcesByLoadCombinationId
 - GetEnvelopeSegmentChainSurfaceSupportForces
 - GetCriticalSegmentChainSurfaceSupportForces
 - GetSegmentChainCrackOpeningsByLoadCaseId
 - GetSegmentChainCrackOpeningsByLoadCombinationId
 - GetEnvelopeSegmentChainCrackOpenings
 - GetCriticalSegmentChainCrackOpenings
 - GetSegmentChainShearCapacitiesByLoadCaseId

- GetSegmentChainShearCapacitiesByLoadCombinationId
 - GetEnvelopeSegmentChainShearCapacities
 - GetCriticalSegmentChainShearCapacities
 - GetSegmentChainSurfaceStressesByLoadCaseId
 - GetSegmentChainSurfaceStressesByLoadCombinationId
 - GetEnvelopeSegmentChainSurfaceStresses
 - GetCriticalSegmentChainSurfaceStresses
- **IAxisVMMModel:**
 - SaveUndo
 - Undo
 - Redo
- **IAxisVMLoadCases:**
 - DeleteAllLoadsFromLoadCase

Deleted properties:

- **IAxisVMLine, IAxisVMLines, IAxisVMMember, IAxisVMMembers**
 - StiffnessReduction

New properties:

- **IAxisVMDomain, IAxisVMDomains**
 - StiffnessReduction
- **IAxisVMLine, IAxisVMLines, IAxisVMMember, IAxisVMMembers**
 - StiffnessReduction_A
 - StiffnessReduction_I
- **IAxisVMMModel**
 - Sections

Changes between versions 6.2 and 6.3

New types in enums:

- [ENationalDesignCode](#)
 - ndcEuroCode_PL
 - ndcEuroCode_DK
 - ndcEuroCode_S
- [Elanguage](#)
 - IngPolish
 - IngBulgarian
- [EReleaseType](#)
 - rtPushover

New fields in records

- [RRCBeamDesignParameters](#)
 - ShortTime
- [RReinforcementParameters](#)
 - ConcreteTensileStrengthCheck

- [**RRCBeamDesignResult**](#)
 - VRdc
 - VRds
- RReinforcementParameters_EC_DIN_SIA_ITA_only
 - ks
- [**RReinforcementParameters_STAS_only**](#)
 - ks
- [**RSteelDesignParameters_EC_SIA_ITA**](#)
 - ks
- [**RRelease**](#)
 - FunctionId

New interface + events:

[IAxisVMPushoverHingeFunctions](#)

New properties:

- [**IAxisVMMModel**](#)
 - PushoverHingeFunctions
 - ProjectName
 - AnalysisBy
 - Comment

Changes between versions 6.3 and 6.4

New functions:

- [**IAxisVMForces**](#)
 - GetEndReleasesDeformationsByLoadCaseId
 - GetEndReleasesDeformationsByLoadCombinationId
 - GetEnvelopeEndReleasesDeformations
 - GetCriticalEndReleasesDeformations
 - GetAllEndReleasesDeformationsByLoadCaseId
 - GetAllEndReleasesDeformationsByLoadCombinationId
 - GetAllEnvelopeEndReleasesDeformations
 - GetAllCriticalEndReleasesDeformations
 - EndReleasesDeformationsByLoadCaseId
 - EndReleasesDeformationsByLoadCombinationId
 - EnvelopeEndReleasesDeformations
 - CriticalEndReleasesDeformations
 - AllEndReleasesDeformationsByLoadCaseId
 - AllEndReleasesDeformationsByLoadCombinationId
 - AllEnvelopeEndReleasesDeformations
 - AllCriticalEndReleasesDeformations

Changes between versions 6.3 and 7.0

Important change:

Compiler's setting for record (struct) align must be set to 8 bytes (quad word)

New interfaces:

[IAxisVMCriticalGroupCombinations](#), [IAxisVMEvelopes](#), [IAxisVMWindows](#), [IAxisVMWindow](#),
[IAxisVMCustomParts](#), [IAxisVMCustomPartFolder](#), [IAxisVMLogicalParts](#)

New properties

- [IAxisVMStresses](#) , [IAxisVMForces](#), [IAxisVMDisplacements](#), [IAxisVMSteelDesignResults](#),
[IAxisVMTimberDesignResults](#), [IAxisVMCalculatedReinforcement](#), [IAxisVMReinforcementCheck](#),
[IAxisVMShearCapacity](#), [IAxisVMCrackOpening](#), [IAxisVMVelocity](#), [IAxisVMAcceleration](#),
[IAxisVMSections](#)
 - EnvelopeUID
- [IAxisVMMModel](#)
 - CriticalGroupCombinations
 - Envelopes
 - Windows
 - CustomParts
 - LogicalParts

New fields in records

- [RResultBlock](#)
 - EnvelopeUID

Renamed fields in records

- [RRCBeamReinforcementParameters_DIN_SIA](#):
 - EnvironmentClass -> EnvironmentClass_DIN

Irrelevant MinMax parameters and properties have been deleted from these interfaces:

- [IAxisVMCalculatedReinforcement](#)
- [IAxisVMCrackOpening](#)
- [IAxisVMShearCapacity](#)
- [IAxisVMSteelDesignResults](#)
- [IAxisVMTimberDesignResults](#)

Renamed interface and all associated properties and enums:

- [IAxisVMCrackOpening](#) -> [IAxisVMCrackWidth](#)
- [*CrackOpening](#) -> [*CrackWidth](#)

New [EResultComponent](#) enums:

- [rc_cw_wkb](#)
- [rc_cw_wkt](#)
- [rc_cw_wk2b](#)
- [rc_cw_wk2t](#)
- [rc_cw_wRb](#)
- [rc_cw_wRt](#)

Changes between versions 7.0 and 7.1

Functions of interface [IAxisVLogicalParts](#) return partUID.

New functions:

- IAxisVMWindow
 - GetWindowDisplayPartUIDs
 - GetPixelPosition
 - SetWindowDisplayPartUIDs
 - SaveWindowToBitmap
 - SaveWindowToClipboard
 - SaveWindowToMetafile
- IAxisVMWindows
 - GetWindowDisplayPartUIDs
 - SetWindowDisplayPartUIDs
 - SaveWindowToBitmap
 - SaveWindowsToBitmap
 - SaveWindowToClipboard
 - SaveWindowsToClipboard
 - SaveWindowToMetafile
 - SaveWindowsToMetafile
- IAxisVLogicalParts
 - GetEnabledLogicalParts
 - SetEnabledLogicalParts
 - SaveDefaultEnabledLogicalParts
- IAxisVMLoads
 - GetDomainPolyLineItems

New properties:

- IAxisVLogicalParts
 - IsLogicalPart
- IAxisVMCustomParts
 - IsCustomPart
- IAxisVMAplication
 - FullExePath
- IAxisVMWindow
 - Width
 - Height

New enums:

- [EResultComponent](#)
 - rc_sd_eff
 - rc_sd_1
 - rc_sd_2
 - rc_sd_3
 - rc_sd_4
 - rc_sd_5
 - rc_sd_6
 - rc_sd_7
 - rc_sd_8
 - rc_sd_9
 - rc_sd_10
 - rc_sd_11
 - rc_sd_12

- rc_sd_13
- rc_sd_14
- rc_sd_15
- rc_sd_16
- rc_sd_17
- rc_sd_18
- rc_sd_19
- rc_sd_20
- rc_sd_21
- rc_sd_22
- rc_sd_23
- rc_sd_24
- rc_sd_25
- rc_td_eff
- rc_td_1
- rc_td_2
- rc_td_3
- rc_td_4
- rc_td_5
- rc_td_6
- rc_td_7
- rc_td_8
- rc_td_9
- rc_td_10
- rc_td_11
- rc_td_12
- rc_td_13
- rc_td_14
- rc_td_15
- rc_td_16
- rc_td_17
- rc_td_18
- rc_td_19
- rc_td_20
- rc_td_21
- rc_td_22
- rc_td_23
- [EEnvironmentClass](#)
 - ecKlasseXD2B
- EExpClass_EC
 - ecXD2b

Changes between versions 7.1 and 7.2

Properties renamed for clarity and consistency:

[IAxisVMCRossSection](#)

- Alfa -> Alpha
- lalfa -> lalpha

Some fields in records have renamed for clarity and consistency:

- [RActualReinforcement](#)
 - Alfa -> Alpha
- [REditingOptions](#)
 - ConstAngle_DeltaAlfa -> ConstAngle_DeltaAlpha
 - ConstAngle_CustomAlfa -> ConstAngle_CustomAlpha
- [EMcrMethod](#)
 - mcrmLTBeam -> mcrmAuto
- [RShowGraphicSymbols](#)
 - StoryCentGrav -> StoreyCentGrav
 - StoryShearCent -> StoreyShearCent
- [EResultComponent](#)
 - rc_scQRzMinusVRdc -> rc_scVEdMinusVRdc
- [RShearCapacityValues](#)
 - QRzMinusVRdc -> VEdMinusVRdc
- [EShearCapacity](#)
 - scQRzMinusVRdc -> scVEdMinusVRdc

New interfaces

- IAxisVMLoadPanels
- IAxisVMLoadPanel
- IAxisVMAtributes
- IAxisVMWorkplanes

New event:

- IAxisVMCalculationEvents
 - Error([\[in\]](#) long Index, [\[in\]](#) long ErrorCode)

New properties:

- IAxisVMLines
 - UID
 - IndexOfUID
- IAxisVMDomain
 - ContourPolygon
- IAxisVMNodes, IAxisVMLines, IAxisVMMembers, IAxisVMSurfaces, IAxisVMDomains
 - Attributes
- IAxisVMWindow
 - View
 - WorkPlaneIndex
 - StoryIndex
 - ActiveStoryIndex
- IAxisVMWindows
 - LoadCaseIndex
 - View

- WorkPlaneIndex
 - StoryIndex
 - ActiveStoryIndex
- IAxisVMMModel
 - LoadPanels
 - WorkPlanes
- IAxisVMLoadGroup
 - Ksi
- IAxisVMShearCapacity
 - MinMaxType

New functions:

- IAxisVMLoads
 - CreateSnowLoadOnLoadPanels
 - DeleteSnowLoadFromAllLoadPanels
 - DeleteSnowLoadFromLoadPanels
 - GetLoadPanelsOfSnowLoad
 - CreateWindLoadOnLoadPanels
 - DeleteWindLoadFromAllLoadPanels
 - DeleteWindLoadFromLoadPanels
 - GetLoadPanelsOfWindLoad
- IAxisVMLoadCases
 - CreateSnowCases
 - GetSnowLoadParams
 - SetSnowLoadParams
 - CreateWindCases
 - GetWindLoadParams
 - SetWindLoadParams
 - GetSeismicParams (*relocated but also retained in IAxisVMMmodel for compatibility*)
 - SetSeismicParams (*relocated but also retained in IAxisVMMmodel for compatibility*)
- IAxisVMAplication
 - EnableMainForm
 - DisableMainForm
- IAxisVMSettings
 - SetGravity
 - GetGravity
- IAxisVMStresses
 - GetEnvelopeLineStress2
 - GetEnvelopeSurfaceStress2
 - EnvelopeLineStress2
 - EnvelopeSurfaceStress2
- IAxisVMForces
 - GetEnvelopeLineForce2
 - GetEnvelopeSurfaceForce2
 - GetEnvelopeNodalSupportForce2
 - GetEnvelopeLineSupportForce2
 - GetEnvelopeSurfaceSupportForce2
 - GetEnvelopeSpringForce2
 - GetEnvelopeGapForce2
 - EnvelopeLineForce2
 - EnvelopeSurfaceForce2
 - EnvelopeNodalSupportForce2
 - EnvelopeLineSupportForce2
 - EnvelopeSurfaceSupportForce2

- EnvelopeSpringForce2
 - EnvelopeGapForce2
 - GetEnvelopeEdgeConnectionForces2
 - EnvelopeEdgeConnectionForces2
 - GetEnvelopeLinkElementForces2
 - EnvelopeLinkElementForces2
 - GetEnvelopeNodalDisplacement2
 - GetEnvelopeLineDisplacement2
 - EnvelopeNodalDisplacement2
 - EnvelopeLineDisplacement2
- IAxisVMCalculatedReinforcement
 - GetEnvelopeCalculatedReinforcements2
 - GetCriticalCalculatedReinforcements2
 - EnvelopeCalculatedReinforcements2
 - CriticalCalculatedReinforcements2
- IAxisVMShearCapacity
 - GetEnvelopeShearCapacities2
 - GetCriticalShearCapacities2
 - EnvelopeShearCapacities2
 - CriticalShearCapacities2
- IAxisVMCrackWidth
 - GetEnvelopeCrackWidths2
 - GetCriticalCrackWidths2
 - EnvelopeCrackWidths2
 - CriticalCrackWidths2
- IAxisVMLine, IAxisVMMember
 - DeleteLineElement
- IAxisVMMModel
 - SetAPIGlobalData
 - GetAPIGlobalData
 - GetAPIGlobalDataSize
 - DeleteAPIGlobalData
 - EnableMainForm
 - DisableMainForm

New events:

- IAxisVMMModelsEvents
 - ModelChanged

Changes between versions 7.2 and 7.3

New functions:

- [IAxisVMAtributes](#)
 - AddDefault_vb
 - FillItemByName_vb
 - FillItemByIndex_vb
 - FillAllItemsByName_vb
 - FillAllItemsByIndex_vb
 - FillItemsByName_vb
 - FillItemsByIndex_vb
 - SetAllItemsByName_vb
 - SetAllItemsByIndex_vb
- [IAxisVMLoads](#)
 - GetDomainsAndSurfaces
- [IAxisVMLines](#)
 - IndexOfFiniteElementNumber
 - GetFiniteElementCount
 - GetLinesLocX
- [IAxisVMSeismicStoreys](#)
 - GetSeismicSensitivityResults
- [IAxisVMEvelopes](#)
 - Update
- [IAxisVMMModel](#)
 - ImportDXF
 - ImportPDF
 - ImportIFC
 - GetIFCExportReinfParams
- [IAxisVMResults](#)
 - GetCapacityCurvePushOver

New interfaces

- [IAxisVMAttachments](#), [IAxisVMSteelCrossSectionOptimization](#)

New properties:

- IAxisVMNodes, IAxisVMLines, IAxisVMMembers, IAxisVMSurfaces, IAxisVMDomains
 - Attachments
- IAxisVMLines, IAxisVMMembers
 - LocalX_is_ij
 - Name
- IAxisVMEvelopes
 - Group
- IAxisVMMModel
 - CallImportProgress
 - Platform
 - [SpectrumPushOver](#)

Function parameters changed:

- [IAxisVMSurface](#), [IAxisVMDomain](#)
 - GetReinforcementParameters
 - SetReinforcementParameters

- [IAxisVMRCbeamDesign](#)
 - SetDesignParameters
 - SetDesignParameters_vb
 - GetDesignParameters

New events:

- [IAxisVMMModelsEvents](#)
 - MainProgress

Changes between versions 7.3 and 7.4

New events:

- [IAxisVMMModelsEvents](#)
 - BeforeMessageDisplay
 - AfterMessageDisplay
 - DisplayedErrors

New properties:

- IAxisVMWindow
 - ShowOnlySelected

Changes between versions 7.4 and 7.5

Renamed properties:

- IAxisVMMModel
 - CallImportProgress -> CallProgress

New functions:

- [IAxisVMLoadCombinations](#)
 - GetValidCombinationTypes
- [IAxisVMSteelDesignResults](#), [IAxisVMTimberDesignResults](#)
 - GetEfficiencyAndCombination
 - GetEfficiencyAndCombinationByLoadCaseId
 - GetEfficiencyAndCombinationByLoadCombinationId
 - GetEnvelopeEfficiencyAndCombination
 - GetCriticalEfficiencyAndCombination

Changes between versions 7.5 and 8.0

Renamed functions:

- GetDomainsAndSurfaces -> GetDomains_Surfaces_LoadPanels

New EGeneralErrors: errCriticalCombinationNotAllowed, errInvalidName, errCombinationTypeNotAllowed

New functions:

- [IAxisVMStresses](#)
 - GetMemberStressesByLoadCaseId
 - GetMemberStressesByLoadCombinationId
 - GetEnvelopeMemberStresses
 - GetCriticalMemberStresses
- [IAxisVMForces](#)
 - GetMemberForcesByLoadCaseId
 - GetMemberForcesByLoadCombinationId
 - GetEnvelopeMemberForces
 - GetCriticalMemberForces
- [IAxisVMDisplacements](#)
 - GetMemberDisplacementsByLoadCaseId
 - GetMemberDisplacementsByLoadCombinationId
 - GetEnvelopeMemberDisplacements
 - GetCriticalMemberDisplacements
- [IAxisVMSteelDesignResults](#)
 - GetEnvelopeSteelDesignResults2
 - GetSteelDesignResultsByLoadCaseId_Abs
 - GetSteelDesignResultsByLoadCombinationId_Abs
 - GetEnvelopeSteelDesignResults_Abs
 - GetCriticalSteelDesignResults_Abs
- [IAxisVMTimberDesignResults](#)
 - GetEnvelopeTimberDesignResults2
 - GetTimberDesignResultsByLoadCaseId_Abs
 - GetTimberDesignResultsByLoadCombinationId_Abs
 - GetEnvelopeTimberDesignResults_Abs
 - GetCriticalTimberDesignResults_Abs
- [IAxisVMResults](#)
 - GetAllModalMassfactors
- [IAxisVMSurface](#)
 - GetVariableThickness
- [IAxisVMDomains](#)
 - GetVariableThickness
 - SetVariableThickness
 - GetExcentricity
 - SetExcentricity
 - CreateNewExcentricityGroup
 - GetRibbedDomainParameters
 - SetRibbedDomainParameters
 - GetXLAMPParameters
 - SetXLAMPParameters
- [IAxisVMLoads](#)
 - AddLoadPanelConcentrated
 - AddLoadPanelLinear
- [IAxisVMModel](#)
 - SaveModelBeforeClose

- [IAxisVMMember](#)
 - GetXofMemberSectionID
 - GetLineAndLineSectionID
- [IAxisVMCrossSections](#)
 - AddRectangularRounded
 - ReplaceWithRectangularRounded
 - AddRectangularHollow
 - ReplaceWithRectangularHollow
 - AddIHaunched
 - ReplaceWithIHaunched
 - AddTWallHaunched
 - ReplaceWithTWallHaunched
 - AddTTopHaunched
 - ReplaceWithTTopHaunched
 - AddCircleHollow
 - ReplaceWithCircleHollow
 - AddTrapezoid
 - ReplaceWithTrapezoid
 - GetDimensionsOfC
 - GetDimensionsOfIHaunched
 - GetDimensionsOfTWallHaunched
 - GetDimensionsOfTTopHaunched
- AxisVMWindow, AxisVMWindows
 - GetVisibleLayerIDs
 - SetVisibleLayerIDs
 - GetDetectedLayerIDs
 - SetDetectedLayerIDs
 - GetLockedLayerIDs
 - SetLockedLayerIDs

New properties:

- [IAxisVMDomain](#)
 - VariableThickness
- [IAxisVMDomains](#)
 - IsRibbed
 - IsXLAM
 - IsExcentic
 - HasVariableThickness
- [IAxisVMMember](#)
 - MemberSectionsCount
- IAxisVMCustomParts
 - Attachments
- IAxisVMLoadPanel
 - GetLines
 - SetLines
 - GetNodes
 - SetNodes
 - Auto

New interfaces:

- IAxisVMMathTexts
- IAxisVMLayers

- [IAxisVMXLAMpanels](#)
- [IAxisVMDrawingsLibrary](#)
- [IAxisVMReports](#)

Changes between versions 8.0 and 8.1

New [EGeneralErrors](#): errInvalidEnvelopeUID

New functions:

- [IAxisVMStresses](#)
 - [GetXLAMSurfaceStressByLoadCaseId](#)
 - [GetXLAMSurfaceStressByLoadCombinationId](#)
 - [GetEnvelopeXLAMSurfaceStress](#)
 - [GetCriticalXLAMSurfaceStress](#)
 - [GetXLAMSurfaceStressValuesForResultBlocks](#)
 - [GetXLAMSurfaceStressesByLoadCaseId](#)
 - [GetXLAMSurfaceStressesByLoadCombinationId](#)
 - [GetEnvelopeXLAMSurfaceStresses](#)
 - [GetCriticalXLAMSurfaceStresses](#)
 - [GetXLAMSurfaceStressesForResultBlocks](#)
- [IAxisVMCatalog](#)
 - [GetXLAMmanufacturers](#)
 - [GetXLAMnamesByManufacturers](#)

New properties:

- [IAxisVMStresses](#)
 - [XLAMSurfaceStressComponent](#)
- [IAxisVMDisplacements](#)
 - [DisplacementSystem](#)

Changes between versions 8.1 and 8.2

New Interfaces:

- [IAxisVMStructuralGrids](#)
- [IAxisVMStructuralGrid](#)
- [IAxisVMDimensions](#)

New Errors:

- [EGeneralError](#)
 - [errInvalidPosition](#)
 - [errIndexDuplication](#)
- [IAxisVMDomains](#)
 - [EDomainsError](#)
 - [deEnvironmentClassNotValidForUsedDesignCode](#)
- [IAxisVMSurfaces](#)
 - [ESurfacesError](#)
 - [seEnvironmentClassNotValidForUsedDesignCode](#)
- [IAxisVMStresses](#)
 - [EStressesError](#)
 - [steNotXLAMpanel](#)
 - [steXLAMmoduleNotAvailable](#)
 - [steStressPointIDOutOfBounds](#)
- [IAxisVMSteelDesignMembers](#)
 - [ESteelDesignMemberError](#)
 - [sdmeDesignParametersNotValidForUsedDesignCode](#)
- [IAxisVMLogicalParts](#)
 - [ELogicalPartsError](#)

- IpeStructuralGridLineUIDOutOfBounds
- [IAxisVMSignals](#)
 - EStructuralGidsError
- [IAxisVMDimensions](#)
 - EDimensionsError

Deleted error codes:

- [IAxisVMCalculation](#)
 - ECalculationError
 - ceCalculationWasCanceled
 - ceCalculationEndsWithError

New enums:

- [IAxisVMSteelDesignMembers](#)
 - ESteelBucklingLengthMode
 - ESteelCantileverFixedEnd
 - ESteelLateralSupports
- [IAxisVMMaterial](#)
 - ECompanyLogoPosition
 - ECompanyLogoSizeOption
 - EGeneralAlignment
- [IAxisVMCrossSections](#)
 - ECrossSectionImageExportOptions
- [IAxisVMSignals](#)
 - EGridLineSpacingDirection
- [IAxisVMSignals](#)
 - EShowStructuralGridLineTitle
 - EStructuralGridPlane
 - EStructuralGridVisibility
 - EStructuralGridLabelTextType
- [IAxisVMSignals](#)
 - EXLAMSurfaceEfficiency
- [IAxisVMDimensions](#)
 - EDimensionType
 - EDimensionLabelOrientation
 - EDimensionStyle
- [IAxisVMSignals](#)
 - ESectionSegmentChainIntegratedResultant
- [IAxisVMCalculationEvents](#)
 - ECalculatedFinishedType

Modified enums:

- [IAxisVMDomain](#)
 - EEnvironmentClass
- [IAxisVMSteelDesignMembers](#)
 - EMcrMethod
 - mcrmUser
- [IAxisVMSignals](#)
 - EXLAMSurfaceStress
 - XSS_Srx_max
 - XSS_Sry_max

New records:

- [IAxisVMMaterial](#)
 - RCompanyLogoParameters
- [IAxisVMSignals](#)
 - RStructuralGridLineParams
 - RStructuralGridParams
 - RStructuralGridGenerationParams
- [IAxisVMSignals](#)

- RXLAMSurfaceEfficiencyValues
- RXLAMSurfaceEfficiencies
- [IAxisVMSections](#)
 - RSectionSegmentIntegratedResultant
 - RSectionSegmentChainIntegratedParameters
- [IAxisVMSteelDesignMembers](#)
 - RSteelLateralSupport
 - RSteelLTBSupport
- [IAxisVMDimensions](#)
 - RDimensionLineParameters
 - RTextBoxParameters

Modified records:

- [IAxisVMLoadCases](#)
 - RSeismicParams
 - RSpectrumData_ITA
 - RSpectrumData_SIA
 - RSpectrumData_DIN
- [IAxisVMSteelDesignMembers](#)
 - RSteelDesignParameters_EC_SIA_ITA
- [IAxisVMColumnRebars](#)
 - RColumnReinforcementParameters
- [IAxisVMRCColumnChecking](#)
 - RColumnCheckResult
- [IAxisVMWindows](#)
 - RShowGraphicSymbols
- [IAxisVMLogicalParts](#)
 - REnabledLogicalParts
- [IAxisVMStresses](#)
 - RXLAMSurfaceStressValues

New functions:

- [IAxisVMSettings](#)
 - EnvironmentClassIsValid
- [IAxisVMStresses](#)
 - GetXLAMSurfaceEfficiencyByLoadCaseId
 - GetXLAMSurfaceEfficiencyByLoadCombinationId
 - GetEnvelopeXLAMSurfaceEfficiency
 - GetCriticalXLAMSurfaceEfficiency
 - GetXLAMSurfaceEfficienciesByLoadCaseId
 - GetXLAMSurfaceEfficienciesByLoadCombinationId
 - GetEnvelopeXLAMSurfaceEfficiencies
 - GetCriticalXLAMSurfaceEfficiencies
 - SaveMemberStressesToMetaFileByLoadCaseID
 - SaveMemberStressesToMetaFileByLoadCombinationID
 - SaveEnvelopeMemberStressesToMetaFile
 - SaveCriticalMemberStressesToMetaFile
- [IAxisVMDisplacements](#)
 - SaveMemberDisplacementsToMetaFileByLoadCaseID
 - SaveMemberDisplacementsToMetaFileByLoadCombinationID
 - SaveEnvelopeMemberDisplacementsToMetaFile
 - SaveCriticalMemberDisplacementsToMetaFile

- [IAxisVMResults](#)
 - GetAllActivatedMasses
 - GetUsedMassOfNodes
 - GetResultsValid
- [IAxisVMCrossSections](#)
 - SaveToMetaFile
 - SaveToBitmapFile
- [IAxisVMSteelDesignMembers](#)
 - Add3
 - GetDesignParameters3
 - SetDesignParameters3
 - GetLateralSupports
 - SetLateralSupports
- [IAxisVMSections](#)
 - GetSegmentChainIntegratedResultantByLoadCaseId
 - GetSegmentChainIntegratedResultantByLoadCombinationId
 - GetEnvelopeSegmentChainIntegratedResultant
 - GetCriticalSegmentChainIntegratedResultant
 - GetSegmentIntegratedResultantChainCount
 - GetSegmentIntegratedResultantChainCoords
 - GetSegmentIntegratedResultantChainData
 - GetSegmentIntegratedResultantParams
- [IAxisVMWindow](#)
 - GetVisibleStructuralGridIDs
 - SetVisibleStructuralGridIDs
- [IAxisVMCustomParts](#)
 - GetPartItemsByUID
- [IAxisVMLogicalParts](#)
 - GetBy_StructuralGridLineUID
 - GetPartItemsByUID
- [IAxisVMMaterial](#)
 - GetCompanyLogoParameters
 - SetCompanyLogoParameters
 - SelectAll

Modified functions:

- [IAxisVMSignatures](#)
 - GetEnvelopeXLAMSurfaceStress
 - GetEnvelopeXLAMSurfaceStresses
- [IAxisVMAplication](#)
 - Platform → AxisVMPlatform

New properties:

- [IAxisVMLineSupports](#)
 - LineID
- [IAxisVMCustomParts](#)
 - IndexOfUID
 - FullName
- [IAxisVMLogicalParts](#)
 - Name
 - FullName
- [IAxisVMMaterial](#)
 - StructuralGrids
 - Dimensions

- [IAxisVMApplication](#)
 - AskSaveOnLastReleased
- [IAxisVMStructuralGrid](#)
 - SpacingDirection

New events:

- [IAxisVMCalculationEvents](#)
 - Finished
- [IAxisVMWorkplanesEvents](#)
 - Cleared
- [IAxisVMWindows](#)
 - New
- [IAxisVMStructuralGridsEvents](#)
- [IAxisVMDimensionsEvents](#)

Modified events:

- [IAxisModelsEvents](#)
 - BeforeMessageDisplay
 - AfterMessageDisplay

Important!

The meaning of values of steel design results (both design and limit values) returned by functions of [IAxisVMSteelDesignResults](#) interface in records [RSteelDesignResult](#) have been changed. See [here](#)

Changes between versions 8.2 and 8.3

New Interface:

- [IAxisVMVirtualBeams](#)

New Errors:

- [EGeneralError](#)
 - errJSONpropertyMissing
- [IAxisVMForces](#)
 - EForcesError
 - feZeroValidLineNumber
 - feVirtualBeamIndexOutOfBounds
 - feVirtualBeamChainIndexOutOfBounds
 - feVirtualBeamSectionIndexOutOfBounds
 - feWindowIdNotValid
- [IAxisVMLine](#)
 - ELineError
 - lneLinesNotContinuous
 - lneInvalidFunctionIDofRelease
 - lneReleaseInitAndLimitMustBe0
 - lneFunctionIDMustBe0
- [IAxisVMDomains](#)
 - EDomainsError
 - deDomainIsNotMeshed
- [IAxisVMMaterial](#)
 - EModelError
 - meRevitModuleNotAvailable
 - meRevitImportTessDegreeOutOfRange
- [IAxisVMMaterials](#)
 - EMaterialsError
 - mbeInvalidFunctionIDofRelease
 - mbeReleaseInitAndLimitMustBe0

- mbeFunctionIdMustBe0
- mbeMembersNotContinuous
- [IAxisVMVirtualBeams](#)
 - EVirtualBeamError
 - vbeDomainIndexOutOfBounds
 - vbeDomainIndexIsInvalid
 - vbeDomainListIsEmpty
 - vbeChainIndexIsInvalid

New enums:

- [IAxisVMForces](#)
 - EVirtualBeamForce
 - EConnectedToNodeType
- [IAxisVMVirtualBeams](#)
 - RVirtualBeamParams
 - RVirtualStripParams
- [IAxisVMMModel](#)
 - [EGeneralAlignmentHorizontal](#)
 - [EGeneralAlignmentVertical](#)
- [IAxisVMResults](#)
 - [EXYchartFillType](#)
 - [EXYchartLabelingStyle](#)

Modified enums:

- [IAxisVMMModel](#)
 - EGeneralAlignment -> [EGeneralAlignmentHorizontal](#)

New records:

- [IAxisVMForces](#)
 - RVirtualBeamForceValues
- [IAxisVMVirtualBeams](#)
 - EVBDomainsDuplicateMode
 - EVirtualBeamType

Modified records:

- [IAxisVMMModel](#)
 - RCompanyLogoParameters

New functions:

- [IAxisVMForces](#)
 - LineForcesByLoadCaseIdConnectedToNode
 - LineForcesByLoadCombinationIdConnectedToNode
 - EnvelopeLineForcesConnectedToNode
 - CriticalLineForcesConnectedToNode
 - VirtualBeamOrStripForcesByLoadCaseId
 - VirtualBeamOrStripForcesByLoadCombinationId
 - EnvelopeVirtualBeamOrStripForces
 - EnvelopeVirtualBeamOrStripForces2
 - CriticalVirtualBeamOrStripForce
 - CriticalVirtualBeamOrStripForce2
 - CriticalVirtualBeamOrStripForces
- [IAxisVMMModel](#)
 - StartModalSelection
 - ImportRAE
- [IAxisVMReports](#)
 - AddRootFolder
 - DeleteImage
 - ImagesInFolder

- [**IAxisVMResults**](#)
 - GetSectionCoordinates
 - SaveXYchartToMetaFile
 - GetXYchartOptionsJSON
- [**IAxisVMDomains**](#)
 - DeleteMeshes
 - DeleteAllMeshes
- [**IAxisVMDomain**](#)
 - DeleteMesh
- [**IAxisVMLines**](#)
 - GetContinuousLineIDs
- [**IAxisVMVirtualBeams**](#)
 - GetDomains
 - Delete
 - IndexOf
 - AddNewVirtualBeam
 - AddNewDomainToVirtualBeam
 - ModifyDomains
 - GetVirtualBeamParams
 - GetVirtualStripParams
 - SetVirtualBeamParams
 - SetVirtualStripParams
 - GetCenterCoordinates
 - AddNewVirtualStrip
- [**IAxisVMVirtualBeams**](#)
 - GetContinuousMemberDs

New properties:

- [**IAxisVMMModel**](#)
 - VirtualBeams
- [**IAxisVMForces**](#)
 - VirtualBeamForceComponent
- [**IAxisVMReports**](#)
 - ImageCount
 - ImageCaption
 - ImagePath
- [**IAxisVMVirtualBeams**](#)
 - Count
 - Name
 - Length
 - VirtualBeamType
 - ChainCount
 - SectionCount
 - UID
 - IndexOfUID

New events:

- [**IAxisVMVirtualBeamsEvents**](#)

Changes between versions 8.3 and 8.4

New Errors:

- [IAxisVMLoadPanels](#)
 - ELoadPanelsError
 - loeNodeIndexListEmpty
 - loeDomainIndexListEmpty
- [IAxisVMLoads](#)
 - ELoadsError
 - loePointIsOutOfLoadPanel
 - loeZeroLoadValueOnLoadPanel

New records:

- [IAxisVMLoads](#)
 - RLoadPanelPolyArea
 - RLoadPanelPolyLine
- [IAxisVMWindows](#)
 - RWorldRectangle

New functions:

- [IAxisVMLoadPanel](#)
 - GetDomains
 - SetDomains
- [IAxisVMLoads](#)
 - AddLoadPanelPolyArea
 - AddLoadPanelPolyLine
- [IAxisVMWindows](#)
 - GetWorldRectangle
 - SetWorldRectangle
- [IAxisVMWindow](#)
 - GetWorldRectangle
 - SetWorldRectangle

Changes between versions 8.4 and 9.0

New Errors:

- [IAxisVMLoads](#)
 - [ELoadsError](#)
 - loeDerivedSurfaceLoadsNotConverted
- [IAxisVMVirtualBeams](#)
 - [EVirtualBeamError](#)
 - vbeVirtualBeamNoSection
 - vbINVALIDSections
- [IAxisVMCrossSections](#)
 - [ECrossSectionError](#)
 - cseTooHigh_e

New enums:

- [IAxisVMForces](#)
 - [ESeismicComponentSumType](#)
- [IAxisVMVirtualBeams](#)
 - [EVBDefinitionType](#)
 - [EVSDefinitionType](#)
- [IAxisVMWindows](#)
 - [EDisplayMode](#)
 - dmDiagramFilled
 - dmSectionFilled
- [IAxisVMLoadCases](#)
 - [ELoadCaseType](#)

- Ictfire
- [IAxisVMILoadGroups](#)
 - [ELoadGroupType](#)
 - Igfire

New functions:

- [IAxisVMForces](#)
 - LineForceByLoadCombinationIdEQ
 - EnvelopeLineForceEQ
 - CrticallineForceEQ
 - SaveVirtualBeamOrStripForcesToMetaFileByLoadCaseID
 - SaveVirtualBeamOrStripForcesToMetaFileByLoadCombinationID
 - SaveEnvelopeVirtualBeamOrStripForcesToMetaFile
 - SaveCriticalVirtualBeamOrStripForcesToMetaFile
- [IAxisVMIloads](#)
 - ConvertDerivedSurfaceLoad
 - ConvertSelectedDerivedSurfaceLoad
- [IAxisVMWindow](#)
 - ReDraw
- [IAxisVMWindows](#)
 - ReDraw
- [IAxisVMAApplication](#)
 - Quit
- [IAxisVMVirtualBeams](#)
 - GetReductionPointCoordinates
 - GetExtendedDomainList

New events:

- [IAxisVMLinesEvents](#)
 - AfterDeleted
- [IAxisVMMembersEvents](#)
 - AfterDeleted
- [IAxisVMMODELSEvents](#)
 - FileChanged

New properties:

- [IAxisVMForces](#)
 - SeismicComponentSumType

Modified functions:

- [IAxisVMForces](#)
 - LineForcesByLoadCombinationIdConnectedToNode
 - EnvelopeLineForcesConnectedToNode
 - CrticallineForcesConnectedToNode
- [IAxisVMRCColumnChecking](#)
 - CheckByParameters
- [IAxisVMVirtualBeams](#)
 - AddNewVirtualBeam

Modified records:

- [IAxisVMColumnRebars](#)
 - [RColumnCheckingParameters](#)
- [IAxisVMRCColumnChecking](#)
 - [RColumnCheckResult](#)
 - [RColumnForces](#)
- [IAxisVMVirtualBeams](#)
 - [RVirtualBeamParams](#)
 - [RVirtualStripParams](#)

Deleted functions:

- [IAxisVMRCColumnChecking](#)
 - CheckLines
 - CheckLines_vb

Changes between versions 9.0 and 9.1

New Errors:

- [EDomainsError](#)
 - deAlphaVRdmaxIsInvalid
 - deThetaVRdmaxIsInvalid
 - deShrinkageEpsMustBePositive
 - deRCNonlinearSurfTypeIsInvalid
- [ESurfacesError](#)
 - seAlphaVRdmaxIsInvalid
 - seThetaVRdmaxIsInvalid
 - seShrinkageEpsMustBePositive
 - seRCNonlinearSurfTypeIsInvalid
- [EDisplacementsError](#)
 - deVirtualBeamIndexOutOfBounds
 - deVirtualBeamChainIndexOutOfBounds
 - deVirtualBeamSectionIndexOutOfBounds
- [ERCBeamDesignError](#)
 - rcbdeInvalidShrinkageValue
- [ERCColumnCheckingError](#)
 - rccceInvalidCheckingParameters
 - rccceShrinkageEpsMustBePositive
- [ELineError](#)
 - lneStartEndCrossSectionTypeIncompatible
 - lneInvalidRCCheckingParameters
 - lneRCShrinkageEpsMustBePositive
- [EMembersError](#)
 - mbeStartEndCrossSectionTypeIncompatible
 - mbeInvalidRCCheckingParameters
 - mbeRCShrinkageEpsMustBePositive

New enums:

- [EShearCapacity](#)
 - scVRdmax
 - scVEdDivVRdmax
 - scaVEd
- [EResultComponent](#)
 - rc_scVRdmax
 - rc_scVEdDivVRdmax
 - rc_scaVEd
- [ERCNonlinearSurfType](#)

New functions:

- [IAxisVMDisplacements](#)
 - VirtualBeamOrStripDisplacementsByLoadCaseId
 - VirtualBeamOrStripDisplacementsByLoadCombinationId
 - EnvelopeVirtualBeamOrStripDisplacements
 - EnvelopeVirtualBeamOrStripDisplacements2
 - CriticalVirtualBeamOrStripDisplacements
 - CriticalVirtualBeamOrStripDisplacement
 - CriticalVirtualBeamOrStripDisplacement2
 - SaveVirtualBeamOrStripDisplacementsToMetaFileByLoadCaseID
 - SaveVirtualBeamOrStripDisplacementsToMetaFileByLoadCombinationID
 - SaveEnvelopeVirtualBeamOrStripDisplacementsToMetaFile
 - SaveCriticalVirtualBeamOrStripDisplacementsToMetaFile
- [IAxisVMSettings](#)
 - GetUnitParams_*
- IAxisVMStresses, IAxisVMForces, IAxisVMDisplacements, IAxisVMShearCapacity, IAxisVMCrackWidth, IAxisVMSections, IAxisVMResults
 - SetUserCreep

- IAxisVMResults
 - UpdateResults
 - GetTotalLoadsByLoadCaseID
- IAxisVMLineSupport, IAxisVMNodalSupport
 - GetStiffnessCalcParams
 - SetStiffnessCalcParams
- IAxisVMLineSupport
 - GetNodeIDs
- IAxisVMLineSupports, IAxisVMNodalSupports
 - GetTrMatrix
- IAxisVMWindow
 - PanToCoord

New interface:

- [IAxisVMTTask](#)

New properties:

- [IAxisVMSettings](#)
 - StiffnessReductionColumns_A
 - StiffnessReductionColumns_I
 - StiffnessReductionBeams_A
 - StiffnessReductionBeams_I
 - StiffnessReductionOtherMembers_A
 - StiffnessReductionOtherMembers_I
 - StiffnessReductionWalls
 - StiffnessReductionSlabs
 - StiffnessReductionOtherDomains
- IAxisVMSignatures, IAxisVMForces, IAxisVMDisplacements, IAxisVMShearCapacity, IAxisVMCrackWidth, IAxisVMSections, IAxisVMResults
 - UserCreep
- IAxisVMLineSupports, IAxisVMNodalSupports
 - HaveStiffnessCalcParams

Modified errors:

Modified functions:

Modified records:

- [RNonlinearAnalysis](#)
- [RShearCapacityValues](#)
- [RReinforcementParameters_DIN](#)
- [RReinforcementParameters_EC](#)
- [RReinforcementParameters_ITA](#)
- [RReinforcementParameters_SIA](#)
- [RRCBeamDesignParameters](#)
- [RColumnCheckingParameters](#)
- [RColumnReinforcementParameters](#)
-

Deleted functions:

Changes between versions 9.1 and 9.2

Changes between versions 9.2 and 9.3

New Errors:

- [EDomainsError](#)
 - deAlphaAngleIsInvalid
 - deBetaAngleIsInvalid
- [ESurfacesError](#)
 - seAlphaAngleIsInvalid
 - seBetaAngleIsInvalid
- [ERCCColumnCheckingError](#)
 - rccceVTCheckIsNotSupported
 - rccceStirrupParametersAreInvalid
 - rccceShearCrackAngleIsInvalid
- [ELineError](#)
 - lneStirrupParametersAreInvalid
 - lneShearCrackAngleIsInvalid
- [EMembersError](#)
 - mbeStirrupParametersAreInvalid
 - mbeShearCrackAngleIsInvalid

New functions:

- [IAxisVMRCColumnChecking](#)
 - CheckVTByParameters
 - CheckVTByParameters_vb
 - CheckVTMembers
 - CheckVTMembers_vb
- [IAxisVMForces](#)
 - GetMembersSupportForcesByLoadCaseId
 - GetMembersSupportForcesByLoadCombinationId
 - GetEnvelopeMembersSupportForces
 - GetCriticalMembersSupportForces

- IAxisVMSurface, IAxisVMDomain
 - GetSurfaceStiffnessFactors
 - SetSurfaceStiffnessFactors
- IAxisVMDomain , IAxisVMDomains
 - GetCustomStiffnessMatrix
 - SetCustomStiffnessMatrix
- IAxisVMLineSupport, IAxisVMNodalSupport
 - GetFootingDimensions
 - GetFootingParams
- IAxisVMMemberss
 - AssembleSelectedMembers
- IAxisVMNodes,
 - RemoveSelectedIntermedNodes

New properties:

- IAxisVMSteelDesignMembers
 - CrossSctionID
- IAxisVMLineSupport, IAxisVMNodalSupport
 - HasFooting
 - FootingType

New records:

- [RColumnVTCheckResult](#)
- [RColumnStirrupDiameters](#)
- [RColumnStirrupSpacing](#)
- [RColumnStirrupZones](#)

Modified records:

- [RColumnCheckingParameters](#)
- [RColumnReinforcementParameters](#)
- [RReinforcementParameters_EC](#)
- [RReinforcementParameters_ITA](#)
- [RReinforcementParameters_SIA](#)
- [RColumnForces](#)

Renamed fields in records:

- [RExtendedDisplayParameters:](#)
 - BasicDisplayParameters -> BasicDispParams

New enum:

- [EReinforcementType](#)

New interfaces:

- IAxisVMNodesSupports
- IAxisVMMembersSupports
- IAxisVMDomainsSupports

Changes between versions 9.3 and 15.0

New Errors:

- [EGeneralError](#)
 - errOutOfMemory
- [ECrossSectionError](#)
 - cseTooLargeInnerCrossSection
 - cseInvalidMaterials
- [EDomainsError](#)
 - deLimitingCrackWidthIsInvalid
 - dePolyLineIsNotContinuous
 - deLineDoesNotReachDomainEdge
 - deNoSelectedLine
 - deNoSelectedLineAndDomain
 - deMaterialIndex
 - deReferenceIndexOutOfBounds
 - deDomainInvalidType
 - deElasticFoundationNegative
- [ESurfacesError](#)
 - seLimitingCrackWidthIsInvalid
 - seMaterialIndex
 - seSurfaceReferenceIndexOutOfBounds
 - seInvalidType
 - seElasticFoundationNegative
- [ERCBeamDesignError](#)
 - rcbdeInvalidDesignParameters
 - rcbdeInvalidPlasticHingeParams
- [ERCColumnCheckingError](#)
 - rccceInvalidDesignParameters
 - rccceVTCapacityDesignIsNotSupported
- [EMembersSupportsError](#)
 - mseInvalidRefType

New enum:

- ECrossSectionShapeEx
- ESeismicDuctilityClass
- ERCBeam_EC_SIA_SeismicZone
- ERCBeam_ECRO_STAS_SeismicZone
- ECompositeInnerCSalign

Deleted enum:

- ERCBeam_STAS_SeismicZone

Modified enum:

- ENationalDesignCode
- ECrossSectionShape
- ELineSupportType
- ECalculationUserInteraction
- ENodalSupportForce
- EResultComponent
- ECrossSectionBasicType
- EXLAMSSurfaceEfficiency
- ESeismicComponentSumType

New records:

- RLineData
- RLineAttr

- RSurface
- RRCBeamPlasticHingeParams
- RRCBeamPlasticHinges
- RRCColumnCapacityDesignParams

Modified records:

- RReinforcementParameters_EC
- RReinforcementParameters_ITA
- RReinforcementParameters_SIA
- RRCBeamDesignParameters_EC_RO
- RRCBeamDesignParameters_EC
- RRCBeamDesignParameters_ITA
- RRCBeamDesignParameters_SIA
- RRCBeamDesignParameters_STAS
- RColumnReinforcementParameters
- RXLAMSurfaceStressValues
- RXLAMSurfaceEfficiencyValues

New functions:

- [IAxisVMSurfaces](#)
 - BulkAdd
 - BulkGetSurfaces
 - BulkSetSurfaces
- [IAxisVMDomains](#)
 - CutSelected
 - BulkAdd
 - BulkGetDomains
 - BulkSetDomains
- [IAxisVMLines](#)
 - BulkAdd
 - BulkGetAttr
 - BulkGetLineData
 - BulkSetAttr
 - BulkSetLineData
 - BulkGetMemberIds
 - GetSurfaces
- [IAxisVMMembers](#)
 - BulkGetMembers
 - BulkSetMembers
- [IAxisVMNodes](#)
 - BulkAdd
 - BulkGetCoord
 - BulkGetDOF
 - BulkSetDOF
 - BulkSetNodeCoord
 - GetNodeLines
- [IAxisVMCrossSections](#)
 - AddCompositePipe
 - AddCompositeBox
 - AddCompositeRound
 - AddCompositeRectangle
 - AddFromDialogEx
 - EditEx
 - ReplaceFromCatalogEx
- [IAxisVMRCColumnChecking](#)
 - CheckVTByParameters_EQCapacityDesign
 - CheckVTByParameters_EQCapacityDesign_vb
- [IAxisVMMembersSupports](#)
 - AddDomainEdgeRefSupport
- [IAxisVMCatalog](#)

- GetCrossSectionNamesEx
- GetCrossSectionTableNamesEx
- GetCrossSectionEx
- [IAxisVMAApplication](#)
 - HandleMessages

New properties:

- IAxisVMCrossSection
 - CrossSectionShapeEx
- IAxisVMCrossSectionOptimization
 - GroupCrossSectionShapeEx
- IAxisVMMembersSupports
 - ReferenceID

Changes between versions 15.0 and 15.1

New Errors:

- [ELineError](#)
 - IneInvalidSteelMaterialId
- [EMembersError](#)
 - mbeInvalidSteelMaterialId
- [ERCColumnCheckingError](#)
 - rccceInvalidSteelMaterialId

Modified enum:

- ESeismicComponentSumType

Modified records:

- RReinforcementParameters_EC
- RReinforcementParameters_ITA
- RReinforcementParameters_SIA
- RRCBeamPlasticHingeParams
- RColumnReinforcementParameters

New functions:

- IAxisVMSpectrum
 - Disable

Changes between versions 15.1 and 15.3

New Errors:

- [ESupportError](#)
 - seIncompatibleReferences
- [ESpringParError](#)

New enum:

- EPadFootingType2
- EPadFootingStepMeasureSource
- ESpringParType
- ESpringParNNTYPE
- ESpringParDOFTYPE
- ESpringParDampingType
- ESpringParNonLinearity
- ESpringParNLDefType
- ESpringParHardeningRule
- ESpringParMatrixType

- ESpringParIsolatorType
- ESeismicLimitState

Modified enum:

- EResultComponent

New records:

- RRectangularFootingSpec
- RRectangularFootingCalced
- RCircularFootingSpec
- RCircularFootingCalced
- RPadFootingParams_V153
- RLinearFootingSpec
- RLinearFootingCalced
- RLinearFootingParams
- RBasicDisplayParameters_V153
- RExtendedDisplayParameters_V153
- RSpringParam
- RSpringParamIndexes
- RNodalSupportSpringParams
- RSeismicParams_V153
- RSpectrumData_ECHU
- RSpectrumData_ECNL
- RSpectrumData_V153

Obsolete records:

- RPadFootingDimensions
- RPadFootingParams
- RBasicDisplayParameters
- RExtendedDisplayParameters
- RSeismicParams
- RSpectrumData

New interface:

- [IAxisVMSpringParam](#)
- [IAxisVMSpringParams](#)

New functions:

- IAxisVMLineSupport
 - GetFootingParams_V153
- IAxisVMNodalSupport
 - GetFootingParams_V153
- IAxisVMNodalSupports
 - AddNodalBeamRelative_V153
 - AddNodalEdgeRelative_V153
 - AddNodalGlobal_V153
 - AddNodalLocal_V153
 - AddNodalReference_V153
 - AddIsolator
- IAxisVMNodesSupports
 - AddIsolator
 - AddNodalGlobal_V153
 - AddNodalLocal_V153
 - AddNodalMemberRelative_V153
 - AddNodalDomainRelative_V153
 - AddNodalReferenceRelative_V153
 - GetFootingParams_V153
- IAxisVMMembersSupports
 - GetFootingParams_V153
- IAxisVMWindows
 - GetBucklingDisplayParameters_V153
 - GetDynamicDisplayParameters_V153

- GetRCDesignDisplayParameters_V153
 - GetStaticDisplayParameters_V153
 - GetSteelDesignDisplayParameters_V153
 - GetTimberDesignDisplayParameters_V153
 - GetVibrationDisplayParameters_V153
 - SetBucklingDisplayParameter_V153
 - SetDynamicDisplayParameters_V153
 - SetRCDesignDisplayParameters_V153
 - SetStaticDisplayParameters_V153
 - SetSteelDesignDisplayParameters_V153
 - SetTimberDesignDisplayParameters_V153
 - SetVibrationDisplayParameters_V153
- IAxisVMWindow
 - GetBucklingDisplayParameters_V153
 - GetDynamicDisplayParameters_V153
 - GetRCDesignDisplayParameters_V153
 - GetStaticDisplayParameters_V153
 - GetSteelDesignDisplayParameters_V153
 - GetTimberDesignDisplayParameters_V153
 - GetVibrationDisplayParameters_V153
 - SetBucklingDisplayParameter_V153
 - SetDynamicDisplayParameters_V153
 - SetRCDesignDisplayParameters_V153
 - SetStaticDisplayParameters_V153
 - SetSteelDesignDisplayParameters_V153
 - SetTimberDesignDisplayParameters_V153
 - SetVibrationDisplayParameters_V153
- IAxisVMLoadCases
 - SeismicGroupIDs
 - GetSeismicParams_V153
 - SetSeismicParams_V153
 - SeismicSpectrumH
 - SeismicSpectrumV
- IAxisVMSpectrum
 - GetSpectrumData_V153
 - SetSpectrumData_V153
- IAxisVMLoads
 - CreateStandardSeismicLoads_V153

Obsolete functions:

- IAxisVMLineSupport
 - GetFootingDimensions
 - GetFootingParams
- IAxisVMNodalSupport
 - GetFootingDimensions
 - GetFootingParams
- IAxisVMNodalSupports
 - AddNodalBeamRelative
 - AddNodalEdgeRelative
 - AddNodalGlobal
 - AddNodalLocal
 - AddNodalReference
- IAxisVMNodesSupports
 - AddNodalGlobal
 - AddNodalMemberRelative
 - AddNodalDomainRelative
 - AddNodalReferenceRelative
 - GetFootingParams
- IAxisVMMembersSupports
 - GetFootingDimensions
 - GetFootingParams
- IAxisVMWindows
 - GetBucklingDisplayParameters

- GetDynamicDisplayParameters
 - GetRCDesignDisplayParameters
 - GetStaticDisplayParameters
 - GetSteelDesignDisplayParameters
 - GetTimberDesignDisplayParameters
 - GetVibrationDisplayParameters
 - SetBucklingDisplayParameter
 - SetDynamicDisplayParameters
 - SetRCDesignDisplayParameters
 - SetStaticDisplayParameters
 - SetSteelDesignDisplayParameters
 - SetTimberDesignDisplayParameters
 - SetVibrationDisplayParameters
- IAxisVMWindow
 - GetBucklingDisplayParameters
 - GetDynamicDisplayParameters
 - GetRCDesignDisplayParameters
 - GetStaticDisplayParameters
 - GetSteelDesignDisplayParameters
 - GetTimberDesignDisplayParameters
 - GetVibrationDisplayParameters
 - SetBucklingDisplayParameter
 - SetDynamicDisplayParameters
 - SetRCDesignDisplayParameters
 - SetStaticDisplayParameters
 - SetSteelDesignDisplayParameters
 - SetTimberDesignDisplayParameters
 - SetVibrationDisplayParameters
- IAxisVMLoadCases
 - GetSeismicParams
 - SetSeismicParams
- IAxisVMSpectrum
 - GetSpectrumData
 - SetSpectrumData
- IAxisVMLoads
 - CreateStandardSeismicLoads

New properties:

- IAxisVMNodalSupport
 - SpringParams
- IAxisVMNodesSupports
 - SpringParams
- IAxisVMSpectrum
 - Disabled
- IAxisVMLoadCases
 - SeismicGroupID

AxisVM COM Plugin, Addon or AddonPlugin

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 for AxisVM. Depending on what functions are exported (Win32/64) and where it is (which subfolder of the AxisVM), it can be a [Plugin](#), [Addon](#) or [AddonPlugin](#).

When AxisVM is launched it looks for *.DLL files in the Plugins folder (*<AxisVM installation folder>\plugins*) and its subfolders and in the Addons folder (*<AxisVM installation folder>\addons*) and its subfolders.

If you want to debug your client, you have to set the program parameter as explained in [Starting the COM server](#).

The main differences:

Plugins:

- The title (...MenuItemText) shows up in a plugins menu.
- The dll should be in the Plugins folder (*<AxisVM installation folder>\plugins*). If DLLs were grouped in subfolders under Plugins folder, the subfolder structure is automatically converted into submenus to allow building a hierarchy of plugins.
- Clicking on a menu title in the *Plugins* menu of AxisVM calls the *OnMenuItemClick_v2* (Win32/64) or *Plugin_Execute* (.NET) process.
- for .NET only: *Addon_ButtonPlaceFormId* should return -1 !

Addons:

- The button with icon shows up on a toolbar as a last button (depend on name of other addons in addons subfolder) and hint of the button is displayed when cursor hovers over button.
- The dll should be in the Addons subfolder (*<AxisVM installation folder>\addons*).
- Clicking on the addon's button calls the *OnAddOnExecute_v2* (Win32/64) or *Addon_Execute* (.NET) process.

AddonPlugins:

- The button with icon shows up on a toolbar as a last button and hint of the button is displayed when cursor hovers over button.
- The dll should be in the Plugins folder (*<AxisVM installation folder>\plugins*). If DLLs were grouped in subfolders under Plugins folder, the subfolder structure is automatically converted into submenus to allow building a hierarchy of Plugins / AddonPlugins.
- Clicking on the button calls the *OnAddOnExecute_v2* (Win32/64) or *Addon_Execute* (.NET) process and clicking on a menu title in the *Plugins* menu of AxisVM calls the *OnMenuItemClick_v2* (Win32/64) or *Plugin_Execute* (.NET) process.

AxisVM COM Plugin

If AxisVM finds a DLL in the plugin folder exporting plugin functions, then a new menu item is automatically created in the *Plugins* menu of AxisVM with the plugin title (MenuItem). Plugin can be started by clicking on this menu item.

Win 32/64 versions 2.0, 2.1, 2.2 & 2.3

v2.1, v2.2 and v2.3 plugins are almost the same as v2.0 plugins except they have an additional exported function(s).

64-bit version of AxisVM (AxisVM_x64.exe) can only open 64-bit dlls and the 32-bit version of AxisVM (AxisVM.exe) can only open 32-bit dlls.

Please note:

If DLL also exports functions of an AddOn, it will become an [AddonPlugin](#). Plugins are simple DLL files exporting these plugin functions:

C syntax:

```
//v2.0
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 SetAxisVMApplicationHandle(const unsigned int32 ApplicationHandle);
void OnMenuItemClick_v2(AxisVM::IAxisVMAplication* AxApp);
unsigned int32 IsMenuItemEnabled(AxisVM::IAxisVMAplication* AxApp);
unsigned int32 IsMenuItemVisible(AxisVM::IAxisVMAplication* AxApp);

// v2.1
unsigned int32 IsPluginModalWindow(void);

// v2.2
__stdcall void InitPlugin(AxisVM::IAxisVMAplication* AxApp);
__stdcall void DeinitPlugin(void);

// v2.3
__stdcall unsigned int32 GetTranslatedMenuItemTextW(const void *Buffer, unsigned int32 BufferSize, AxisVM::ELanguage PrgLang);
```

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 SetAxisVMApplicationHandle(const ApplicationHandle: THandle); stdcall;
procedure OnMenuItemClick_v2(AxApp: IAxisVMAplication); stdcall;
function IsMenuItemEnabled(AxApp: IAxisVMAplication) : DWORD; stdcall;
function IsMenuItemVisible(AxApp: IAxisVMAplication) : DWORD; stdcall;

// v2.1
function IsPluginModalWindow : DWORD; stdcall;
// v2.2
procedure InitPlugin (AxApp: IAxisVMAplication); stdcall;
procedure DeinitPlugin; stdcall;
// v2.3
function GetTranslatedMenuItemTextW(const Buffer: Pointer; BufferSize: DWORD; PrgLang: ELanguage) : DWORD; stdcall;
```

Functions to export

unsigned long	GetMenuItemTextA ([in] void* Buffer , [in] unsigned int32 BufferSize) Buffer plugin name BufferSize maximum number of bytes in the buffer Copies the plugin name into the Buffer. The name must contain 8-bit ANSI characters. Returns the actual length of the plugin name.
unsigned long	GetMenuItemTextW ([in] void* Buffer , [in] unsigned int32 BufferSize) Buffer plugin name BufferSize maximum number of bytes in the buffer Copies the plugin name into the Buffer. The name must contain 16-bit Unicode characters. Returns the actual length of the plugin name (in Unicode characters).
unsigned int32	GetTranslatedMenuItemTextW ([in] void* Buffer , [in] unsigned int32 BufferSize , [in] AxisVM::ELanguage PrgLang) Buffer plugin name BufferSize maximum number of bytes in the buffer PrgLang the current program language of AxisVM Copies the plugin name into the Buffer. The name must contain 16-bit Unicode characters (ended with two zero bytes). Returns the actual length of the plugin name (in Unicode characters). This function is called by AxisVM each time when the program language changed. If this function not exists in the DLL then it will be loaded as a 2.0 or 2.1 or 2.2 plugin.
unsigned long	GetPluginVersion Returns the plugin version (must be 2). If this function returns some other value, the plugin does not appear in the AxisVM Plugins menu.
void	OnMenuItemClick_v2 ([in] AxisVM::IAxisVMAApplication* AxApp) AxApp COM object pointer of the AxisVM application which initiated the execution This function is called when the user clicks the plugin menu item. Plugin is executed in AxisVM's main thread (the thread where AxisVM's message loop is running). If the plugin is an AxisVM COM client then it must use the IAxisVMAApplication argument otherwise (because of the single instance mode of the COM server) it will start another instance of AxisVM.
void	SetAxisVMMainFormHandle ([in] unsigned int32 FormHandle) FormHandle the handle of the AxisVM main window This function is called by AxisVM to pass the handle of the AxisVM main window.
void	SetAxisVMAplicationHandle ([in] unsigned int32 ApplicationHandle) ApplicationHandle the handle of the AxisVM application This function is called by AxisVM to pass the handle of the application.
unsigned long	IsMenuItemEnabled ([in] AxisVM::IAxisVMAApplication* AxApp) AxApp AxisVM COM object implementing IAxisVMAApplication interface This function is called by AxisVM when the plugin menu is displayed to determine whether the plugin's menu item can be enabled or not. If the return value is 0 then menu item is disabled otherwise enabled. Important NOTE: If COM module is not available then the item will not be enabled!

unsigned long	IsMenuItemVisible ([in] AxisVM::IAxisVMAApplication* AxApp)
	AxApp Axis VM COM object implementing IAxisVMAApplication interface
<i>This function is called by AxisVM when the plugin menu is displayed to determine whether the plugin's menu item can be visible or not. If the return value is 0, then menu item is hidden otherwise visible.</i>	
unsigned long	IsPluginModalWindow
	<i>This function is called by AxisVM to determine whether the plugin must be run as a so-called "Modal Window" or not. If the return value is 0 then it is not modal otherwise, it is modal. The modal plugins are only allowed for legacy reasons, for new developments non modal mode (i.e. return value 0) is highly recommended.</i>
<i>If this function does not exist in the DLL then it will be loaded as a 2.0 plugin and the plugin will be displayed as a non-modal window.</i>	
void	InitPlugin ([in] AxisVM::IAxisVMAApplication* AxApp)
	AxApp Axis VM COM object implementing IAxisVMAApplication interface
<i>This function is called by AxisVM when AxisVM is fully loaded (IAxisVMAApplication Loaded event is fired) and COM server is available. If this function not exists in the DLL then it will be loaded as a 2.0 or 2.1 plugin.</i>	
void	DeinitPlugin
	<i>This function is called by AxisVM before AxisVM unloads plugin dlls. If this function not exists in the DLL then it will be loaded as a 2.0 or 2.1 plugin.</i>

.NET

The .NET class library must implement all functions, processes and properties of the [IAxisVMDotNetAddonPluginInterface](#).

Addon_ButtonPlaceFormId should return -1 and the *Addon_IconBitmap* should return an empty array.

AxisVM .NET Plugin System v2.3 for .NET Framework 4.x

Please note this system was updated and this interface was preserved for compatibility with older plugins, all new .NET developers are strongly encouraged to implement [IAxisVMDotNetAddonPluginInterface_v1_0](#) interface instead of this one. It has more functionality and supports more versions of the .NET Framework.

Plugins are loaded the same way as plugins developed for in [AxisVM .NET Plugin System v2.0](#) but checked for implemented [IAxisVMDotNetPluginInterface_v2_3](#) interface. Plugins for [IAxisVMDotNetPluginInterface_v2](#) interface are checked and loaded.

The interface description can be found in the AxisVMDotNetPluginInterface_v2.3.FW4.dll (.NET Framework 4.x) Class Libraries installed with AxisVM.

Add these references to your project: AxisVMDotNetPluginInterface_v2.3.FW4.dll and Interop.AxisVM.FW4.dll for .NET Framework 4.0).

The interface declaration:

C#

```
public interface IAxisVMDotNetPluginInterface_v2_3
{
    int PluginVersion { get; }
    String MenuItemText { get; }
    int AxisVMMainFormHandle { get; set; }
    int AxisVMApplicationHandle { get; set; }
    int IsMenuItemEnabled (AxisVMApplicationClass iAxisVMApp);
    int IsMenuItemVisible (AxisVMApplicationClass iAxisVMApp);
    int MenuItemClick_v2 (AxisVMApplicationClass iAxisVMApp);
    int IsPluginModalWindow { get; }
    void InitPlugin(AxisVMApplicationClass iAxisVMApp);
    void DeinitPlugin(AxisVMApplicationClass iAxisVMApp);
    String GetTranslatedMenuItemText(ELanguage PrgLang);
}
```

VB

```
Public Interface IAxisVMDotNetPluginInterface_v2_3
    ReadOnly Property PluginVersion() As Integer
    ReadOnly Property MenuItemText() As String
    Property AxisVMMainFormHandle() As Integer
    Property AxisVMApplicationHandle() As Integer
    Function IsMenuItemEnabled(ByVal iAxisVMApp As AxisVMApplicationClass) As Integer
    Function IsMenuItemVisible(ByVal iAxisVMApp As AxisVMApplicationClass) As Integer
    Function MenuItemClick_v2(ByVal iAxisVMApp As AxisVMApplicationClass) As Integer
    ReadOnly Property IsPluginModalWindow() As Integer
    Sub InitPlugin(ByVal iAxisVMApp As AxisVMApplicationClass)
    Sub DeinitPlugin(ByVal iAxisVMApp As AxisVMApplicationClass)
        Function GetTranslatedMenuItemText(PrgLang As ELanguage) As String
    End Interface
```

VC++

```
public interface class IAxisVMDotNetPluginInterface_v2_3
{
    property int PluginVersion { int get(); };
    property String^ MenuItemText { String^ get(); };
    property int AxisVMMainFormHandle;
    property int AxisVMApplicationHandle;
    int IsMenuItemEnabled(AxisVMApplicationClass ^ iAxisVMApp)
    int IsMenuItemVisible(AxisVMApplicationClass ^ iAxisVMApp)
    int MenuItemClick_v2(AxisVMApplicationClass ^ iAxisVMApp);
    property int IsPluginModalWindow { int get(); };
    void InitPlugin(AxisVMApplicationClass ^ iAxisVMApp)
    void DeinitPlugin(AxisVMApplicationClass ^ iAxisVMApp)
    String^ GetTranslatedMenuItemText(ELanguage PrgLang)
};
```

Functions to export

long **PluginVersion**

Returns the plugin version (must be 2).

If this function returns some other value, the plugin does not appear in the AxisVM Plugins menu.

String **MenuItemText**

Unicode string that represents the name of the plugin. This will be displayed in AxisVM plugin menu as menu item.

long **AxisVMMainFormHandle**

AxisVM will set this property to the AxisVM's main form handle.

long **AxisVMApplicationHandle**

AxisVM will set this property to the AxisVM's application handle (this is a hidden form). More info about it: <http://stackoverflow.com/questions/2204804/delphi-what-is-application-handle>

long	IsMenuItemEnabled ([in] AxisVM::IAxisVMAApplication* AxApp)	
	AxApp	AxisVM COM object of the application in which this plugin resides
<i>This function is called by AxisVM when the plugin menu is displayed to determine whether the plugin's menu item can be enabled or not. If the return value is 0 then menu item is disabled otherwise enabled.</i>		
long	IsMenuItemVisible ([in] AxisVM::IAxisVMAApplication* AxApp)	
	AxApp	AxisVM COM object of the application in which this plugin resides
<i>This function is called by AxisVM when the plugin menu is displayed to determine whether the plugin's menu item can be visible or not. If the return value is 0, then menu item is hidden otherwise visible.</i>		
long	MenuItemClick_v2 ([in] AxisVM::IAxisVMAApplication* AxApp)	
	AxApp	COM object pointer of the AxisVM application which initiated the execution
<i>This function is called when the user clicks the plugin menu item. The plugin is executed in AxisVM's main thread (the thread where AxisVM's message loop is running). Return value currently is not handled but for future usage the plugin should return 0 if it executed successfully.</i>		
long	IsPluginModalWindow	
<i>This function is called by AxisVM to determine whether the plugin must be run as a so-called "Modal Window" or not. If the return value is 0 then it is not modal otherwise, it is modal. The modal plugins are only allowed for legacy reasons, for new developments non modal mode (i.e. return value 0) is highly recommended.</i>		
void	InitPlugin ([in] AxisVM::IAxisVMAApplication* AxApp)	
	AxApp	AxisVM COM object of the application in which this plugin resides
<i>This function is called by AxisVM when AxisVM is fully loaded (IAxisVMAApplication Loaded event is fired) and COM server is available.</i>		
void	DeinitPlugin	
<i>This function is called before AxisVM unloads plugin dlls when AxisVM application is shutting down.</i>		
String	GetTranslatedMenuItemText ([in] ELanguage PrgLang)	
	PrgLang	the current program language of AxisVM
<i>This function is called by AxisVM each time when the program language changed. The plugin can specify language specific plugin menu item text.</i>		

AxisVM .NET Plugin System v2.0 for .NET Framework 2.x, 3.x

Please note this system was updated and this interface was preserved for compatibility with older plugins, all new .NET developers are strongly encouraged to implement [IAxisVMDotNetAddonPluginInterface_v1_0](#) interface instead of this one. It has more functionality and supports more versions of .NET Framework.

This system was updated to [newer](#) version and this version was preserved for compatibility with older plugins. .NET Class Libraries are loaded and checked for implemented [AxisVMDotNetPluginInterface_v2](#) or [IAxisVMDotNetPluginInterface_v2_3](#) interface.

The interface description can be found in the AxisVMDotNetPluginInterface_v2.dll Class Library installed with AxisVM.

Add these references to your project: AxisVMDotNetPluginInterface_v2.dll and Interop.AxisVM.dll

The interface declaration:

C#

```
public interface IAxisVMDotNetPluginInterface_v2
{
    int PluginVersion { get; }
    String MenuItemText { get; }
    int AxisVMMainFormHandle { get; set; }
    int AxisVMAplicationHandle { get; set; }
    int IsMenuItemEnabled(AxisVMAplicationClass iAxisVMApp);
    int IsMenuItemVisible(AxisVMAplicationClass iAxisVMApp);
    int MenuItemClick_v2(AxisVMAplicationClass iAxisVMApp);
}
```

VB

```
Public Interface IAxisVMDotNetPluginInterface_v2
    ReadOnly Property PluginVersion() As Integer
    ReadOnly Property MenuItemText() As String
    Property AxisVMMainFormHandle() As Integer
    Property AxisVMAplicationHandle() As Integer
    Function IsMenuItemEnabled(ByVal iAxisVMApp As AxisVMAplicationClass) As Integer
    Function IsMenuItemVisible(ByVal iAxisVMApp As AxisVMAplicationClass) As Integer
    Function MenuItemClick_v2(ByVal iAxisVMApp As AxisVMAplicationClass) As Integer
End Interface
```

VC++

```
public interface class IAxisVMDotNetPluginInterface_v2
{
    property int Pluginversion { int get(); };
    property String^ MenuItemText { String^ get(); };
    property int AxisVMMainFormHandle;
    property int AxisVMAplicationHandle;
    int IsMenuItemEnabled(AxisVMAplicationClass ^ iAxisVMApp)
    int IsMenuItemVisible(AxisVMAplicationClass ^ iAxisVMApp)
    int MenuItemClick_v2(AxisVMAplicationClass ^ iAxisVMApp);
};
```

For description of each function, see functions with same name in [AxisVM .NET Plugin System v2.3](#)

AxisVM COM Addon

When AxisVM is launched it looks for *.DLL files in the addons folder (*<AxisVM installation folder>\addons*).

Please note:

If DLL also exports functions of [plugins](#) and DLL is in the *plugins* folder, then it will be recognized as an **AddonPlugin** and AxisVM will display the icon of AddonPlugin and show the new plugin item in the plugins menu of AxisVM.

The AddonPlugin can be launched by calling any of these two procedures: *OnMenuItemClick_v2* or *OnAddOnExecute*.

Win 32/64 versions 1.0 and 2.0

64-bit version of AxisVM (AxisVM_x64.exe) can only open 64-bit dlls and the 32-bit version of AxisVM (AxisVM.exe) can only open 32-bit dlls.

Addons are simple DLL files exporting these addon functions:

C syntax:

```
unsigned int __stdcall GetAddOnVersion(void);
unsigned int __stdcall GetHintTextW(const void * Buffer, unsigned int BufferSize);
unsigned int __stdcall GetIconBitmap(const void * Buffer, unsigned int BufferSize, unsigned int * TransparentColor);
unsigned int __stdcall IsAddOnModalWindow(void);
void __stdcall GetButtonPlace(unsigned int * FormId, unsigned int * ToolbarId);
void __stdcall SetFormHandle(const unsigned int FormHandle);
void __stdcall OnAddOnExecute(AxisVM::IAxisVMApplcation * iAxApp);

// version 2.0: functions below are available only from AxisVM 12 release 2
unsigned int __stdcall IsItemEnabled (AxisVM::IAxisVMApplcation * iAxApp);
unsigned int __stdcall IsItemVisible (AxisVM::IAxisVMApplcation * iAxApp);
void __stdcall InitAddon (AxisVM::IAxisVMApplcation * iAxApp);
void __stdcall DeinitAddon;
unsigned int __stdcall GetTranslatedHintTextW (const void *Buffer, unsigned int32 BufferSize,
AxisVM::ELanguage PrgLang);
void __stdcall SetAxisVMMainFormHandle(const unsigned int32 FormHandle);
void __stdcall SetAxisVMApplcationHandle(const unsigned int32 ApplicationHandle);
```

Pascal syntax:

```
function GetAddOnversion : DWORD; stdcall;
function GetHintTextW(const Buffer: Pointer; BufferSize: DWORD) : DWORD; stdcall;
function GetIconBitmap(const Buffer: Pointer; BufferSize: DWORD; var TransparentColor: DWORD) :
DWORD; stdcall;
function IsAddOnModalWindow : DWORD; stdcall;
procedure GetButtonPlace(var FormId: DWORD; var ToolbarId: DWORD); stdcall;
procedure SetFormHandle(const Handle: THandle); stdcall;
procedure OnAddOnExecute(const AxApp: IAxisVMApplcation); stdcall;

// version 2.0: functions below are available only from AxisVM 12 release 2
function IsItemEnabled(const AxApp: IAxisVMApplcation): DWORD; stdcall;
function IsItemVisible(const AxApp: IAxisVMApplcation): DWORD; stdcall;
procedure InitAddon (const AxApp: IAxisVMApplcation); stdcall;
procedure DeinitAddon; stdcall;
function GetTranslatedHintTextW (const Buffer: Pointer; BufferSize: DWORD; PrgLang: ELanguage) :
DWORD; stdcall;
procedure SetAxisVMMainFormHandle(const FormHandle: THandle); stdcall;
procedure SetAxisVMApplcationHandle(const ApplicationHandle: THandle); stdcall;
```

If AxisVM finds a DLL in the addon folder, following functions are exported, and **GetAddOnVersion** returns the appropriate value, then new toolbar button is created on the specified form's toolbar. Addon can be called by clicking on this button and the *OnAddOnExecute* procedure will be executed.

DLLs in subfolders under *<AxisVM installation folder>\addons* folder will be loaded too.

Functions to export

unsigned int **GetAddOnVersion**

Returns the addon version 1 or 2.

If this function returns some other value then the addon will not be loaded.

unsigned int **GetHintTextW ([in] void * Buffer, [in] unsigned int BufferSize)**

Buffer *Addon button's hint text*

BufferSize *maximum number of bytes in the buffer*

Copies the addon button's hint text into the Buffer. The hint text must contain 16-bit Unicode characters. Returns the actual length of the hint text in Unicode chars.

unsigned int **GetIconBitmap ([in] void * Buffer, [in] unsigned int BufferSize, [out] unsigned int * TransparentColor)**

Buffer *memory pointer to hold a Windows Bitmap file*

BufferSize *maximum number of bytes in the buffer*

TransparentColor *this RGB colour will be used as transparent colour on the bitmap*

Copies the addon's icon into the buffer. The buffer must contain a Windows Bitmap file (24bit RGB, 22x22 pixels).

Returns the actual length of the file in the buffer.

TransparentColor also must be set. If TransparentColor is 0xFFFFFFFF then bitmap will not have transparent pixels. Otherwise TransparentColor is in 0x00RRGGBB format where RR = Red, GG = Green and BB = Blue component of the RGB palette (E.g. 0x00FF0000 is full red and 0x00FFFFFF is white) and pixels having this colour will be transparent.

If this function returns 0 value then no icon will be defined for the addon and AxisVM's default addon icon will be displayed on the button (this case TransparentColor is not used).

unsigned int **IsAddOnModalWindow**

This function is called by AxisVM to determine whether the addon must be run as a so-called "Modal Window" or not. Determines whether the addon is a modal window (AxisVM application/forms will be disabled while addon is running including the selection toolbar) or a non-modal window (AxisVM application/forms can be Accessed while addon is running). To enable/disable the AxisVM form use procedures EnableMainForm and DisableMainForm in [IAxisVMAApplication](#) interface. The modal plugins are only allowed for legacy reasons, for new developments non modal mode (i.e. return value 0) is highly recommended.

If the return value is 0 then it is not modal otherwise, it is modal.

void **GetButtonPlace ([out] unsigned int * FormId, [out] unsigned int * ToolbarId)**

FormId *the id of the form where addon button will be displayed.*

[List of IDs](#)

ToolbarId *the id of the toolbar in the form where addon button will be displayed. [List of IDs](#)*

This function is called by AxisVM determine where to display addon toolbar button. Addon buttons will be added at the end of original AxisVM toolbars. The order of the buttons on a toolbar is determined by the full filename (including path since addon dlls can be in subfolders inside the "addon" folder).

FormId and ToolbarId values are listed below.

void **SetFormHandle ([in] unsigned int FormHandle)**

FormHandle *the handle of the calling window*

This function is called by AxisVM to pass the handle of the form window where the button is.

void **OnAddOnExecute** ([in] AxisVM::IAxisVMAApplication * **AxApp**)
AxApp COM object pointer of the AxisVM application which initiated the execution

This function is called when the user clicks the addon button. The addon is executed in AxisVM's main thread (the thread where AxisVM's message loop is running). **If the addon is an AxisVM COM client then it must use the IAxisVMAApplication argument otherwise (because of the single instance mode of the COM server) it will start another instance of AxisVM.**

unsigned int **IsItemEnabled**([in] AxisVM::IAxisVMAApplication * **AxApp**)
AxApp AxisVM COM object pointer implementing IAxisVMAApplication interface

This function is called by AxisVM when the addon's icon should be displayed to determine whether the addon's icon can be enabled or not. If the return value is 0 then the addon's icon is disabled otherwise enabled.

Important NOTE:

If COM module is not available then the item will not be enabled!

unsigned int **IsItemVisible** ([in] AxisVM::IAxisVMAApplication * **AxApp**)
AxApp AxisVM COM object pointer implementing IAxisVMAApplication interface

This function is called by AxisVM whether the addon's icon should be displayed or not. If the return value is 0 then the addon's icon is hidden otherwise it is displayed.

void **InitAddOn** ([in] AxisVM::IAxisVMAApplication* **AxApp**)
AxApp AxisVM COM object implementing IAxisVMAApplication interface

This function is called by AxisVM when AxisVM is fully loaded (IAxisVMAApplication Loaded event is fired) and COM server is available.

void **DeinitAddOn**
This function is called by AxisVM before AxisVM unloads addon dlls.

unsigned int32 **GetTranslatedHintTextW** ([in] void* **Buffer**, [in] unsigned int32 **BufferSize**, [in] AxisVM::ELanguage **PrgLang**)
Buffer plugin name
BufferSize maximum number of bytes in the buffer
PrgLang the current program language of AxisVM

Copies the addon hint (text) into the Buffer. The hint (text) must contain 16-bit Unicode characters (ended with two zero bytes). Returns the actual length of the hint (in Unicode characters). Hint is shown when cursor hovers over the Addon's button.
This function is called by AxisVM each time when the program language changed.

void **SetAxisVMMainFormHandle** ([in] unsigned int32 **FormHandle**)
FormHandle the handle of the AxisVM main window
This function is called by AxisVM to pass the handle of the AxisVM main window.

void **SetAxisVMApplicationHandle** ([in] unsigned int32 **ApplicationHandle**)
ApplicationHandle the handle of the AxisVM application
This function is called by AxisVM to pass the handle of the application.

GetButtonPlace **FormId** and **ToolbarId** values:

FormId

0 AxisVM's main form

ToolbarId

- 0 geometry
- 1 elements
- 2 loads
- 3 mesh
- 4 static
- 5 buckling
- 6 vibration
- 7 dynamic
- 8 RC design
- 9 steel design
- 10 timber design

1 Cross Section editor form

ToolbarId

- 0 thin walled cross sections
- 1 solid cross sections

E.g. if FormId = 1 and ToolbarId = 1 then addon button will be displayed in the cross section editor form's solid cross section toolbar.

.NET (**IAxisVMDotNetAddonPluginInterface_v1.0**)

Implement all functions, processes and properties of the [IAxisVMDotNetAddonPluginInterface](#)

AxisVM COM AddonPlugin

AxisVM will show the button with the icon of this AddonPlugin on a toolbar and the title in the *Plugins* menu of the AxisVM.

Win32/64 version 1.0

The AddonPlugin can be started by calling one of these functions: *OnMenuItemClick_v2* or *OnAddOnExecute*.

64-bit version of AxisVM (AxisVM_x64.exe) can only open 64-bit dlls and the 32-bit version of AxisVM (AxisVM.exe) can only open 32-bit dlls.

AddonPlugins are simple DLL files exporting functions of both [Plugins](#) and [Addons](#):

C syntax:

```
// v1.0
// plugin functions
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 SetAxisVMAplicationHandle(const unsigned int32 ApplicationHandle);
void OnMenuItemClick_v2(AxisVM::IAxisVMAplication* AxApp);
unsigned int32 IsMenuItemEnabled(AxisVM::IAxisVMAplication* AxApp);
unsigned int32 IsMenuItemVisible(AxisVM::IAxisVMAplication* AxApp);
unsigned int32 IsPluginModalWindow(void);
__stdcall void InitPlugin(AxisVM::IAxisVMAplication* AxApp);
__stdcall void DeinitPlugin(void);
__stdcall unsigned int32 GetTranslatedMenuItemTextW(const void *Buffer, unsigned int32 BufferSize,
AxisVM::ELanguage PrgLang);

//addon functions
unsigned int __stdcall GetAddOnversion(void);
unsigned int __stdcall GetHintTextW(const void * Buffer, unsigned int BufferSize);
unsigned int __stdcall GetIconBitmap(const void * Buffer, unsigned int BufferSize, unsigned int * TransparentColor);
unsigned int __stdcall IsAddonModalWindow(void);
void __stdcall GetButtonPlace(unsigned int * FormId, unsigned int * ToolbarId);
void __stdcall SetFormHandle(const unsigned int FormHandle);
void __stdcall OnAddOnExecute(AxisVM::IAxisVMAplication * iAxApp);
unsigned int __stdcall IsItemEnabled (AxisVM::IAxisVMAplication * iAxApp);
unsigned int __stdcall IsItemVisible (AxisVM::IAxisVMAplication * iAxApp);
void __stdcall InitAddon (AxisVM::IAxisVMAplication * iAxApp);
void __stdcall DeinitAddon;
unsigned int __stdcall GetTranslatedHintTextw (const void *Buffer, unsigned int32 BufferSize,
AxisVM::ELanguage PrgLang);
```

Pascal syntax:

```
// v1.0
// plugin functions
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 SetAxisVMAplicationHandle(const ApplicationHandle: THandle); stdcall;
procedure OnMenuItemClick_v2(AxApp: IAxisVMAplication); stdcall;
function IsMenuItemEnabled(AxApp: IAxisVMAplication) : DWORD; stdcall;
function IsMenuItemVisible(AxApp: IAxisVMAplication) : DWORD; stdcall;
function IsPluginModalWindow : DWORD; stdcall;
procedure InitPlugin (AxApp: IAxisVMAplication); stdcall;
procedure DeinitPlugin; stdcall;
function GetTranslatedMenuItemTextW(const Buffer: Pointer; BufferSize: DWORD; PrgLang: ELanguage) : DWORD; stdcall;
```

```
// addon functions
function GetAddonVersion : DWORD; stdcall;
function GetHintTextW(const Buffer: Pointer; BufferSize: DWORD) : DWORD; stdcall;
function GetIconBitmap(const Buffer: Pointer; BufferSize: DWORD; var TransparentColor: DWORD) : DWORD; stdcall;
function IsAddOnModalWindow : DWORD; stdcall;
procedure GetButtonPlace(var FormId: DWORD; var ToolbarId: DWORD); stdcall;
procedure SetFormHandle(const Handle: THandle); stdcall;
procedure OnAddonExecute(const AXApp: IAxisVMApplication); stdcall;
function IsItemEnabled(const AXApp: IAxisVMApplication): DWORD; stdcall;
function IsItemVisible(const AXApp: IAxisVMApplication): DWORD; stdcall;
procedure InitAddon (const AXApp: IAxisVMApplication); stdcall;
procedure DeinitAddon; stdcall;
function GetTranslatedHintTextW (const Buffer: Pointer; BufferSize: DWORD; PrgLang: ELanguage) : DWORD; stdcall;
```

.NET (IAxisVMDotNetAddonPluginInterface_v1.0)

Plugins are checked for implemented [IAxisVMDotNetAddonPluginInterface_v1_0](#) interface.

The interface description can be found in the AxisVMDotNetAddonPluginInterface_v1.0.FW4.dll (.NET Framework 4.x) Class Libraries installed with AxisVM.

Add these references to your project: AxisVMDotNetAddonPluginInterface_v1.0.FW4.dll and Interop.AxisVM.FW4.dll for the .NET Framework 4.x

The AddonPlugin can be started by calling one of these functions: *Plugin_Execute* or *Addon_Execute*.

All functions, processes and properties of this interface must be implemented!

The interface declaration:

C#

```
public interface IAxisVMDotNetAddonPluginInterface_v1_0
{
    // common
    int Version { get; }
    int SubVersion { get; }
    int AxisVMMainFormHandle { get; set; }
    int AxisVMAApplicationHandle { get; set; }

    // plugin
    int Plugin_Execute (AxisVMAApplicationClass iAxisVMAApp);
    string Plugin_GetTranslatedMenuItemText(int PrgLang);
    void Plugin_Deinit(AxisVMAApplicationClass iAxisVMAApp);
    void Plugin_Init(AxisVMAApplicationClass iAxisVMAApp);
    int Plugin_IsMenuItemEnabled(AxisVMAApplicationClass iAxisVMAApp);
    int Plugin_IsMenuItemVisible(AxisVMAApplicationClass iAxisVMAApp);
    int Plugin_IsModalWindow { get; }
    String Plugin_MenuItemText { get; }

    // addon
    int Addon_ButtonPlaceFormId { get; }
    int Addon_ButtonPlaceToolbarId { get; }
    int Addon_Execute(AxisVMAApplicationClass iAxisVMAApp);
    int Addon_FormHandle { get; set; }
    string Addon_GetTranslatedHintText (int PrgLang);
    String Addon_HintText { get; }
    byte[] Addon_IconBitmap { get; }
    int Addon_IconBitmapTransparentColor { get; }
    int Addon_IsModalWindow { get; }
    void Addon_Deinit(AxisVMAApplicationClass iAxisVMAApp);
    void Addon_Init(AxisVMAApplicationClass iAxisVMAApp);
    int Addon_IsItemEnabled(AxisVMAApplicationClass iAxisVMAApp);
    int Addon_IsItemVisible(AxisVMAApplicationClass iAxisVMAApp);

}
```

VB

```
Public Interface IAxisVMDotNetAddonPluginInterface_v1_0
    ' common
    ReadOnly Property Version () As Integer
    ReadOnly Property SubVersion () As Integer
    Property AxisVMMainFormHandle () As Integer
    Property AxisVMAApplicationHandle () As Integer

    ' plugin
    Function Plugin_Execute (ByVal iAxisVMAApp As AxisVMAApplicationClass) As Integer
    ReadOnly Property Plugin_GetTranslatedMenuItemText (PrgLang as Integer) As String
    sub Plugin_Deinit (ByVal iAxisVMAApp As AxisVMAApplicationClass)
    sub Plugin_Init (ByVal iAxisVMAApp As AxisVMAApplicationClass)
    Function Plugin_IsMenuItemEnabled(ByVal iAxisVMAApp As AxisVMAApplicationClass) As Integer
    Function Plugin_IsMenuItemVisible(ByVal iAxisVMAApp As AxisVMAApplicationClass) As Integer
    ReadOnly Property Plugin_IsModalWindow () As Integer
    ReadOnly Property Plugin_MenuItemText () As String

    ' addon
    ReadOnly Property Addon_ButtonPlaceFormId As Integer
    ReadOnly Property Addon_ButtonPlaceToolbarId As Integer
    Function Addon_Execute (ByVal iAxisVMAApp As AxisVMAApplicationClass) As Integer
    Property Addon_FormHandle () As Integer
    Function Addon_GetTranslatedHintText(PrgLang As Integer) As String
    ReadOnly Property Addon_HintText As String
    Function Addon_IconBitmap As byte()
    Function Addon_IconBitmapTransparentColor As Integer
    Function Addon_IsModalwindow As Integer
    sub Addon_Deinit (ByVal iAxisVMAApp As AxisVMAApplicationClass)
    sub Addon_Init (ByVal iAxisVMAApp As AxisVMAApplicationClass)
    Function Addon_IsItemEnabled(iAxisVMAApp As AxisVMAApplicationClass) As Integer
    Function Addon_IsItemVisible (iAxisVMAApp As AxisVMAApplicationClass) As Integer

End Interface
```

```

VC++
public interface class IAxisVMDotNetAddonPluginInterface_v1_0
{
    // common
    property int Version { int get(); };
    property int Subversion { int get(); };
    property int AxisVMMainFormHandle;
    property int AxisVMApplicationHandle;

    // plugin
    int Plugin_Execute (AxisVMApplicationClass ^ iAxisVMApp);
    property String^ Plugin_GetTranslatedMenuItemText(int PrgLang) { String^ get(); };
    void Plugin_Deinit(AxisVMApplicationClass ^ iAxisVMApp);
    void Plugin_Init(AxisVMApplicationClass ^ iAxisVMApp);
    int Plugin_IsMenuItemEnabled(AxisVMApplicationClass ^ iAxisVMApp)
    int Plugin_IsMenuItemVisible(AxisVMApplicationClass ^ iAxisVMApp)
    property int Plugin_IsModalwindow { int get(); };
    property String^ Plugin_MenuItemText { String^ get(); };

    // addon
    property int Addon_ButtonPlaceFormId { int get(); };
    property int Addon_ButtonPlaceToolbarId { int get(); };
    int Addon_Execute (AxisVMApplicationClass ^ iAxisVMApp);
    property int Addon_FormHandle { int get(); set() };
    property String^ Addon_GetTranslatedHintText(int PrgLang)
    property String^ Addon_HintText
    property array<unsigned char>^ Addon_IconBitmap{ array<unsigned char>^ get(); };
    property int Addon_IconBitmapTransparentColor { int get(); };
    property int Addon_IsModalwindow { int get(); };
    void Addon_Deinit (AxisVMApplicationClass ^ iAxisVMApp);
    void Addon_Init (AxisVMApplicationClass ^ iAxisVMApp);
    int Addon_IsItemEnabled(AxisVMApplicationClass ^ iAxisVMApp)
    int Addon_IsItemVisible (AxisVMApplicationClass ^ iAxisVMApp)
};


```

Functions to export

long	Version
	<i>Must return 1</i>
long	SubVersion
	<i>Must return 0</i>
long	AxisVMMainFormHandle
	<i>AxisVM will set this property to the AxisVM's main form handle.</i>
long	AxisVMApplicationHandle
	<i>AxisVM will set this property to the AxisVM's application handle (this is a hidden form). More info about it: http://stackoverflow.com/questions/2204804/delphi-what-is-application-handle</i>
long	Plugin_Execute ([in] AxisVM::IAxisVMAppliation* AxApp)
	AxApp COM object pointer of the AxisVM application which initiated the execution
	<i>This function is called when the user clicks the plugin menu item. The AddonPlugin is executed in AxisVM's main thread (the thread where AxisVM's message loop is running). Return value currently is not handled but for future usage the AddonPlugin should return 0 if it executed successfully.</i>
String	Plugin_GetTranslatedMenuItemText ([in] ELanguage PrgLang)
	PrgLang the current program language of AxisVM
	<i>This function is called by AxisVM each time when the program language changed. The AddonPlugin can specify language specific menu item text.</i>
void	Plugin_Init ([in] AxisVM::IAxisVMAppliation* AxApp)
	AxApp AxisVM COM object of the application in which this AddonPlugin resides
	<i>This function is called by AxisVM when AxisVM is fully loaded (IAxisVMAppliation Loaded event is fired) and COM server is available.</i>

void	Plugin_Deinit ([in] AxisVM::IAxisVMAApplication* AxApp)	
	AxApp	AxisVM COM object of the application in which this AddonPlugin resides
<i>This function is called before AxisVM unloads plugin dlls when AxisVM application is shutting down.</i>		
long	Plugin_IsMenuItemEnabled ([in] AxisVM::IAxisVMAApplication* AxApp)	
	AxApp	AxisVM COM object of the application in which this AddonPlugin resides
<i>This function is called by AxisVM when the AddonPlugin menu is displayed to determine whether the menu item can be enabled or not. If the return value is 0 then menu item is disabled otherwise enabled.</i>		
long	Plugin_IsMenuItemVisible ([in] AxisVM::IAxisVMAApplication* AxApp)	
	AxApp	AxisVM COM object of the application in which this AddonPlugin resides
<i>This function is called by AxisVM when the AddonPlugin menu is displayed to determine whether the menu item can be visible or not. If the return value is 0, then menu item is hidden otherwise visible.</i>		
long	Plugin_IsModalWindow	
<i>Determines whether the AddonPlugin is a modal window (AxisVM application/forms will be disabled while AddonPlugin is running including the selection toolbar) or a non-modal window (AxisVM application/forms can be accessed while plugin is running). To enable/disable the AxisVM form use procedures EnableMainForm and DisableMainForm in IAxisVMAApplication interface. Modal AddonPlugins are only allowed for legacy reasons, for new developments non modal mode (i.e. return value 0) is highly recommended.</i>		
<i>A modal AddonPlugin must return 1 and a non-modal must return 0.</i>		
String	Plugin_MenuItemText	
<i>Unicode string that represents the name of the plugin. This will be displayed in AxisVM plugin menu as menu item.</i>		
long	Addon_ButtonPlaceFormId	
<i>Determines the id of the form where addon button will be displayed. List of Form IDs where to display the addon toolbar button.</i>		
long	Addon_ButtonPlaceToolbarId	
<i>Determines the id of the toolbar (tab) on the form, where AddonPlugin's button will be displayed. List of IDs where to display the AddonPlugin's toolbar button. AddonPlugin's button will be added at the end of original AxisVM toolbars. The order of the buttons on a toolbar is determined by the full filename (including path since AddonPlugin's dlls can be in subfolders inside the "addon" folder).</i>		
long	AddOn_Execute ([in] AxisVM::IAxisVMAApplication * AxApp)	
	AxApp	COM object pointer of the AxisVM application which initiated the execution
<i>This function is called when the user clicks the AddonPlugin button. The AddonPlugin is executed in AxisVM's main thread (the thread where AxisVM's message loop is running). Return value currently not handled but for future usage it should return 0 if it executed successfully.</i>		
long	Addon_FormHandle	
<i>This function is called by AxisVM to pass the handle of the form window where the button is.</i>		

String	Addon_GetTranslatedHintText ([in] ELanguage PrgLang)
	PrgLang <i>the current program language of AxisVM</i>
	<i>Returns AddonPlugin's hint (text) depending on the used language. Hint is shown when cursor hovers over the Addon's button. This function is called by AxisVM each time the program language has changed.</i>
String	Addon_HintText
	<i>Returns AddonPlugin's hint (text). Hint is shown when cursor hovers over the Addon's button.</i>
byte[]	Addon_IconBitmap
	<i>Determines the AddonPlugin's button icon. The transparent colour must be also set with Addon_IconBitmapTransparentColor. If this function does not return any data, then no icon will be defined and AxisVM's default addon icon will be displayed on the button (in this case TransparentColor is not used).</i>
long	Addon_IconBitmapTransparentColor
	<i>Determines the transparent colour of the button icon. If the transparent colour is 0xFFFFFFFF then bitmap won't have transparent pixels. Otherwise the transparent colour is in 0x00RRGGBB format where RR = Red, GG = Green and BB = Blue component of the RGB palette (E.g. 0x00FF0000 is full red and 0x00FFFFFF is white) and pixels having this colour will be transparent</i>
long	Addon_IsModalWindow
	<i>Determines whether the AddonPlugin is a modal window (AxisVM application/forms will be disabled while AddonPlugin is running including the selection toolbar) or a non-modal window (AxisVM application/forms can be accessed while plugin is running). To enable/disable the AxisVM form use procedures EnableMainForm and DisableMainForm in IAxisVMAplication interface. Modal AddonPlugins are only allowed for legacy reasons, for new developments non modal mode (i.e. return value 0) is highly recommended.</i>
	<i>A modal AddonPlugin must return 1 and a non-modal must return 0.</i>
long	Addon_Deinit ([in] AxisVM::IAxisVMAplication* AxApp)
	AxApp <i>AxisVM COM object of the application in which this AddonPlugin resides</i>
	<i>This function is called before AxisVM unloads Addon dlls when AxisVM application is shutting down.</i>
long	Addon_Init ([in] AxisVM::IAxisVMAplication* AxApp)
	AxApp <i>AxisVM COM object of the application in which this AddonPlugin resides</i>
	<i>This function is called by AxisVM when AxisVM is fully loaded (IAxisVMAplication Loaded event is fired) and COM server is available.</i>
long	Addon_IsItemEnabled ([in] AxisVM::IAxisVMAplication* AxApp)
	AxApp <i>AxisVM COM object of the application in which this AddonPlugin resides</i>
	<i>AxisVM will call this routine when AddonPlugin's button is about to be displayed. If it returns 0 then the button will be disabled otherwise it will be enabled.</i>
long	Addon_IsItemVisible ([in] AxisVM::IAxisVMAplication* AxApp)
	AxApp <i>AxisVM COM object of the application in which this AddonPlugin resides</i>
	<i>AxisVM will call this routine when AddonPlugin's button is about to be displayed. If it returns 0 then the button won't be displayed otherwise it will be displayed.</i>

Important Notes for .NET

All .NET plugins, addons and AddonPlugins must be recompiled with corresponding version of *Interop.AxisVM.dll* (.NET Framework 2.x and 3.x) or *Interop.AxisVM.FW4.dll* (.NET Framework 4.x and above). The version of interop file should match with the version of COM server, otherwise it will not be loaded. If continuous maintenance of a .NET plugin/addon cannot be guaranteed, a Win32/64 plugin/addon is the safe solution, as that doesn't require maintenance until a major update is implemented in the COM server. To keep the older plugins/addons working, such a major update is avoided until it becomes absolutely necessary.

The default platform target in compiler's build settings is **Any CPU**. If the default platform target is used, the generated IL code is taken to native code by the CLR at runtime using the just-in-time compiler depending on CPU of the PC where it is launched. The client will run in 64-bit mode on 64-bit machine and can launch the 32-bit version of AxisVM, if only 32-bit version is registered. This scenario could lead to misaligned data in data records/structures or other misbehaviour. The client running in 64-bit mode will launch 64-bit version of AxisVM if that is also registered.

The platform target should be set in accordance with the version (32/64 bit) of AxisVM that it is intended to use with. The safest approach is to use **x86** setting for platform target. The client then will run on 32 and 64 bit of AxisVM without any problems.

More about this topic here: <http://visualstudiohacks.com/articles/visual-studio-net-platform-target-explained/>

1. Copy your referenced dlls into the same folder where your dll is.
2. No need to copy the "interface dll" (*AxisVMDotNetPluginInterface_*.dll* or *AxisVMDotNetAddonPluginInterface_*.dll*) and "interop dll" (*Interop.AxisVM.*.dll*) because these dlls will be loaded automatically by AxisVM.
3. VC++ .NET developers: In project properties the 'Common Language Runtime support' must be set to "Safe MSIL Common Language Runtime Support (/clr:safe)" otherwise plugin won't be loaded !
4. SafeArrays in AxisVM always use 1 as lower bound (see ".NET how to use AxisVM COM server SafeArrays" section for more information)
5. Applications must to be compiled with the same version of reference dlls (*Interop.AxisVM.*.dll*) as the version of used COM server otherwise they will not be loaded.

Late binding and early binding

Binding is the process when an object is assigned to an object variable.

An object is *late bound* when it is assigned to a variable declared to be a generic object. The actual object information is resolved runtime.

An object is *early bound* when it is assigned to a variable declared to be of that specific object type. Early bound objects allow the compiler to allocate memory and perform other optimizations before an application executes.

You should use early-bound objects whenever possible, because they allow the compiler to make important optimizations that yield applications that are more efficient. Early-bound objects are significantly faster than late-bound objects and make your code easier to read and maintain by stating exactly what kind of objects are being used. Early binding reduces the number and severity of run-time errors because it allows the compiler to report errors when a program is compiled.

It is recommended to use AxisVM COM objects in the early binding model.

To get the constant names, object structures and function calling syntax in compile time the type library information has to be imported.

Find GUID of the record

Each record has its own unique identification number GUID. It can vary between different versions of COM server.

The easiest way to obtain a GUID of the record is shown [here](#).

Troubleshooting

AxisVM results

Issues may arise during reading AxisVM results in wide range of interfaces, when user tries to access non-existent or not-valid analysis results considering creep of concrete.

Concrete creep

Nonlinear analysis can be performed with and without considering creep of reinforced concrete elements. With this parameter enabled, user can select whether the calculated results during nonlinear analysis consider results with or without creep of concrete.

Wide range of error codes can be returned values by interface functions when user tries to read results considering creep of concrete, but analysis parameters are not set to consider this effect.

UserCreep property of various sub-interfaces of IAxisVMResults interface are set to a default value, which depends on whether the set national design code supports calculations considering creep of the reinforced concrete.

Development

Issues arised during development in various environments.

.NET

Assemblies

The assemblies added as references to the client should be copied to AxisVM folder or subfolder of AxisVM folder named by the assembly, e.g.: assembly *Newtonsoft.Json.dll* should be in subfolder <AxisVM installation folder>\Newtonsoft.Json

No-PIA mode

Compilation error:

vbc : error BC31541: Reference to class 'AxisVMApplicationClass' is not allowed when its assembly is linked using No-PIA mode.

Solution:

Open project properties, select *References* then *Interop.AxisVM.FW4* and in properties change *Embed Interop Types* to *False*.

Plugin not loaded (not shown in Plugins menu)

Possible ways to solve the problem:

1. Try to re-register the AxisVM.NET server:

[**IAxisVMDotNetPluginInterface_v2_3**](#)

run !UNREGISTER_DOT_NET_PLUGIN_SERVER_v2.3.BAT
then run !REGISTER_DOT_NET_PLUGIN_SERVER_v2.3.BAT

[**IAxisVMDotNetAddonPluginInterface_v1_0**](#)

run !UNREGISTER_DOT_NET_ADDONPLUGIN_SERVER_v1.0.BAT
then run !REGISTER_DOT_NET_ADDONPLUGIN_SERVER_v1.0.BAT

2. Set adequate target framework version in *Advanced compiler settings*.
3. Look in *PluginFinder.dll_ERROR.LOG* located in AxisVM folder for System.IO.FileNotFoundException exceptions logging which .NET assemblies are required for DLL.
4. See [Important Notes for .NET](#)

SafeArrays

The problem:

AxisVM COM server uses SafeArrays with lower bound = 1 but the default marshalling for arrays assumes lower bound = 0. That can lead to SafeArrayRankMismatchException exceptions.

Default Marshalling for Arrays

<http://msdn.microsoft.com/en-us/library/z6cfh6e6.aspx>

Troubleshooting Exceptions: System.Runtime.InteropServices.SafeArrayRankMismatchException
<http://msdn.microsoft.com/en-us/library/d334fhe8.aspx>

"Because the rank and bounds of a safe array cannot be determined from the type library, the rank is assumed to equal 1, and the lower bound is assumed to equal 0. The rank and bounds must be defined in the managed signature produced by the Type Library Importer (Tlbimp.exe)."

Solution:

Add installed 'AxisVM.Interop.dll' as a reference to your project

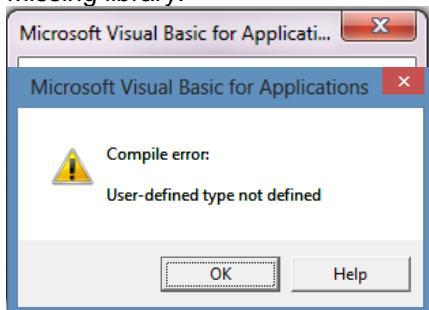
Example of using SafeArrays in VB.NET:

```
'declaration:  
    Dim lengths() As Integer = {2} 'this is for one-dimensional array with 2 elements  
    Dim lowerBounds() As Integer = {1} 'this specifies lower bound=1  
    Dim LineIDs As Array = Array.CreateInstance(GetType(Int32), lengths, lowerBounds)  
    'int32 = 4 byte integer value (long), see AxisVM COM data types  
  
'fill the array with SetValue function:  
    LineIDs.SetValue(11, 1) '1st index of array = 1, 11 = index of 1st line  
    LineIDs.SetValue(12, 2) '2nd index of array = 2, 12 = index of 2nd line
```

Excel

Users can get several types of error messages when the AxisVM type library is not properly linked to Excel like these:

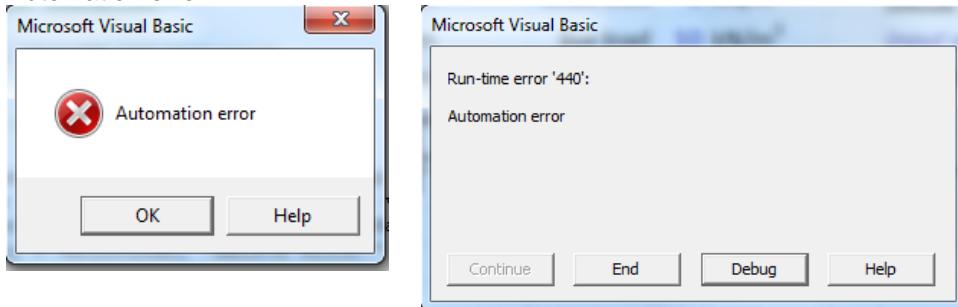
Missing library:



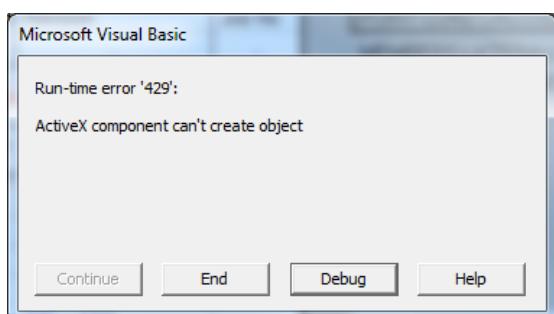
or Compile error:

Microsoft Visual Basic for Applications

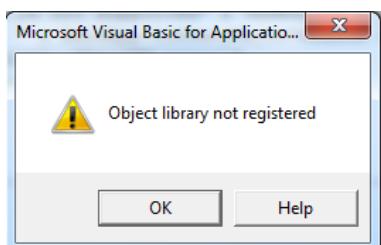
Automation error:



Run-time error



Object library error



Please note:

Other types of error messages can be also fixed by trying the steps below.

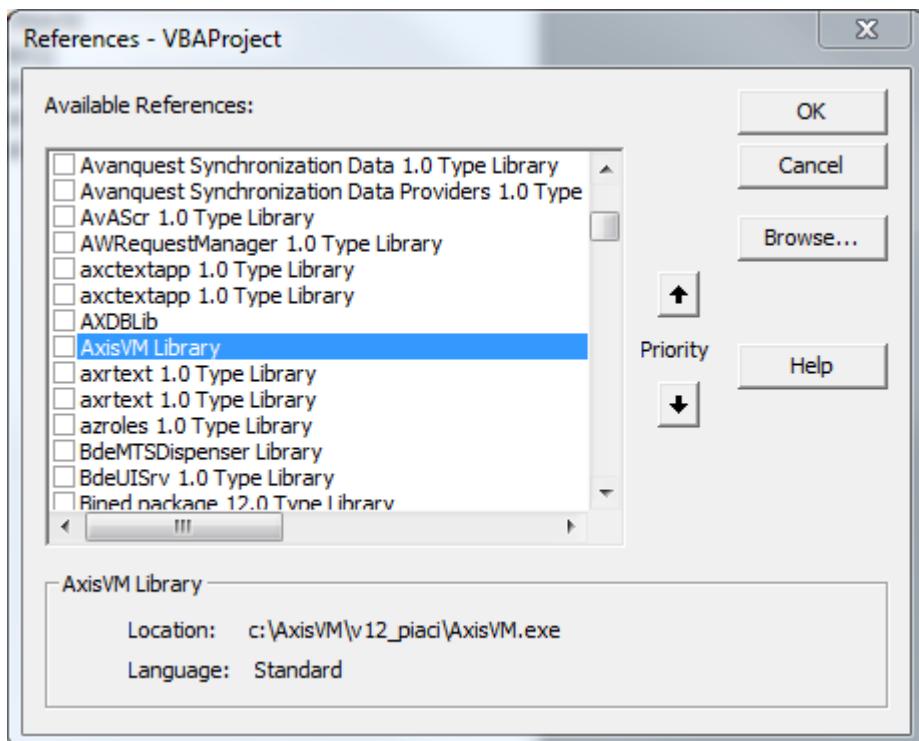
Solution:

For Excel 2007 and newer:

1. Enable the **Developer** tab if disabled
2. On the File tab, choose **Options** to open the **Excel Options** dialog box.
3. Click Customize Ribbon on the left side of the dialog box.
4. Under Choose commands from on the left side of the dialog box, select **Popular Commands**.
5. Under Customize the ribbon on the right side of the dialog box, select Main tabs, and then select the **Developer** check box.
6. Click **OK**.
7. Open xls file
8. Press Reset button (blue rectangle)
9. Go to: **Tools - References**
10. Unselect: *AxisVM Library (see picture below)*
11. Close *Excel*
12. Re-register AxisVM COM server
13. Start Excel and repeat steps 7 to 9
14. Roll down the list and select: *AxisVM Library (the AxisVM .Net versions are for different purposes, they must remain unchecked)*
15. Press **OK**
16. Close Visual Basic window

For Excel 95-2003:

1. Open xls file.
2. Press Reset button (blue rectangle)
3. Go to: **Tools - References**
4. Unselect: *AxisVM Library (see picture below)*
5. Close *Excel*
6. Re-register AxisVM COM server
7. Start Excel and repeat steps 1 to 3
8. Roll down the list and select: *AxisVM Library (the AxisVM .Net versions are for different purposes, they must remain unchecked)*
9. Press **OK**
10. Close Visual Basic window



Re-register AxisVM COM server:

Only in the case of matching versions (32bit AxisVM and 32 bit Excel) or (64bit AxisVM and 64 bit Excel) will AxisVM Library appear in *Available References* list, see picture above. By default AxisVM installs the 64 bit version, and Excel installs the 32 bit version, making the reference/type library not available. A quick solution is to (re)install AxisVM, selecting the 32 bit version (by purchasing AxisVM, you get access to both versions). The 64 bit version of AxisVM will still remain installed. Another solution would be to install a 64 bit Excel (available only starting with MS Office 2010) for the default 64 bit AxisVM.

If an AxisVM Library of the correct bit depth is registered, by selecting it, at the bottom part of the references window you will see the path to it. If it is not the path to your newest AxisVM, you should unregister the old one by running as administrator *!UNREGISTER_AXISVM.BAT* then *!UNREGISTER_AXISVM_X64.BAT* from the displayed AxisVM directory.

Finally, run as administrator *!REGISTER_AXISVM.BAT* from your newest/current AxisVM directory (default location: C:\AxisVM****, where **** depends on the AxisVM version), to register the 32 bit version of AxisVM or *!REGISTER_AXISVM_X64.BAT* to register the 64bit version. There is no harm registering both.

Python

Known issues

comtypes package

- minimum required version is 1.1.4, the older version has issues with return types)
- when COM function returns an empty safearray there will be an exception in safearray.py.
 - temporary fix: force safearrays to have length 1 with 0 values, call CustomFunction with this JSON string:

```
{  
    "InterfaceName": "IAxisVMAplication",  
    "FunctionName": "SetEmptySafeArrays",  
    "Value": False  
}
```

Check this option with JSON string:

```
{  
    "InterfaceName": "IAxisVMAplication",  
    "FunctionName": "SetEmptySafeArrays",  
    "Value": False  
}
```

Tips and tricks

- In order to have all data updated in AxisVM it is beneficiary to mimic the user interaction. Set the [tab](#) of the AxisVM's main form to the same, where you would perform the operations with your mouse, this is needed to perform several invalidations with the other input data, results, etc.
- Speed up reading and writing of results by using multiple element reader functions like *AllLineForcesByLoadCaseId* instead of single element reader functions like *LineForcesByLoadCaseId*.
- Use functions BeginUpdate and EndUpdate when you are modifying the model.
- Hide the AxisVM while modifying the model using property Visible in the [IAxisVMAplication](#) interface.
- Use global variables for instances of often used interfaces (IAxisVMAplication, IAxisVMMModels, IAxisVMMModel) instead of calling functions with full path. It can slow down COM server dramatically, see example below:

From Delphi example: Example #2: Truss model construction

WRONG:

In this case two instances of IAxisVMMModels interface , two instances of IAxisVMMModel interface and two instances of IAxisVMCrossSections interface will be created and released in the function

```
TrussCSId := fAxApp.Models.Item[n].CrossSections.AddFromCatalog(fTrussType, fTrussName);
ChordCSId := fAxApp.Models.Item[n].CrossSections.AddFromCatalog(fchordType, fchordName);
```

CORRECT:

In this case one instance of IAxisVMMModels interface , one instance of IAxisVMMModel interface which are global and one local instance of IAxisVMCrossSections interface is created

Only instance of IAxisVMCrossSections interface will be released when function ends the other two instances will be released when client is released from memory.

```
// gAxApp should be a global variable of the client application, AxApp is the COM object
// obtained as function parameter from OnMenuItemClick_v2 (plugin) or OnAddOnExecute (addon)
gAxApp := AxApp;

// gAxModels should be a global variable of the client application used for IAxisVMMModels
// events
gAxModels := gAxApp.Models;

// AxModel should be a global variable of the client application
AxModel := AxModels.Item[1];

// AxCrossSections can be a local variable of function working with AxisVM cross-sections
AxCrossSections := AxModel.CrossSections; // AxCrossSections is a local variable
// TrussCSId and ChordCSId is a local variable
TrussCSId := AxCrossSections.AddFromCatalog(fTrussType, fTrussName);
ChordCSId := AxCrossSections.AddFromCatalog(fchordType, fchordName);
```

Importing type library

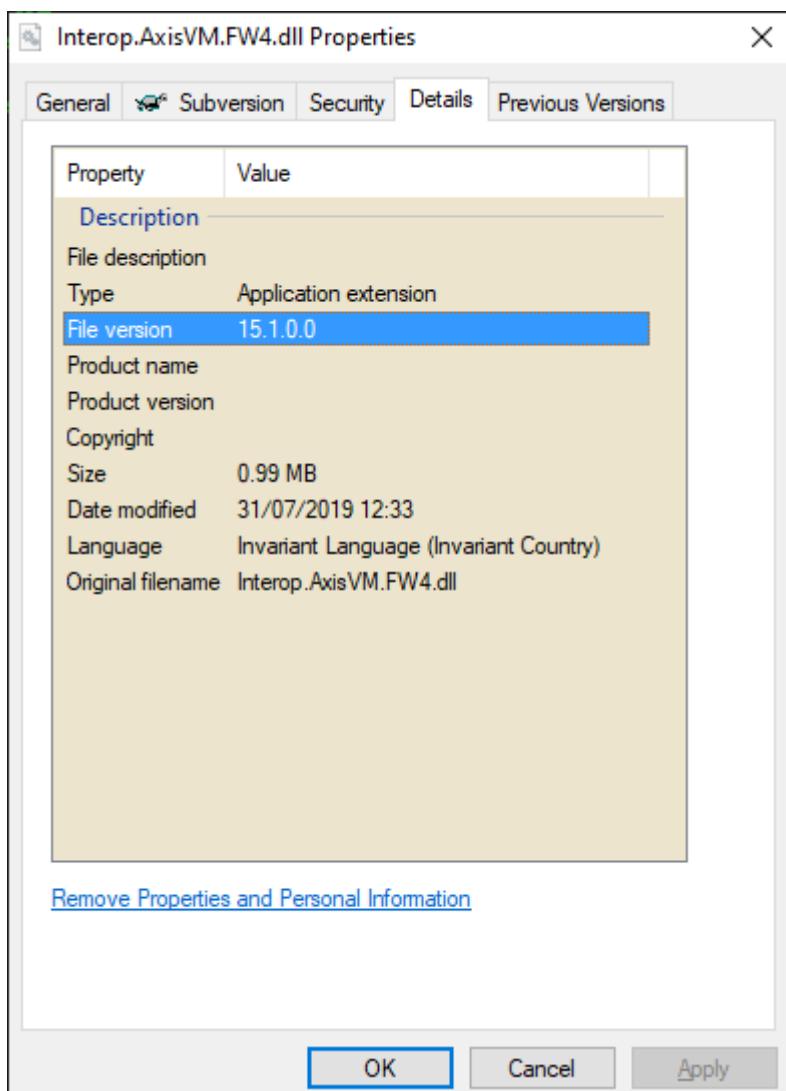
Type library should be imported or linked to the project in order to get the constant names, object structures and function calling syntax.

Some of 32-bit versions of integrated development environments IDE (or editor) will see only the registration of the 32-bit version of AxisVM.

If 64-bit version of AxisVM is registered only, then the AxisVM type library might not be listed in the 32-bit IDE (or editor). Register the 32-bit version to see the library in the list or use 64-bit version of the IDE (or editor).

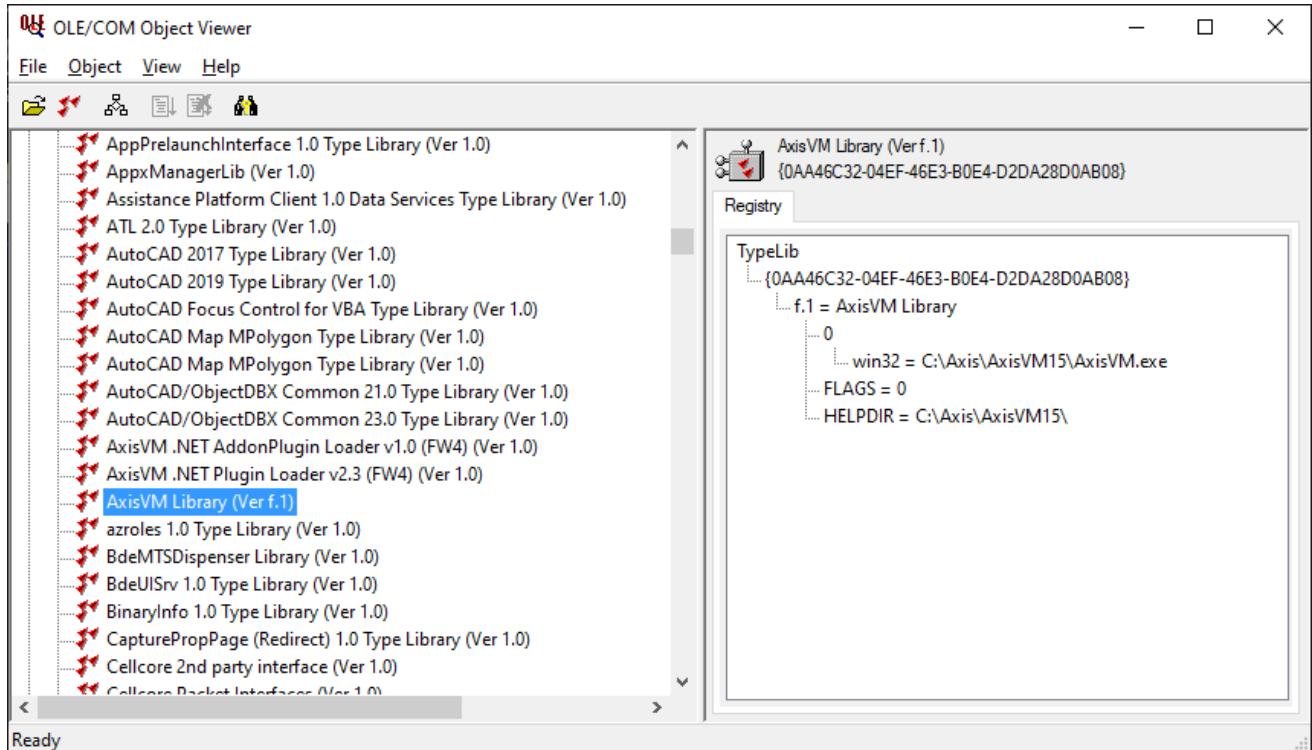
Determining the major and minor version of the COM server

The major and minor versions of the COM server are correlated to the version of AxisVM, for example AxisVM X5R2 corresponds to AxisVM COM server 15.1. A safe way to determine the COM version for an installed AxisVM is to check the properties of Interop.AxisVM.FW4.dll by right clicking to it, selecting Properties in the menu then switching to the Details tab. There the File version will contain this information. In the case of AxisVMX5R2 it will be 15.1.0.0.



The mere existence of an AxisVM directory doesn't necessarily mean that the COM server associated with that version of AxisVM is active. The registered COM servers can be checked with external programs, like

OleView.exe. In the picture it can be seen that only the 32 bit version of the COM server is installed, and only version 15.1 (shown here as f.1)



Unfortunately the availability of this program depends on the whim of Microsoft. It used to be obtainable for free, then eventually it was bundled into Visual Studio. As of 2019, it can be obtained freely by installing a trial version of Visual Studio. While Visual Studio will expire, this program can be used even after that, as long as Visual Studio remains installed.

An AxisVM COM server can be re-registered any time by running as an administrator !REGISTER_AXISVM.BAT for the 32 bit version and/or !REGISTER_AXISVM_X64.BAT for the 64 bit version. Do note that by default only the 64 bit version of AxisVM is installed, if you need the 32 version too it can be obtained by reinstalling AxisVM, and selecting the 32 bit version of the program. Doing this will not affect the already installed 64 bit version. You can easily see which AxisVM versions are installed : in the AxisVM folder AxisVM.exe is the 32 bit version, AxisVM_x64.exe is the 64 bit version of the program.

Sometimes the existence of several AxisVM COM servers causes conflicts in the clients. A clean state can be obtained by running !UnregAxisVMComServer.bat as administrator. This will remove all references from all AxisVM versions. After running !UnregAxisVMComServer.bat, you should reinstall the AxisVM whose COM server you want to use.

Visual C++

In C++ the AxisVM type library can be imported by inserting the #import directive into the cpp file. The syntax is #import „C:\Program Files\AxisVM\AxisVM.exe”, where C:\Program Files\AxisVM\AxisVM.exe stands for the application file name with full path. The #import directive converts content of the type library into C++ classes.

If Microsoft Visual Studio 2005 or later is used targeting .NET, the #import directive is not supported. The solution is to select the project in the Solution Explorer, choose Project / References from the main menu. Under Common Properties, select References. Click on the Add New Reference... button, click on the COM tab, and select AxisVM Library. Visual Studio will create a managed wrapper (Interop.AxisVM.dll) around the COM server.

AxisVM COM server uses SafeArrays with lower bound = 1 but the default marshalling for arrays assumes lower bound = 0. That can lead to SafeArrayRankMismatchException exceptions. To avoid them use TypeLibraryImporter. Enter this command at the Visual Studio Command Prompt using the appropriate filenames and paths in your system:

```
tlbimp.exe „C:\Program Files\AxisVM\AxisVM.exe” /out:“C:\Programming\Interop.AxisVM.dll”
/namespace:AxisVM /sysarray
```

Visual Basic

In Microsoft Visual Basic 2008 click Project / Add Reference, select COM and select the AxisVM Library from the list of available COM objects.

Visual Basic For Applications (MS Office)

Excel 95-2003

Click on Tools – Macro – Visual Basic Editor. Then click on Tools – References, find in the list and select: AxisVM Library. Press OK. Close Visual Basic window. If you don't see AxisVM Library in the list, see the Troubleshooting/Development/Excel section.

Excel 2010

Before selecting Type library as in previous version, you must enable the Developer tab if disabled: On the File tab, choose Options to open the Excel Options dialog box. Click Customize Ribbon on the left side of the dialog box. Under Choose commands from on the left side of the dialog box, select Popular Commands. Under Customize the ribbon on the right side of the dialog box, select Main tabs, and then select the Developer check box. Click on Developer ribbon and open Visual Basic Editor. Proceed the same way as in 95 – 2003 versions.

Borland Delphi

In Borland Delphi 5, 6, 7 click the Project / Import Type Library menu item. In Borland Developer Studio 2006 on use Component / ImportComponent. Select Import Type Library, click Next, and select the AxisVM Library from the list of available COM objects. Delphi then builds an AxisVM_TLB.pas file, which contains all declarations necessary to use the COM server.

Python

Run generate_module.py to generate AxisVM type library module. The 32 bit version of python interpreter can generate module for 32bit version of AxisVM and vice versa for 64 bit python and AxisVM. In the generate_module.py major_version and minor_version must be set according to the intended installed version of the AxisVM COM server (for example major_version = 15 and minor_version = 1). See [Determining the COM server version](#).

Required packages

- comtypes (minimum version 1.1.4, the older version has issues with return types)

Additional packages used in the example:

- numpy
- matplotlib

Import AxisVM module with

```
import comtypes.gen._0AA46C32_04EF_46E3_B0E4_D2DA28D0AB08_0_MajorVersion_MinorVersion as ax
```

where MajorVersion and MinorVersion must be replaced with the numbers used in the generate_module.py. For example, for a type library generated for AxisVM COM server 15.1 (i.e. AxisVM X5R2) the import line will look :

```
import comtypes.gen._0AA46C32_04EF_46E3_B0E4_D2DA28D0AB08_0_15_1 as ax
```

Example contains several files, start with main.py.

Examples

Programming examples can be downloaded from our website.

Example #1: Random lines (C++)

This plugin example is written in C++ (using Microsoft Visual Studio 2005). It creates 100 random lines in AxisVM.

The content of the `PluginSample01.h` header file shows the functions the DLL has to implement to be identified as an AxisVM plugin.

```
// !! do not use __declspec(dllexport) because linker will generate decorated export names and AxisVM  
needs undecorated names !!  
// !! use instead DEF file !! (Project Properties - Linker - Module Definition File)  
  
unsigned int GetPluginVersion(void);  
unsigned int GetMenuItemTextA(const void *Buffer,  
unsigned int BufferSize); unsigned int GetMenuItemTextW(const void *Buffer, unsigned int BufferSize);  
void SetAxisVMMainFormHandle(const unsigned int FormHandle);  
void OnMenuItemClick(void);
```

The content of the `PluginSample01.DEF` file is:

```
; created manually  
  
LIBRARY      "PluginSample01"  
  
EXPORTS  
  
GetPluginVersion      @1  
GetMenuItemTextA      @2  
GetMenuItemTextW      @3  
SetAxisVMMainFormHandle    @4  
OnMenuItemClick        @5
```

The entire `Plugin.cpp` is listed below.

```
// PluginSample01.cpp : Defines the exported functions for the DLL application.  
//  
  
#include "stdafx.h"  
#define _CRT_RAND_S  
#include "stdlib.h"  
#include "objbase.h"  
#include "PluginSample01.h"  
  
// import AxisVM COM Server's type library  
#import "C:\Programming\Munka\InterCAD\AxisVM9\axisvm.exe"  
  
HWND h MainForm = NULL;  
These are general functions of AxisVM plugins.  
  
unsigned int GetPluginVersion(void)  
{  
return 1;  
}  
  
unsigned int GetMenuItemTextA(const void *Buffer, unsigned int BufferSize)  
{  
strcpy_s((char *)Buffer, BufferSize, "Visual C Plugin Sample");  
return 22;  
}  
  
unsigned int GetMenuItemTextW(const void *Buffer, unsigned int BufferSize)  
{  
// BufferSize is in BYTE !  
wcscpy_s((wchar_t *)Buffer, BufferSize/2, L"Visual C Plugin Sample");  
return 22;  
}  
  
void SetAxisVMMainFormHandle(const unsigned int FormHandle)  
{  
h MainForm = (HWND)FormHandle;  
}
```

The following procedure will be called when the user clicks the *Visual C plugin sample* menu item.

```
void OnMenuItemClick(void)
{
BOOL fgNeedToUninitialize;
int n;
unsigned int number;
    AxisVM::IAxisVMMODELSptr      iAxisModels;
    AxisVM::IAxisVMMODELptr        iAxisModel;
    AxisVM::IAxisVMLINESptr       iAxisLines;
    AxisVM::RLineGeomData         rLineGeomData;
    AxisVM::RPoint3d              rP1, rP2;

fgNeedToUninitialize = CoInitializeEx(NULL, COINIT_APARTMENTTHREADED) == S_OK;
// create AxisVM COM server: always create 'AxisVMAplication' first!
AxisVM::IAxisVMAplicationPtr iAxisApp(__uuidof(AxisVM::AxisVMAplication));
// It was started as a plugin so no need to check the .Loaded property !!
// when this code is finished do not close AxisVM and no do not ask for closing AxisVM iAxisApp->AskCloseOnLastReleased = FALSE;
A new model is created in AxisVM, then 100 random lines are defined.

// create a new model
iAxisModels = iAxisApp->Models;
n = iAxisModels->New();
iAxisModel = iAxisModels->Item[n];

// get interfaces
iAxisLines = iAxisModel->Lines;
// add some lines (straight lines)
memset(&rLineGeomData, 0, sizeof(rLineGeomData));
for (n = 1; n <= 100; n++)
{
    rand_s(&number);
    rP1.x = (double) number / (double) UINT_MAX * 200.0 - 100.0;
    rand_s(&number);
    rP1.y = (double) number / (double) UINT_MAX * 200.0 - 100.0;
    rand_s(&number);
    rP1.z = (double) number / (double) UINT_MAX * 200.0 - 100.0;
    rand_s(&number);
    rP2.x = (double) number / (double) UINT_MAX * 200.0 - 100.0;
    rand_s(&number);
    rP2.y = (double) number / (double) UINT_MAX * 200.0 - 100.0;
    rand_s(&number);
    rP2.z = (double) number / (double) UINT_MAX * 200.0 - 100.0;
    iAxisLines->AddWithXYZ(rP1, rP2, AxisVM::lgtStraightLine, rLineGeomData);
}

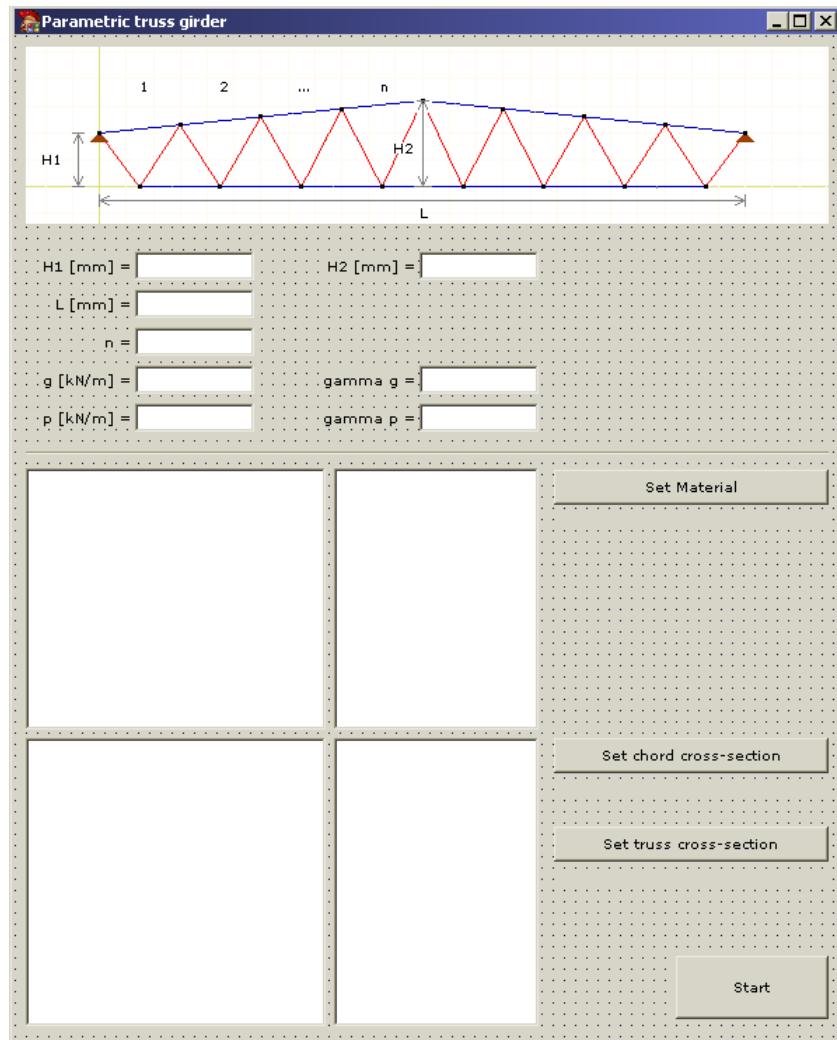
// show main form if it is hidden
if (!iAxisApp->Visible)
iAxisApp->Visible = TRUE;
// change to perspective view and fit view
iAxisModel->View = AxisVM::vPerspective;
iAxisModel->FitInView();

if (fgNeedToUninitialize)
CoUninitialize();
MessageBox(h MainForm, L"Done.", L"PluginSample01", MB_OK | MB_ICONINFORMATION);
}
```

Example #2: Truss model construction

This dialog is the user interface of a construction example. This Delphi program builds a parametric truss girder in AxisVM based on several parameters. A Visual C++ .NET source is also presented.

Download all source code examples from www.axisvm.com.



Delphi example

When the program starts and the dialog is displayed the following code is executed:

```
olecheck(CoCreateInstance(CLASS_AxisVMAApplication, nil, CLSCTX_ALL, IID_IAxisVMAApplication,  
    fAxApp));  
fAxApp.AskCloseOnLastReleased := True;  
// wait until fully loaded  
while fAxApp.Loaded = 1bFalse do  
    Sleep(100);
```

Where `fAxApp: AxisVMAApplication` is a private field of the form, `CLASS_AxisVMAApplication` and `IID_IAxisVMAApplication` is made available by including `AXISVM_TLB.pas` into the `uses` list of the interface section.

The `fAxApp.Catalog` object is used to get material and cross-section libraries. By selecting from the trees and clicking `SetMaterial`, `Set chord cross-section` and `Set truss cross-section` buttons the user can select the material and cross-sections to be used when building the model.

Upon clicking the Start button, a procedure (`btnStartClick`) will be executed:

The following local variables are defined within the procedure:

```
AxModels: AxisVMMODELS;
AxModel: AxisVMModel;
AxMaterials: AxisVMMaterials;
AxCrossSections: AxisVMCrossSections;
AxNodes: AxisVMNodes;
AxLines: AxisVMLines;
AxLine: AxisVMLine;
AxLoadCases: AxisVMLoadCases;
AxLoadComb: AxisVMLoadCombinations;
AxLoads: AxisVMLoads;
AxNodalSupports: AxisVMNodalSupports;
```

IMPORTANT NOTE:

Use variables instead of calling functions with full path. It can slow down COM server dramatically.

See [this](#) for more information

First a new model is created:

```
{create new model}
AxModels := fAxApp.Models;
n := AxModels.New;
AxModel := AxModels.Item[n];
```

Then materials and cross-sections are added to the list of model materials and cross-sections.

```
frmStatus.lblStatus.Caption := 'Adding material && cross-sections from catalog...';
frmStatus.Refresh;
{add material from catalog}
AxMaterials := AxModel.Materials;
MaterialId := AxMaterials.AddFromCatalog(fMaterialNDC, fMaterialName);

{add crosssections from catalog}
AxCrossSections := AxModel.CrossSections;
TrussCSId := AxCrossSections.AddFromCatalog(fTrussType, fTrussName);
ChordCSId := AxCrossSections.AddFromCatalog(fChordType, fChordName);
```

User fields are read into variables:

```
{get form data}
n := 2*StrToInt(Ledn.Text);
SetLength(UpperNodes, n+1);
SetLength(LowerNodes, n);
L := StrToFloat(LedL.Text) / 1000; // [m]
H1 := StrToFloat(LedH1.Text) / 1000; // [m]
H2 := StrToFloat(LedH2.Text) / 1000; // [m]
dx := L / n;
dz := (H2 - H1) / (n div 2);
p := StrToFloat(LedP.Text);
g := StrToFloat(LedG.Text);

{left support position}
StartX := 0;
StartZ := H1;
```

Nodes of the upper and lower chord are added to the Nodes object of the model.

```
frmStatus.lblStatus.Caption := 'Adding nodes...';
frmStatus.Refresh;

{adding nodes}
AxNodes := AxModel.Nodes;
for i := 0 to (n div 2) do
begin
  x := StartX + i*dx;
  z := StartZ + i*dz;
  UpperNodes[i] := AxNodes.Add(x, 0, z);
end;
for i := (n div 2)+1 to n do
begin
  x := StartX + i*dx;
  z := StartZ + (n div 2)*dz - (i - (n div 2))*dz;
  UpperNodes[i] := AxNodes.Add(x, 0, z);
end;
for i := 0 to n - 1 do
begin
  x := StartX + L / n / 2 + i*dx;
  z := StartZ - H1;
  LowerNodes[i] := AxNodes.Add(x, 0, z);
end;
```

Chords are made of beam elements:

```

frmStatus.lblStatus.Caption := 'Adding chords...';
frmStatus.Refresh;

{adding chords}
AxLines := AxModel.Lines;
FillChar(GeomData, Sizeof(GeomData), 0);
exc.x := 0;
exc.y := 0;
exc.z := 0;
for i := 0 to n-1 do
begin
  Lineid := AxLines.Add(UpperNodes[i], UpperNodes[i+1], lgtStraightLine, GeomData);
  AxLine := AxLines.Item[Lineid];
  AxLine.DefineAsBeam(MaterialId, ChordCSId, ChordCSId, exc, exc);
end;
for i := 0 to n-2 do
begin
  Lineid := AxLines.Add(LowerNodes[i], LowerNodes[i+1], lgtStraightLine, GeomData);
  AxLine := AxLines.Item[Lineid];
  AxLine.DefineAsBeam(MaterialId, ChordCSId, ChordCSId, exc, exc);
end;

```

Then trusses connecting the upper and lower chords are created:

```

frmStatus.lblStatus.Caption := 'Adding trusses...';
frmStatus.Refresh;

{adding trusses}
for i := 0 to n-1 do
begin
  Lineid := AxLines.Add(LowerNodes[i], UpperNodes[i], lgtStraightLine, GeomData);
  AxLine := AxLines.Item[Lineid];
  AxLine.DefineAsTruss(MaterialId, TrussCSId, lntensionAndCompression, 0);

  Lineid := AxLines.Add(LowerNodes[i], UpperNodes[i+1], lgtStraightLine, GeomData);
  AxLine := AxLines.Item[Lineid];
  AxLine.DefineAsTruss(MaterialId, TrussCSId, lntensionAndCompression, 0);
end;

```

Global nodal supports are placed at the first and last node of the upper chord.

```

{supports}
AxNodalSupports := AxModel.NodalSupports;
Stiffnesses.x := 1e10; Nonlinearity.x := lntensionAndCompression; Resistances.x := 0;
Stiffnesses.y := 1e10; Nonlinearity.y := lntensionAndCompression; Resistances.y := 0;
Stiffnesses.z := 1e10; Nonlinearity.z := lntensionAndCompression; Resistances.z := 0;
Stiffnesses.xx := 1e8; Nonlinearity.xx := lntensionAndCompression; Resistances.xx := 0;
Stiffnesses.yy := 1e8; Nonlinearity.yy := lntensionAndCompression; Resistances.yy := 0;
Stiffnesses.zz := 1e8; Nonlinearity.zz := lntensionAndCompression; Resistances.zz := 0;
AxNodalSupports.AddNodalGlobal(Stiffnesses, Nonlinearity, Resistances, UpperNodes[0]);
AxNodalSupports.AddNodalGlobal(Stiffnesses, Nonlinearity, Resistances, UpperNodes[n]);

```

```

frmStatus.lblStatus.Caption := 'Adding load cases...';
frmStatus.Refresh;

```

Two standard load cases named 'g' and 'p' are created.

```

{adding load cases}
AxLoadCases := AxModel.LoadCases;
lc_g := AxLoadCases.Add('g', lctStandard);
lc_p := AxLoadCases.Add('p', lctStandard);

```

Now the model contains three load cases: 'ST1' (which is automatically created for new models), 'g' and 'p'. Fields of the work record `LoadNodalForce` are filled and nodal loads are placed onto the structure.

First for load case 'g',

```

frmStatus.lblStatus.Caption := 'Adding loads...';
frmStatus.Refresh;

{adding loads}
AxLoads := AxModel.Loads;
LoadNodalForce.LoadCaseId := lc_g;
LoadNodalForce.Fx := 0;
LoadNodalForce.Fy := 0;
LoadNodalForce.Fz := - (g*L)/n / 2;
LoadNodalForce.Mx := 0;
LoadNodalForce.My := 0;
LoadNodalForce.Mz := 0;
LoadNodalForce.ReferenceId := 0; // automatic reference
LoadNodalForce.NodeId := UpperNodes[0];
AxLoads.AddNodalForce(LoadNodalForce);
LoadNodalForce.NodeId := UpperNodes[n];
AxLoads.AddNodalForce(LoadNodalForce);
LoadNodalForce.Fz := - (g*L)/n;
for i := 1 to n - 1 do
begin
  LoadNodalForce.NodeId := UpperNodes[i];
  AxLoads.AddNodalForce(LoadNodalForce);
end;
```

then for load case 'p'

```

LoadNodalForce.LoadCaseId := lc_p;
LoadNodalForce.Fx := 0;
LoadNodalForce.Fy := 0;
LoadNodalForce.Fz := - (p*L)/n / 2;
LoadNodalForce.Mx := 0;
LoadNodalForce.My := 0;
LoadNodalForce.Mz := 0;
LoadNodalForce.ReferenceId := 0; // automatic reference
LoadNodalForce.NodeId := UpperNodes[0];
AxLoads.AddNodalForce(LoadNodalForce);
LoadNodalForce.NodeId := UpperNodes[n];
AxLoads.AddNodalForce(LoadNodalForce);
LoadNodalForce.Fz := - (p*L)/n;
for i := 1 to n - 1 do
begin
  LoadNodalForce.NodeId := UpperNodes[i];
  AxLoads.AddNodalForce(LoadNodalForce);
end;
```

Four load combinations are created.

```

frmStatus.lblStatus.Caption := 'Adding load combinations...'; frmStatus.Refresh;
{adding load combinations}
AxLoadComb := AxModel.LoadCombinations;
SetLength(Factors, 3);
SetLength(LoadCaseIds, 3);

1*ST1+1.35*g+1.5*p

Factors[0] := 0;
LoadCaseIds[0] := 1; // ST1
Factors[1] := 1.35;
LoadCaseIds[1] := lc_g;
Factors[2] := 1.5;
LoadCaseIds[2] := lc_p;
DoubleArrayToSafeArray(Factors, saFactors);
IntArrayToSafeArray(LoadCaseIds, saLoadCaseIds);
combid := AxLoadComb.Add('1', ctOther, saFactors, saLoadCaseIds);
SafeArrayDestroy(saFactors);
SafeArrayDestroy(saLoadCaseIds);
if combid < 1 then
ShowMessage('Load combination error: '+IntToStr(combid));

1*ST1+1.35*g

Factors[0] := 0;
LoadCaseIds[0] := 1; // ST1
Factors[1] := 1.35;
LoadCaseIds[1] := lc_g;
Factors[2] := 0;
LoadCaseIds[2] := lc_p;
DoubleArrayToSafeArray(Factors, saFactors);
IntArrayToSafeArray(LoadCaseIds, saLoadCaseIds);
combid := AxLoadComb.Add('2', ctOther, saFactors, saLoadCaseIds);
SafeArrayDestroy(saFactors);
SafeArrayDestroy(saLoadCaseIds);
if combid < 1 then
ShowMessage('Load combination error: '+IntToStr(combid));
```

```
1*ST1+g+p
```

```
    Factors[0] := 0;
    LoadCaseIds[0] := 1; // ST1
    Factors[1] := 1.0;
    LoadCaseIds[1] := lc_g;
    Factors[2] := 1.0;
    LoadCaseIds[2] := lc_p;
    DoubleArrayToSafeArray(Factors, saFactors);
    IntArrayToSafeArray(LoadCaseIds, saLoadCaseIds);
    combid := AxLoadComb.Add('3', ctOther, saFactors, saLoadCaseIds);
    SafeArrayDestroy(saFactors);
    SafeArrayDestroy(saLoadCaseIds);
    if combid < 1 then
        ShowMessage('Load combination error: '+IntToStr(combid));
```

```
g
```

```
    Factors[1] := 1.0;
    LoadCaseIds[1] := lc_g;
    Factors[2] := 0.0;
    LoadCaseIds[2] := lc_p;
    DoubleArrayToSafeArray(Factors, saFactors);
    IntArrayToSafeArray(LoadCaseIds, saLoadCaseIds);
    combid := AxLoadComb.Add('4', ctOther, saFactors, saLoadCaseIds);
    SafeArrayDestroy(saFactors);
    SafeArrayDestroy(saLoadCaseIds);
    if combid < 1 then
        ShowMessage('Load combination error: '+IntToStr(combid));
```

```
remove default ST1 load case
```

```
AxLoadCases.Delete(1);
```

Deleting the default 'ST1' load case automatically updates load combinations.

Double and integer arrays are converted to safearrays by two routines of the `safeArrayutils.pas` unit:

```
procedure DoubleArrayToSafeArray(const aItems: ADouble; var ppItems: PSafeArray);
```

```
var
  i, n: Integer;
begin
  begin
    n := Length(aItems);
    ppItems := SafeArrayCreateVector(VT_R8, 1, n);
    for i := 1 to n do
      SafeArrayPutElement(ppItems, i, aItems[i-1]);
  end;
end;
```

```
procedure IntArrayToSafeArray(const aItems: AInteger; var ppItems: PSafeArray);
```

```
var
  i, n: Integer;
begin
  begin
    n := Length(aItems);
    ppItems := SafeArrayCreateVector(VT_I4, 1, n);
    for i := 1 to n do
      SafeArrayPutElement(ppItems, i, aItems[i-1]);
  end;
end;
```

`fAxApp.Visible := True` displays AxisVM to show the model.

If AxisVM main window is visible while executing commands screen updates make the application slower. So it is worth hiding AxisVM before running the code.

```
finally
  // show AxisVM
  fAxApp.Visible := True;
  // set view
  AxModel.View := vPerspective;
  AxModel.FitInView;
  btnStart.Enabled := True;
  frmStatus.Hide;
end;
end;
```

C++ .NET example

When the program starts and the dialog is displayed the following code is executed:

```
fAxApp = (gcnew AxisVMApplicationClass);
fAxApp->AskCloseOnLastReleased = ELongBoolean::lbTrue;
// wait until fully loaded
while (fAxApp->Loaded != ELongBoolean::lbTrue)
{
    System::Threading::Thread::Sleep(100);
    Refresh();
}
```

Where `AxisVMApplicationClass ^fAxApp` is a private field of the form. COM server definitions are made available by the line using namespace `AxisVM`; in the header file.

The `fAxApp->Catalog` object is used to get material and cross-section libraries. By selecting from the trees and clicking *SetMaterial*, *Set chord cross-section* and *Set truss cross-section* buttons the user can select the material and cross-sections to be used when building the model.

Upon clicking the Start button a procedure (`btnStart_Click`) will be executed.

The following local variables are defined within the procedure:

```
AxisVMMaterials ^AxMaterials;
AxisVMCrossSections ^AxCrossSections;
AxisVMNodes ^AxNodes;
AxisVMLines ^AxLines;
AxisVMLine ^AxLine;
AxisVMNodalSupports ^AxNodalSupports;
AxisVMLoadCases ^AxLoadCases;
AxisVMLoads ^AxLoads;
AxisVMLoadCombinations ^AxLoadComb;
```

IMPORTANT NOTE:

Use variables instead of calling functions with full path. It can slow down COM server dramatically.

See [this](#) for more information

First a new model is created:

```
{create new model}
AxModels = fAxApp->Models;
n = AxModels->New();
AxModel = AxModels->Item[n];
```

Then materials and cross-sections are added to the list of model materials and cross-sections.

```
// add material from catalog tsslblInfo->Text = L"Adding material from catalog..."; Refresh();
AxMaterials = AxModel->Materials;
MaterialId = AxMaterials->AddFromCatalog(fMaterialNDC, fMaterialName);
// add crosssections from catalog tsslblInfo->Text = L"Adding cross-sections from catalog...";
Refresh();
AxCrossSections = AxModel->CrossSections;
TrussCSId = AxCrossSections->AddFromCatalog(fTrussType, fTrussName);
ChordCSId = AxCrossSections->AddFromCatalog(fChordType, fChordName);
```

User fields are read into variables:

```
// get form data
tsslblInfo->Text = L"Getting form data..."; Refresh();
n = 2 * Convert::ToInt32(tbN->Text);
UpperNodes = gcnew array<int,1>(n+1);
LowerNodes = gcnew array<int,1>(n);
L = Convert::ToDouble(tbL->Text) / 1000; // [m]
H1 = Convert::ToDouble(tbH1->Text) / 1000; // [m]
H2 = Convert::ToDouble(tbH2->Text) / 1000; // [m]
dx = L / n;
dz = (H2 - H1) / (n / 2);
p = Convert::ToDouble(tbP->Text);
g = Convert::ToDouble(tbG->Text);

// left support position
StartX = 0;
StartZ = H1;
```

Nodes of the upper and lower chord are added to the Nodes object of the model.

```

// adding nodes
tsslblInfo->Text = L"Adding nodes..."; Refresh();
AxNodes = AxModel->Nodes;
for (i = 0; i <= (n/2); i++)
{
    x = StartX + i*dx;
    z = StartZ + i*dz;
    UpperNodes[i] = AxNodes->Add(x, 0, z);
}
for (i = (n/2)+1; i <= n; i++)
{
    x = StartX + i*dx;
    z = StartZ + (n/2)*dz - (i - (n/2))*dz;
    UpperNodes[i] = AxNodes->Add(x, 0, z);
}
for (i = 0; i < n; i++)
{
    x = StartX + L / n / 2 + i*dx;
    z = StartZ - H1;
    LowerNodes[i] = AxNodes->Add(x, 0, z);
}

```

Chords are made of beam elements:

```

// adding chords
tsslblInfo->Text = L"Adding chords..."; Refresh();
AxLines = AxModel->Lines;
exc.x = 0;
exc.y = 0;
exc.z = 0;
for (i = 0; i < n; i++)
{
    lineid = AxLines->Add(UpperNodes[i], UpperNodes[i+1], ELineGeomType::lgtStraightLine, GeomData);
    AxLine = AxLines->Item[lineid];
    AxLine->DefineAsBeam(MaterialId, ChordCSId, ChordCSId, exc, exc);
}
for (i = 0; i < (n-1); i++)
{
    lineid = AxLines->Add(LowerNodes[i], LowerNodes[i+1], ELineGeomType::lgtStraightLine, GeomData);
    AxLine = AxLines->Item[lineid];
    AxLine->DefineAsBeam(MaterialId, ChordCSId, ChordCSId, exc, exc);
}

```

Then trusses connecting the upper and lower chords are created:

```

// adding trusses
tsslblInfo->Text = L"Adding trusses..."; Refresh();
for (i = 0; i < n; i++)
{
    lineid = AxLines->Add(LowerNodes[i], UpperNodes[i], ELineGeomType::lgtStraightLine, GeomData);
    AxLine = AxLines->Item[lineid];
    AxLine->DefineAsTruss(MaterialId, TrussCSId, ELineNonLinearity::lnlTensionAndCompression, 0);
    lineid = AxLines->Add(LowerNodes[i], UpperNodes[i+1], ELineGeomType::lgtStraightLine, GeomData);
    AxLine = AxLines->Item[lineid];
    AxLine->DefineAsTruss(MaterialId, TrussCSId, ELineNonLinearity::lnlTensionAndCompression, 0);
}

```

Global nodal supports are placed at the first and last node of the upper chord.

```
// adding supports
tsslblInfo->Text = L"Adding supports..."; Refresh();
AxNodalSupports = AxModel->NodalSupports;
Stiffnesses.x = 1e10;
Nonlinearity.x = ELineNonLinearity::lnlTensionAndCompression; Resistances.x = 0;
Stiffnesses.y = 1e10;
Nonlinearity.y = ELineNonLinearity::lnlTensionAndCompression; Resistances.y = 0;
Stiffnesses.z = 1e10;
Nonlinearity.z = ELineNonLinearity::lnlTensionAndCompression; Resistances.z = 0;
Stiffnesses.xx = 1e8;
Nonlinearity.xx = ELineNonLinearity::lnlTensionAndCompression; Resistances.xx = 0;
Stiffnesses.yy = 1e8;
Nonlinearity.yy = ELineNonLinearity::lnlTensionAndCompression; Resistances.yy = 0;
Stiffnesses.zz = 1e8;
Nonlinearity.zz = ELineNonLinearity::lnlTensionAndCompression; Resistances.zz = 0;
AxNodalSupports->AddNodalGlobal(Stiffnesses, Nonlinearity, Resistances, UpperNodes[0]);
AxNodalSupports->AddNodalGlobal(Stiffnesses, Nonlinearity, Resistances, UpperNodes[n]);
```

Two standard load cases named 'g' and 'p' are created.

```
// adding load cases
tsslblInfo->Text = L"Adding load cases..."; Refresh();
AxLoadCases = AxModel->LoadCases;
lc_g = AxLoadCases->Add(L"g", ELoadCaseType::lctstandard);
lc_p = AxLoadCases->Add(L"p", ELoadCaseType::lctstandard);
```

Now the model contains three load cases: 'ST1' (which is automatically created for new models), 'g' and 'p'. Fields of the work record LoadNodalForce are filled and nodal loads are placed onto the structure.

First for load case 'g',

```
// adding loads
tsslblInfo->Text = L"Adding loads..."; Refresh();
AxLoads = AxModel->Loads;
LoadNodalForce.LoadCaseId = lc_g;
LoadNodalForce.Fx = 0;
LoadNodalForce.Fy = 0;
LoadNodalForce.Fz = - (g*L)/n / 2;
LoadNodalForce.Mx = 0;
LoadNodalForce.My = 0;
LoadNodalForce.Mz = 0;
LoadNodalForce.ReferenceId = 0; // automatic reference
LoadNodalForce.NodeId = UpperNodes[0];
AxLoads->AddNodalForce(LoadNodalForce);
LoadNodalForce.NodeId = UpperNodes[n];
AxLoads->AddNodalForce(LoadNodalForce);
LoadNodalForce.Fz = - (g*L)/n;
for (i = 1; i < n; i++)
{
    LoadNodalForce.NodeId = UpperNodes[i];
    AxLoads->AddNodalForce(LoadNodalForce);
}
```

then for load case 'p'

```
LoadNodalForce.LoadCaseId = lc_p;
LoadNodalForce.Fx = 0;
LoadNodalForce.Fy = 0;
LoadNodalForce.Fz = - (p*L)/n / 2;
LoadNodalForce.Mx = 0;
LoadNodalForce.My = 0;
LoadNodalForce.Mz = 0;
LoadNodalForce.ReferenceId = 0; // automatic reference
LoadNodalForce.NodeId = UpperNodes[0];
AxLoads->AddNodalForce(LoadNodalForce);
LoadNodalForce.NodeId = UpperNodes[n];
AxLoads->AddNodalForce(LoadNodalForce);
LoadNodalForce.Fz = - (p*L)/n;
for (i = 1; i < n; i++)
{
    LoadNodalForce.NodeId = UpperNodes[i];
    AxLoads->AddNodalForce(LoadNodalForce);
}
```

Four load combinations are created.

```
// adding load combinations tsslblInfo->Text = L"Adding load combinations..."; Refresh();
AxLoadComb = AxModel->LoadCombinations;
lengths[0] = 3;
lowerBounds[0] = 1;
Factors = Array::CreateInstance(double::typeid, lengths, lowerBounds);
LoadCaseIds = Array::CreateInstance(int::typeid, lengths, lowerBounds);
1*ST1+1.35*g+1.5*p
Factors->SetValue(0.0, 1);
LoadCaseIds->SetValue(1, 1); // ST1
Factors->SetValue(1.35, 2);
LoadCaseIds->SetValue(lc_g, 2);
Factors->SetValue(1.5, 3);
LoadCaseIds->SetValue(lc_p, 3);
combid = AxLoadComb->Add(L"1", ECombinationType::ctOther, Factors, LoadCaseIds);
if (combid < 1)
{
    MessageBox::Show(L"Load combination error: „ + Convert::ToString(combid));
}
1*ST1+1.35*g

Factors->SetValue(1.35, 2);
LoadCaseIds->SetValue(lc_g, 2);
Factors->SetValue(0.0, 3);
LoadCaseIds->SetValue(lc_p, 3);
combid = AxLoadComb->Add(L"2", ECombinationType:: ctOther, Factors, LoadCaseIds);
if (combid < 1)
{
    MessageBox::Show(L"Load combination error: „ + Convert::ToString(combid));
}
1*ST1+g+p

Factors->SetValue(1.0, 2);
LoadCaseIds->SetValue(lc_g, 2);
Factors->SetValue(1.0, 3);
LoadCaseIds->SetValue(lc_p, 3);
combid = AxLoadComb->Add(L"3", ECombinationType:: ctOther, Factors, LoadCaseIds);
if (combid < 1)
{
    MessageBox::Show(L"Load combination error: „ + Convert::ToString(combid));
}

Factors->SetValue(1.0, 2);
LoadCaseIds->SetValue(lc_g, 2);
Factors->SetValue(0.0, 3);
LoadCaseIds->SetValue(lc_p, 3);
combid = AxLoadComb->Add(L"4", ECombinationType:: ctOther, Factors, LoadCaseIds);
if (combid < 1)
{
    MessageBox::Show(L"Load combination error: „ + Convert::ToString(combid));
}
// remove default ST1 load case
AxLoadCases->Delete(1);
```

Deleting the default 'ST1' load case automatically updates load combinations.

fAxApp.Visible := True displays AxisVM to show the model.

If AxisVM main window is visible while executing commands screen updates make the application slower. So it is worth hiding AxisVM before running the code.

```
// show AxisVM
fAxApp->Visible = ELongBoolean::lbTrue;
// set view
AxModel->View = EView::vPerspective;
AxModel->FitInView();
```

Example #3: Line and load definition (Visual Basic)

This example is written in VisualBasic (using Microsoft VisualStudio 2008). It defines a line and place distributed loads on it. The entire vb file is listed. The main part is executed when the Start button is clicked.

```
Imports System.Runtime.InteropServices           'we need that for COM object releasing
Public Class frmMain
Private Sub btnStart_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles btnStart.Click
On Error GoTo ErrHandler
btnStart.Enabled = False
```

First the COM server is created and the AxisModel variable is set to the current AxisVM model.

```
' create AxisVM COM server: always create 'AxisVMAplication' first!
Dim AxisApp As New AxisVM.AxisVMAplication
' check for Loaded state
while AxisApp.Loaded = False
    ' let other threads do their work without 100% CPU usage
    System.Threading.Thread.Sleep(100)
End While
' when com client releases AxisVM COM Server's last object com server will terminate immediately
without asking to save current model
' if AskCloseOnLastReleased set to TRUE then AxisVM will ask the user whether to close AxisVM or not
AxisApp.AskCloseOnLastReleased = True
' current AxisVM supports only one model to be loaded -> get first model
Dim AxisModels = AxisApp.Models
Dim AxisModel = AxisModels.Item(1)
```

Two nodes are created

```
' //----- GEOMETRY BEGIN -----\\
' get the Nodes interface of current model
Dim AxisNodes = AxisModel.Nodes
' add two nodes
Dim nNodeId1 = AxisNodes.Add(1.0, 1.0, 1.0)
If nNodeId1 < 1 Then
    MsgBox("Error adding node #1. Error code: " + Str(nNodeId1), MsgBoxStyle.Critical, "Error")
End If
Dim nNodeId2 = AxisNodes.Add(10.0, 10.0, 1.0)
If nNodeId2 < 1 Then
    MsgBox("Error adding node #2. Error code: " + Str(nNodeId2), MsgBoxStyle.Critical, "Error")
End If
```

The two nodes are connected with a straight line.

```
Dim rLineGeomData As AxisVM.RLineGeomData
' get the Lines interface of current model
Dim AxisLines = AxisModel.Lines
' connect two nodes with a straight line (no need to fill the RLineGeomData because it has only
circle or ellipse related data)
Dim nLineId = AxisLines.Add(nNodeId1, nNodeId2, AxisVM.ELineGeomType.lgtStraightLine, rLineGeomData)
If nLineId < 1 Then
    MsgBox("Error adding line. Error code: " + Str(nLineId), MsgBoxStyle.Critical, "Error")
End If
' //----- GEOMETRY END -----\\
```

The line is defined as a beam element.

```

' //---- ELEMENTS BEGIN -----\\
' need crosssection(s) and material to define a line element (eg. beam)
' get materials interface of current model
Dim AxisMaterials = AxisModel.Materials
Dim nMaterial = AxisMaterials.AddFromCatalog(AxisVM.ENationalDesignCode.ndcEuroCode, "S 235")
If nMaterial < 1 Then
MsgBox("Error adding material. Error code: " + Str(nMaterial), MsgBoxStyle.Critical, "Error")
End If
' get crosssection interface of current model
Dim AxisCrossSections = AxisModel.CrossSections
Dim nCrossSection = AxisCrossSections.AddFromCatalog(AxisVM.ECrossSectionShape.cssI, "IPE 300")
If nCrossSection < 1 Then
MsgBox("Error adding crosssection. Error code: " + Str(nCrossSection), MsgBoxStyle.Critical, "Error")
End If
' get line interface of added line
Dim AxisLine = AxisLines.Item(nLineId)

' define this line as beam with added material and added crosssection
Dim rP As AxisVM.RPoint3d
rP.x = 0.0
rP.y = 0.0
rP.z = 0.0
Dim nResult = AxisLine.DefineAsBeam(nMaterial, nCrossSection, nCrossSection, rP, rP)
If nResult < 1 Then
MsgBox("Error adding beam. Error code: " + Str(nResult), MsgBoxStyle.Critical, "Error")
End If

```

Nodal supports are placed at both ends.

```

' get nodalsupports interface of current model
Dim AxisNodalSupports = AxisModel.NodalSupports

' add two supports
Dim rStiffnesses As AxisVM.RStiffnesses
rStiffnesses.x = 10000000000.0
rStiffnesses.y = 10000000000.0
rStiffnesses.z = 10000000000.0
rStiffnesses.xx = 10000000000.0
rStiffnesses.yy = 10000000000.0
rStiffnesses.zz = 10000000000.0
Dim rNonLinearity As AxisVM.RNonLinearity
rNonLinearity.x = AxisVM.ELineNonLinearity.lnlTensionAndCompression
rNonLinearity.y = AxisVM.ELineNonLinearity.lnlTensionAndCompression
rNonLinearity.z = AxisVM.ELineNonLinearity.lnlTensionAndCompression
rNonLinearity.xx = AxisVM.ELineNonLinearity.lnlTensionAndCompression
rNonLinearity.yy = AxisVM.ELineNonLinearity.lnlTensionAndCompression
rNonLinearity.zz = AxisVM.ELineNonLinearity.lnlTensionAndCompression
Dim rResistances As AxisVM.RResistances
rResistances.x = 0
rResistances.y = 0
rResistances.z = 0
rResistances.xx = 0
rResistances.yy = 0
rResistances.zz = 0
Dim nSupport1 = AxisNodalSupports.AddNodalGlobal(rStiffnesses, rNonLinearity, rResistances, nNodeId1)
If nSupport1 < 1 Then
MsgBox("Error adding nodal support #1. Error code: " + Str(nSupport1), MsgBoxStyle.Critical, "Error")
End If
Dim nSupport2 = AxisNodalSupports.AddNodalGlobal(rStiffnesses, rNonLinearity, rResistances, nNodeId2)
If nSupport2 < 1 Then
MsgBox("Error adding nodal support #2. Error code: " + Str(nSupport2), MsgBoxStyle.Critical, "Error")
End If
' //---- ELEMENTS END -----\\

```

Default ST1 load case is used to store concentrated and distributed beam loads.

First concentrated loads are applied.

```
' //----- LOADS BEGIN -----\\
' get the loads interface of current model
Dim AxisLoads = AxisModel.Loads

' add beam concentrated load to LoadCase #1 (AxisVM creates ST1 Load Case by default)
Dim rLoadBeamConc As AxisVM.RLoadBeamConcentrated
rLoadBeamConc.Fgx = 0.0
rLoadBeamConc.Fgy = 0.0
rLoadBeamConc.Fgz = 100.0
rLoadBeamConc.Mgx = 30.0
rLoadBeamConc.Mgy = 0.0
rLoadBeamConc.Mgz = 0.0
rLoadBeamConc.Position = 1.0 ' 1.0m from start node of the line/beam
rLoadBeamConc.SystemGLR = AxisVM.ESystem.sysGlobal
Dim nLoadConc = AxisLoads.AddBeamConcentrated(nLineId, 1, rLoadBeamConc)
If nLoadConc < 1 Then
MsgBox("Error adding beam concentrated load. Error code: " + Str(nLoadConc), MsgBoxStyle.Critical,
"Error")
End If
```

Then distributed loads are applied.

```
' add beam distributed load to LoadCase #1
Dim rLoadBeamDist As AxisVM.RLoadBeamDistributed
rLoadBeamDist.qx1 = 0.0
rLoadBeamDist.qy1 = 0.0
rLoadBeamDist.qz1 = -200.0
rLoadBeamDist.mx1 = 0.0
rLoadBeamDist.my1 = 0.0
rLoadBeamDist.mz1 = 0.0
rLoadBeamDist.qx2 = 0.0
rLoadBeamDist.qy2 = 0.0
rLoadBeamDist.qz2 = -500.0
rLoadBeamDist.mx2 = 0.0
rLoadBeamDist.my2 = 0.0
rLoadBeamDist.mz2 = 0.0
rLoadBeamDist.SystemGLR = AxisVM.ESystem.sysGlobal
rLoadBeamDist.Position1 = 2.0
rLoadBeamDist.Position2 = 6.0
rLoadBeamDist.DistributionType = AxisVM.EBeamRibDistributionType.brdtLength
rLoadBeamDist.Trapezoid = False
Dim nLoadDist = AxisLoads.AddBeamDistributed(nLineId, 1, rLoadBeamDist)
If nLoadDist < 1 Then
MsgBox("Error adding beam distributed load. Error code: " + Str(nLoadDist), MsgBoxStyle.Critical,
"Error")
End If
' //----- LOADS END -----\\
```

AxisVM dialog is made visible.

```
' com server application starts invisible -> make it visible
AxisApp.Visible = True
' fit view
AxisModel.View = AxisVM.EView.vPerspective
AxisModel.FitInView()

' sample code finished
GoTo Finish

ErrorHandler:
MsgBox(Err.Description, , "Error!")
Finish:
' Release COM objects VB6 way:
' AxisLoads = Nothing
' AxisNodalSupports = Nothing
' AxisLine = Nothing
' AxisCrossSections = Nothing
' AxisMaterials = Nothing
' AxisLines = Nothing
' AxisNodes = Nothing
' AxisModel = Nothing
' AxisModels = Nothing
' AxApp = Nothing

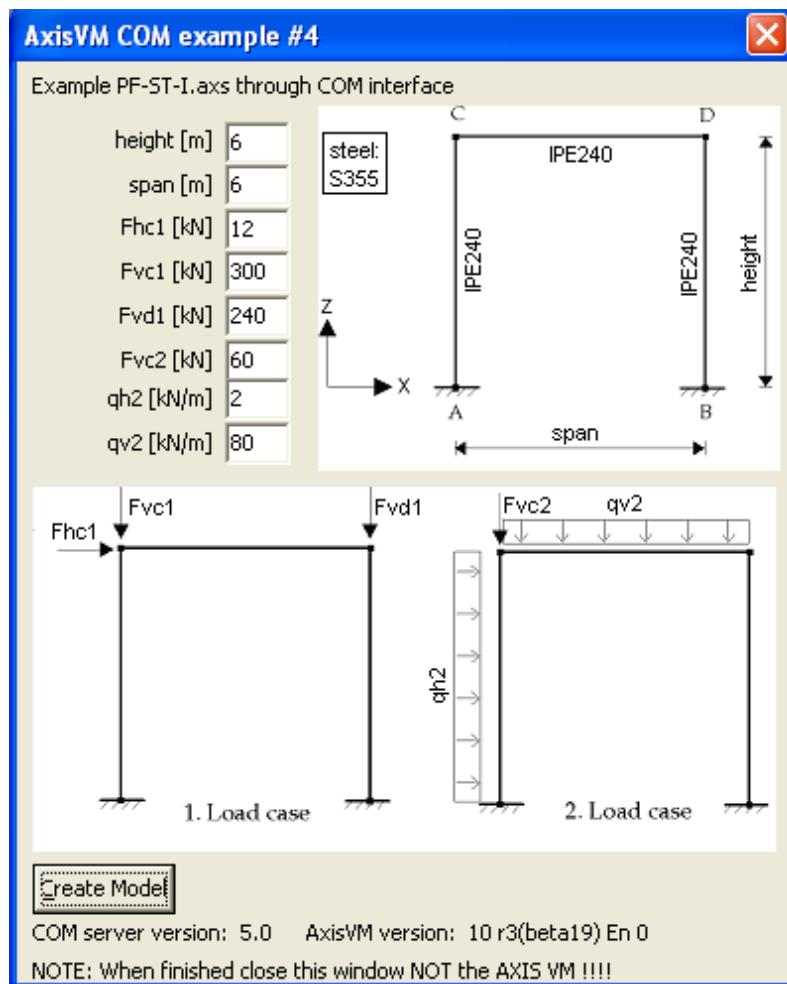
' Release COM objects VB.NET way:
Marshal.ReleaseComObject(AxisLoads)
Marshal.ReleaseComObject(AxisNodalSupports)
Marshal.ReleaseComObject(AxisLine)
Marshal.ReleaseComObject(AxisCrossSections)
Marshal.ReleaseComObject(AxisMaterials)
Marshal.ReleaseComObject(AxisLines)
Marshal.ReleaseComObject(AxisNodes)
Marshal.ReleaseComObject(AxisModel)
Marshal.ReleaseComObject(AxisModels)
Marshal.ReleaseComObject(AxApp)

btnStart.Enabled = True
End Sub
End Class
```

Example #4: Portal frame based on example PF-ST-I. axs (Delphi)

This example is based on model in file PF-ST-I. axs which can be downloaded from:

www.axisvm.com It's written in Delphi (using Borland Developer Studio 2006). It defines a steel portal frame and places point and distributed loads on it. The part, which creates the model and reads results is listed. The main part is executed when the "Create Model" button is clicked.



Check whether AxisVM model is already open

```
{create new model, current AxisVM supports only one model to be loaded -> get first model}
AxModels := fAxApp.Models;
AxModel := AxModels.Item[1];
if AxModel.Nodes.Count <> 0 then //checks if there any nodes in the model
begin// nodes found
  remodel:=MessageDlg('AxisVM model already opened, do you want ' +
  'to create a new one?',mtConfirmation,mbYesNo,0);
  case remodel of //re-model or skip
    idYes: //create new model
      begin
        m := AxModels.New;
        AxModel := AxModels.Item[m];
        fAxApp.Visible:=lbFalse;
      end;
    idNo: exit; //skip re-modeling
  end;
end
else
begin //nodes not found model will be created
  m := AxModels.New;
  AxModel := AxModels.Item[m];
end;
```

Then material and cross-section is defined.

```
{add material}
fMaterialNDC:=ndcEuroCode;
fMaterialName:='S 355'; //steel
AxModel.Settings.NationalDesignCode:=fMaterialNDC; //set national code

AxMaterials := AxModel.Materials;
MaterialId := AxMaterials.AddFromCatalog(fMaterialNDC, fMaterialName);

{add crosssection}
AxCrossSections := AxModel.CrossSections;
fSectName:='IPE 240' ; // cross section
fSectShape:=cssI; //cross-section shape: I section
SectCSId := AxCrossSections.AddFromCatalog(fSectShape, fSectName);
```

User fields are read into variables:

```
{read values from EditBoxes}
height := strtoint(heightEdt.Text); //in metres
span := strtoint(spanEdt.Text); //in metres
Fhc1:= strtoint(Fhc1Edt.Text); //in kN
Fvc1:= strtoint(Fvc1Edt.Text); //in kN
Fvd1:= strtoint(Fvd1Edt.Text); //in kN
Fvc2:= strtoint(Fvc2Edt.Text); //in kN
qh2:= strtoint(qh2Edt.Text); //in kN/m
qv2:= strtoint(qv2Edt.Text); //in kN/m
```

Nodes are added

```
{adding nodes}
n:=4; //number of nodes
SetLength(ANodes, n); // set size of the ANodes array

{first coordinate and left support position}
StartX := 0;
StartZ := 0;

AxNodes := AxModel.Nodes;
ANodes[0]:= AxNodes.AddWithDOF(StartX, 0, StartZ,dofFrameXZ );
ANodes[1]:= AxNodes.AddWithDOF(StartX, 0, height,dofFrameXZ);
ANodes[2]:= AxNodes.AddWithDOF(span, 0, height,dofFrameXZ);
ANodes[3]:= AxNodes.AddWithDOF(span, 0, StartZ,dofFrameXZ);
```

Created nodes are named according to scheme

```
NodeNames[1]:='A';
NodeNames[2]:='C';
NodeNames[3]:='D';
NodeNames[4]:='B';

for i := 1 to 4 do
begin
  AxNodes.Selected[ANodes[i-1]]:=lbTrue;
  m:=AxNodes SelCount;
  m:=AxNodes.RenameSelectedNodes(1,NodeNames[i]);
  AxNodes.Selected[ANodes[i-1]]:=lbFalse;
end;
```

Lines are created

```
{adding beams & columns}
linesTot:=n-1;
SetLength(ALines, n-1); // set size of the ALines array
AxLines := AxModel.Lines;
FillChar(GeomData, SizeOf(GeomData), 0); //initialize GeomData record
FillChar(exc, SizeOf(exc), 0); //initialize exc record = 0 eccentricity

for i := 0 to linesTot-1 do
begin
  LineId := AxLines.Add(ANodes[i], ANodes[i+1], lgtStraightLine, GeomData);
  AxLine := AxLines.Item[LineId];
  AxLine.DefineAsBeam(MaterialId, SectCSId, SectCSID, exc, exc); //material,start CS,end CS,eccentricity at begining and end
  ALines[i]:=LineId;
end;
```

Add nodal supports

```
{supports}
AxNodalSupports := AxModel.NodalSupports;
Stiffnesses.x := 1e10; Nonlinearity.x := lntensionAndCompression; Resistances.x := 0;
Stiffnesses.y := 1e10; Nonlinearity.y := lntensionAndCompression; Resistances.y := 0;
Stiffnesses.z := 1e10; Nonlinearity.z := lntensionAndCompression; Resistances.z := 0;
Stiffnesses.xx := 1e10; Nonlinearity.xx := lntensionAndCompression; Resistances.xx := 0;
Stiffnesses.yy := 1e10; Nonlinearity.yy := lntensionAndCompression; Resistances.yy := 0;
Stiffnesses.zz := 1e10; Nonlinearity.zz := lntensionAndCompression; Resistances.zz := 0;

AxNodalSupports.AddNodalGlobal(Stiffnesses,Nonlinearity,Resistances,ANodes[0]);//first node
AxNodalSupports.AddNodalGlobal(Stiffnesses,Nonlinearity,Resistances,ANodes[n-1]);//last node
```

Add load cases

```
{adding load cases}
AxLoadCases := AxModel.LoadCases;
lc1 := AxLoadCases.Add('1. Load case', lctStandard);
lc2 := AxLoadCases.Add('2. Load case', lctStandard);
```

Add nodal loads to loadcase with index lc1

```
{adding loads}
AxLoads := AxModel.Loads;
//loadcase lc1
fillchar(LoadNodalForce,sizeof(LoadNodalForce),0);
LoadNodalForce.LoadCaseId := lc1;
LoadNodalForce.Fx := Fhc1; // in kN
LoadNodalForce.Fz := -Fvc1;// in kN
LoadNodalForce.ReferenceId := 0; // automatic reference
LoadNodalForce.NodeId := ANodes[1];
AxLoads.AddNodalForce(LoadNodalForce);//add the nodal force
//null everything in LoadNodalForce record then set only non-zero values
fillchar(LoadNodalForce,sizeof(LoadNodalForce),0);
LoadNodalForce.LoadCaseId := lc1;
LoadNodalForce.Fz := -Fvd1;// in kN
LoadNodalForce.NodeId := ANodes[2];
AxLoads.AddNodalForce(LoadNodalForce);//add the nodal force
```

Add distributed loads to loadcase with index lc2

```
//loadcase lc2
// -80kn/m on beam 1
fillchar(LoadBeamDistributed,sizeof(LoadBeamDistributed),0);
LoadBeamDistributed.LoadCaseId:=lc2;
LoadBeamDistributed.LineId:=ALines[1];
LoadBeamDistributed.qz1:=-qv2;//in kn/m
LoadBeamDistributed.qz2:=LoadBeamDistributed.qz1;//same as beginning
LoadBeamDistributed.Position1:=0;
LoadBeamDistributed.Position2:=-1;//if (-), relative pos. (100%)
LoadBeamDistributed.DistributionType:=brdLength; //options length or projected
AxLoads.AddBeamDistributed(LoadBeamDistributed); //add the distr. load
// -2kN/m in beam index 0
fillchar(LoadBeamDistributed,sizeof(LoadBeamDistributed),0);
LoadBeamDistributed.LoadCaseId:=lc2;
LoadBeamDistributed.LineId:=ALines[0];
LoadBeamDistributed.qx1:=qh2;//at beginning of the line in kn/m GLOBAL direction
LoadBeamDistributed.qx2:=LoadBeamDistributed.qx1;//at the end of line (same as beginning)
LoadBeamDistributed.Position1:=0; // relative pos. (0%:=beginning)
LoadBeamDistributed.Position2:=-1;//if (-), relative pos. (100%:=end)
LoadBeamDistributed.DistributionType:=brdLength; //options length or projected
AxLoads.AddBeamDistributed(LoadBeamDistributed); //add the distr. load
```

Add nodal loads to loadcase with index lc2

```
// -60kN on node C
fillchar(LoadNodalForce,sizeof(LoadNodalForce),0);
LoadNodalForce.LoadCaseId := lc2;
LoadNodalForce.Fz := -Fvc2;// in kN
LoadNodalForce.NodeId := ANodes[1];
//add the nodal force
AxLoads.AddNodalForce(LoadNodalForce);
```

Add load combinations

```
{adding load combinations}
AxLoadComb := AxModel.LoadCombinations;
SetLength(Factors, 3); //number of factors incl. ST1
SetLength(LoadCaseIds, 3); //number of loadcases incl. ST1
//the default loadcase ST1 needs to be preserved at beginning will be deleted afterwards
Factors[0] := 0; // ST1 factor=0
LoadCaseIds[0] := 1; // ST1
Factors[1] := 1.35;
LoadCaseIds[1] := 1c1;
Factors[2] := 1.5;
LoadCaseIds[2] := 1c2;
DoubleArrayToSafeArray(Factors, saFactors);
IntArrayToSafeArray(LoadCaseIds, saLoadCaseIds);
combid := AxLoadComb.Add('1', ctOther, saFactors, saLoadCaseIds); // add 1 combo
SafeArrayDestroy(saFactors);
SafeArrayDestroy(saLoadCaseIds);
if combid < 1 then
    ShowMessage('Load combination error: '+IntToStr(combid));
```

Delete default loadcase

```
AxLoadCases.Delete(1); // remove default ST1 load case index of other load
//cases must be lowered by 1
dec(1c1);
dec(1c2);
```

Run calculation

```
calculationUserInteraction:=cuiUserInteraction;
AxModel.Calculation.LinearAnalysis(calculationUserInteraction);
Read calculated results
```

```
AxisVMResults:=AxModel.Results;
AxisVMDisplacements:=AxisVMResults.Displacements;
AxisVMForces:=AxisVMResults.Forces;
```

Read horizontal displacement at point C from loadcase 1

```
loadcaseID:=1c1;
nodeID:=ANodes[1];//point C
AxisVMDisplacements.LoadCaseId:= loadcaseID;
j:=AxisVMDisplacements.NodalDisplacementByLoadCaseId(nodeID,DisplacementValues,1cName);
ex_c1:=DisplacementValues.Ex*1000; //from metres to mm
```

Read moment My at point A from loadcase 1

```
LineID:=ALines[0];
SectionID:=1; //for beginning of line
AxisVMForces.LoadCaseId:= loadcaseID;
i:=AxisVMForces.LineForceByLoadCaseId(LineID,SectionID,LineForceValues,PosX,1cName);
My_a1:=LineForceValues.lfvMy;
```

Read horizontal displacement at point C from loadcase 2

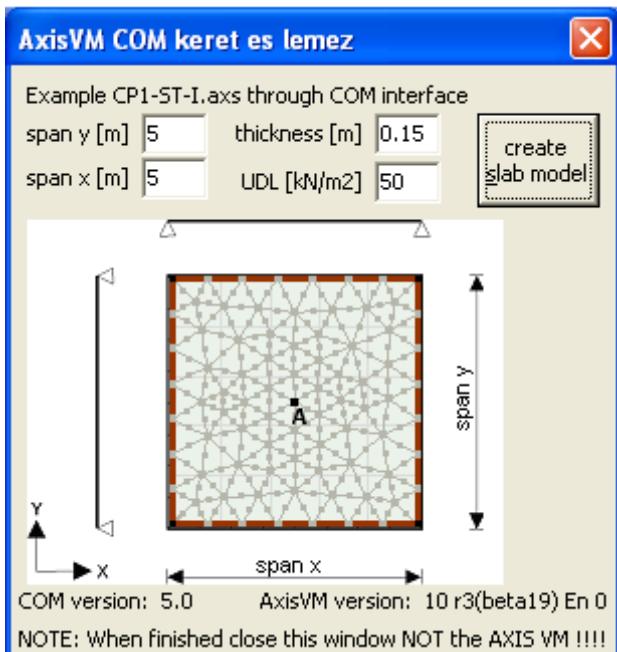
```
loadcaseID:=1c2;
nodeID:=ANodes[1];//point C
AxisVMDisplacements.LoadCaseId:= loadcaseID;
j:=AxisVMDisplacements.NodalDisplacementByLoadCaseId(nodeID,DisplacementValues,1cName);
ex_c2:=DisplacementValues.Ex*1000; //from metres to mm
```

Read moment My at point A from loadcase 2

```
LineID:=ALines[0];
SectionID:=1; //for beginning of line
PosX:=0;
AxisVMForces.LoadCaseId:= loadcaseID;
i:=AxisVMForces.LineForceByLoadCaseId(LineID,SectionID,LineForceValues,PosX,1cName);
My_a2:=LineForceValues.lfvMy;
```

Example #5: Concrete plate based on example CP-ST-I. axs (Delphi)

This example is based on model in file CP-ST-I. axs which can be downloaded from: www.axisvm.com. It has been written in Delphi (using Borland Developer Studio 2006). It defines a steel portal frame and places point and distributed loads on it. The part, which creates the model and reads results is listed. The main part is executed when the “Create Model” button is clicked.



Custom material is defined.

```
{add custom material}
AxMaterials := AxModel.Materials;
MaterialId:=AxMaterials.AddConcrete_EuroCode('EU concrete', 'My concrete', 'conc
E880',1,2,8.8e6,8.8e6,8.8e6,0,0,0,0,0,0,2500,28000,1.5,0.9,2.3 );
```

Add nodes

```
{adding nodes}
{first coordinate position}
StartX := 0;
StartY := 0;
//number of nodes
n:=4;
// set size of the ANodes array
SetLength(ANodes, n+1); //extra 1 for centre point

AxNodes := AxModel.Nodes;
//add nodes
ANodes[0]:= AxNodes.AddwithDOF(StartX, StartY, 0,dofPlateXY);
ANodes[2]:= AxNodes.AddwithDOF(StartX+spanx, StartY+spany, 0,dofPlateXY);
ANodes[1]:= AxNodes.AddwithDOF(StartX, StartY+spany, 0,dofPlateXY);
ANodes[3]:= AxNodes.AddwithDOF(StartX+spanx, 0, 0,dofPlateXY);
ANodes[4]:= AxNodes.AddwithDOF(StartX+spanx/2, StartY+spany/2, 0,dofPlateXY); //centre point
```

Add straight lines

```
{adding lines}
SetLength(AlineIDs, 4); // set size of the ALines array
AxLines := AxModel.Lines;
FillChar(GeomData, SizeOf(GeomData), 0);
for i := 0 to n-2 do
begin
  LineId := AxLines.Add(ANodes[i], ANodes[i+1], lgtStraightLine, GeomData);
  AxLine := AxLines.Item[LineId];
  AlineIDs[i]:=LineId;
end;
LineId := AxLines.Add(ANodes[3], ANodes[0], lgtStraightLine, GeomData);
AxLine := AxLines.Item[LineId];
AlineIDs[n-1]:=LineId; //last line closing polygon
```

Create domain

```
{adding domain}
AxDomains := AxModel.Domains;
fillchar(SurfaceAttr,sizeof(SurfaceAttr),0);
SurfaceAttr.Thickness:=thickness; //in m
SurfaceAttr.MaterialId:=MaterialId;
surfaceAttr.SurfaceType:=stPlate;// plate
IntArrayToSafeArray(AlineIDs,pSALineids); //from Alines to SafeArray LineIDs
domainID:=AxDomains.Add(pSALineids,surfaceAttr); // create domain from Lines
SafeArrayDestroy(psaLineIDs);
```

Define supports

```
{supports}
AxLineSupports := AxModel.LineSupports;
Stiffnesses.x := 1e10; Nonlinearity.x := lntTensionAndCompression; Resistances.x := 0;
Stiffnesses.y := 1e10; Nonlinearity.y := lntTensionAndCompression; Resistances.y := 0;
Stiffnesses.z := 1e10; Nonlinearity.z := lntTensionAndCompression; Resistances.z := 0;
Stiffnesses.xx := 0; Nonlinearity.xx := lntTensionAndCompression; Resistances.xx := 0;
Stiffnesses.yy := 0; Nonlinearity.yy := lntTensionAndCompression; Resistances.yy := 0;
Stiffnesses.zz := 0; Nonlinearity.zz := lntTensionAndCompression; Resistances.zz := 0;
for i := 0 to n-1 do
  begin //add edge supports to all defined lines
    AxLineSupports.AddEdgeGlobal(Stiffnesses,Nonlinearity,Resistances,AlineIDs[i],0,0,domainID,0);
  end;
```

Add loadcase

```
{adding load cases}
AxLoadCases := AxModel.LoadCases;
lc1 := AxLoadCases.Add('load case 1', lctStandard);
```

Add distributed load to domain

```
{adding loads}
AxLoads := AxModel.Loads;
fillchar(LoadDomainDistributed,sizeof(LoadDomainDistributed),0);
LoadDomainDistributed.LoadCaseId := lc1;
LoadDomainDistributed.DomainId:=domainID;
LoadDomainDistributed.qz := UDL; // in kN/m2
LoadDomainDistributed.systemGLR:=sysGlobal;
LoadDomainDistributed.DistributionType:=sddtProjected;
AxLoads.AddDomainDistributed(LoadDomainDistributed);
```

Generate mesh on the domain

```
{generating mesh}
fillchar(meshParameters,sizeof(meshParameters),0);
//mesh size=minimum of 3x thickness and 0.5m
meshParameters.MeshSize:=Math.min(3*thickness,0.5);
meshParameters.MeshType:=mtAdaptive;//can be also mtUniform
AxDomains.Selected[domainID]:=lbTrue;//select the domain for meshing
AxDomains.GenerateMeshOnSelectedDomains(meshParameters,errCodes,errNodes,errLines);
```

Delete default loadcase

```
AxLoadCases.Delete(1); // remove default ST1 load case index of other load
//cases must be lowered by 1
dec(lc1);
```

Run calculation

```
calculationUserInteraction:=cuiUserInteraction;
AxModel.Calculation.LinearAnalysis(calculationUserInteraction);
```

Read calculated results

```
AxisVMResults:=AxModel.Results;
AxisVMSurfaces:=AxModel.Surfaces;
AxisVMDisplacements:=AxisVMResults.Displacements;
AxisVMForces:=AxisVMResults.Forces;
//set load case index
AxisVMDisplacements.LoadCaseId:= lc1;
AxisVMForces.LoadCaseId:= lc1;
```

Read vertical displacement at point A

```
nodeID:=ANodes[4];//centre point
j:=AxisVMDisplacements.NodalDisplacementByLoadCaseId(nodeID,DisplacementValues,lcName);
ez:=DisplacementValues.EZ * 1000;
```

Read bending moment mx at point A

```
SurfaceID:=GetSurfaceIDbyNodeID(AxisVMSurfaces,nodeID); //set surface to the first found around node  
NO. 5  
AxisVMSurface:=AxisVMSurfaces.Item[SurfaceID];  
SurfaceVertexType:=svtContourPoint; //contour point  
SurfaceVertexID:=nodeID;  
AxisVMForces.SurfaceForceByLoadCaseId(SurfaceID, SurfacevertexType, SurfacevertexID, SurfaceForcevalues,  
lCName);  
mx_a:=SurfaceForcevalues.sfvMx;
```

Function GetSurfaceIDbyNodeID

```
function TMainForm.GetSurfaceIDbyNodeID(const AxisVMSurfaces:IAxisVMSurfaces; const NodeID:  
integer):integer;  
var  
surfaceID_i,Vertex_i,SurfVertexCount:integer;  
APoints:AIInteger;  
psaPoints:pSAFEARRAY;  
AxisVMSurface:IAxisVMSurface;  
  
begin  
for surfaceID_i := 1 to AxisVMSurfaces.Count do  
begin  
    AxisVMSurface:=AxisVMSurfaces.Item[surfaceID_i];  
    //SurfVertexCount can be 3 or 4 depending on shape of surface element  
    SurfVertexCount:=AxisVMSurface.GetContourPoints(psaPoints); //contour points of surface to  
    safe array  
    if SurfVertexCount>0 then  
        SafeArrayToIntArray(3,psaPoints,APoints)//from SafeArray LineIDs  
        //to Alines array size 3 for triangle element  
    else  
        begin//error while reading  
            result:=-1;  
            SafeArrayDestroy(psaPoints);  
            exit;  
        end;  
    for Vertex_i := 0 to SurfVertexCount-1 do  
    if APoints[Vertex_i]=NodeID then  
        begin  
            result:=surfaceID_i;  
            SafeArrayDestroy(psaPoints);  
            exit;  
        end;  
    end;  
end;
```

Example #6: Steel frame (Python)

This example is written for Python version 3.6. The python COM client builds a steel frame model in AxisVM, runs analysis and reads deflection and moments about major axis My.

Generate the interop file, see [Importing type library - Python](#). The generated interop file should be imported to the COM client

```
try:  
    import comtypes.gen._0AA46C32_04EF_46E3_B0E4_D2DA28D0AB08_0_9_3 as ax  
except:  
    print("Regenerate AxisVM module in generate_module.py !")  
    sys.exit()
```

The COM server is created with function cc.CreateObject(). The function returns an object of AxisVM COM server.

```
try:  
    axApp=comtypes.client.CreateObject(AX_APP_PROGID,comtypes.CLSCTX_ALL,None,ax.IAxisVMAplication)  
except:  
    sys.exit()
```