X7 API

COM interface

(AxisVM Library **17.2**)

Rev. 1
Reference Guide

Updated: 2023.06.21

CONTENTS

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 of the operating system. Registration makes AxisVM type library available for use. One COM server is registered automatically after the installation of the AxisVM (by default it will be the 64 bit version, unless the user explicitly installs the 32 bit version).

Depending on type of selected installation, the 64 bit or the 32 bit version of the AxisVM will be registered. Registering both the 32-bit and 64-bit versions is valid, and in fact required if both 32 and 64 bit applications has to use the COM server.

Older versions of AxisVM must be unregistered from the windows registry before new registration ! If in doubt, you can always run !UnregAxisVMComServer.bat as an administrator, which will delete all AxisVM COM server related entries. After that you will have to install the AxisVM COM server(s) that you want. An easy way to do this is (re)installing the AxisVM that you use.

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

Files refered below are in the same folder where AxisVM.exe is located. Both for registering and unregistering you must run the .bat as administrator.

Manuall unregistering of the type libraries (only for the version installed from that folder):

!UNREGISTER_AXISVM_X64.BAT for 64-bit version of AxisVM

!UNREGISTER_AXISVM.BAT for 32-bit version of AxisVM

IAxisVMDotNetPluginInterface_v2_3

!UNREGISTER_DOT_NET_PLUGIN_SERVER_v2.3.BAT unregisters both the 32 and 64-bit versions

IAxisVMDotNetAddonPluginInterface

!UNREGISTER_DOT_NET_ADDONPLUGIN_SERVER_v1.0.BAT unregisters both the 32 and 64-bit versions

Manual registering of the type libraries (only for the version from that folder):

Run:

!REGISTER_AXISVM_X64.BAT to register the 64-bit version

!REGISTER_AXISVM.BAT to register the 32-bit version

IAxisVMDotNetAddonPluginInterface(.NET AddonPlugins).

!REGISTER_DOT_NET_ADDONPLUGIN_SERVER_v1.0.BAT .NET addons (both the 32 and 64 bit versions)

IAxisVMDotNetPluginInterface_v2_3 (.NET Plugins).

!REGISTER_DOT_NET_PLUGIN_SERVER_v2.3.BAT .NET plugins (both the 32 and 64 bit versions)

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

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 succesfull, 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 I*AxisVMApplication* 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 *AxisVMApplication* object
(2) Handling the **Loaded** event of the *AxisVMApplicationEvents* 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 *AxisVMApplication*.**Loaded** property should be checked first even if the second method is used. If the server connects to a running application, the *AxisVMApplicationEvents***.Loaded** event will never be called.

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 [°C]
time	second [s]
degree	radian [rad]
force	kilo newton [kN]
frequency	hertz [Hz]

Other units have been derived from these.

Data types

enum	**ELongBoolean** { **lbFalse** = 0, **lbTrue** = 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
errIndexOutOfBounds = –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. Can be returned also if mtMinMax is not allowed (in which case you must specify either mtMin or mtMax).
errInvalidName = -114,	Name already exists or empty
errCombinationTypeNotAllowed = -115,	Combination type not allowed for current national design code
errInvalidEnvelopeUID = -116,	Check valid EnvelopeUID in IAxisVMEnvelopes 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
errObsolete = –123,	This function has become obsolete, it will no longer perform it's function. Check it's description, there is probably a newer function replacing it.

}

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 IAxisVMModel 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 IAxisVMApplication 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}
Dialog buttons

enum **EMessageDialogType** { **mdtWarning** = 0, **mdtError** = 1, **mdtInformation** = 2, **mdtConfirmation** = 3, **mdtCustom** = 4, **mdtErrorConfirmation** = 5 }
Dialog types

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

Dialog types

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

Test if client application is responding

enum **EMainFormTab**= {

mftGeometry = 0x0,	*Geometry tab in AxisVM*
mftElements = 0x1,	*Elements tab in AxisVM*
mftLoads = 0x2,	*Loads tab in AxisVM*
mftMesh = 0x3,	*Mesh tab in AxisVM*
mftStatic = 0x4,	*Static tab in AxisVM*
mftVibration = 0x5,	*Vibration tab in AxisVM*
mftDynamic= 0x6,	*Dynamic tab in AxisVM*
mftBuckling = 0x7,	*Buckling tab in AxisVM*
mftRCDesign = 0x8,	*Design tab in AxisVM*
mftSteelDesign = 0x9,	*Steel design tab in AxisVM*
mftTimberDesign = 0x10}	*Timber design tab in AxisVM*

Error codes

enum **EApplicationError** = {

appeUnknownUnitSystem = –100001	*Unknown Unit System*
appeRCBeamDesignObjectAlreadyCreated = - 100002	*RC beam design object is already created, only one instance of this object is allowed to run at a time*

}

Error during setting unit system.

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)*

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

EMessageDialogButton* **MessageDlg_vb** (Visual Basic compatible function of **MessageDlg**)

long **UnLoadCOMclients**

Stops, unloads and releases memory taken by COM clients (addons, plugins and addon-plugins). Returns 1 if successful.

long **Quit**

Quit from AxisVM. Stops, unloads and releases memory taken by COM clients (addons, plugins and addonplugins) then AxisVM quits.

Properties

EApplicationClose **ApplicationClose •**
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

ELongBoolean **AskCloseOnLastReleased •**
Determines whether the COM server shutdown displays a query to close the AxisVM application as well. Read and write property

ELongBoolean **AskSaveOnLastReleased•**
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.

EAxisVMPlatform **AxisVMPlatform**
Get AxisVM version type (32/64bit)

AxisVMCatalog* **Catalog**
Get AxisVM catalog interface for standard materials and cross-sections.

EClientAliveTest **ClientAliveTest**
Get or set what happens in AxisVM if client application is not responding

long **ClientAliveTestIntervalSec**
Get or set interval (seconds) of how often AxisVM checks if client application is responding or not

ELongBoolean **CloseOnLastReleased •**
If AskCloseOnLastReleased = False this property determines if AxisVM is closed after completing COM server shutdown. Read and write property

AxisVMCrossSectionEditor **CrossSectionEditor**
Get an interface for editing cross-sections.

ELongBoolean **COMclientsLoaded**
Returns lbTrue if AxisVM has loaded any COM clients (addons, plugins or addon-plugins). Read only property

BSTR **FullExePath**
Get axisvm.exe path

long **LibraryMajorVersion**
Get major version number of the COM-server.

long **LibraryMinorVersion**
Get minor version number of the COM-server.

ELongBoolean **Loaded**
Shows if AxisVM has been loaded or not. Read only property.

EWindowState **MainFormWindowState •**
Get or set window state of the AxisVM.

RWindowPosition **MainFormWindowPosition •**
Get or set window position of the AxisVM.

long	**MainFormWindowLeft** • *Get or set left distance of the main window of AxisVM.*
long	**MainFormWindowTop** • *Get or set top distance of the main window of AxisVM.*
long	**MainFormWindowWidth** • *Get or set width of the main window of AxisVM.*
long	**MainFormWindowHeight** • *Get or set height of the main window of AxisVM.*
EMainFormTab	**MainFormTab** • *Get or set which main tab in AxisVM should be activated.*
long	**ModalLevel** *Get number of opened modal forms in AxisVM. If zero, then no modal forms are open.*
AxisVMModels*	**Models** *Gets an interface containing the opened models. The number of models is limited to one.*
AxisVMObjectCreator	**ObjectCreator** *Get an interface to ObjectCreator*
BSTR	**Version** *Get AxisVM program version string. Not the same as the COM-server version.*
ELongBoolean	**Visible** • *Determines if the main form of the AxisVM application is visible or not. Hiding the main window can considerably speed up certain operations. If AxisVM has been launched by, the COM-server Visible remains False by default. Read and write property*

IAxisVMCatalog

AxisVM catalog interface for materials and cross-sections.

Enumerated types

enum **ENationalDesignCode** = {

ndcOther = $00000000	*Design code not specified*
ndcHungarian_MSZ = $00000001	*Hungarian design code MSZ*
ndcEuroCode = $00000002	*Eurocode design code*
ndcRomanian_STAS = $00000004	*Romanian design code STAS*
ndcDutch_NEN = $00000005	*Dutch design code NEN*
ndcGerman_DIN1045_1 = $00000006	*German design code DIN 1045-1*
ndcSwiss_SIA26x = $00000007	*Swiss design codes SIA26x*
ndcEuroCode_GER = $00000008	*Eurocode with German national annex*
ndcItalian = $00000009	*Eurocode with Italian national annex*
ndcEuroCode_Austrian = $0000000A	*Eurocode with Austrian national annex*
ndcEuroCode_UK = $0000000B	*Eurocode with UK national annex*
ndcEuroCode_NL = $0000000C	*Eurocode with Netherland national annex*
ndcEuroCode_FIN = $0000000D	*Eurocode with Finish national annex*
ndcEuroCode_RO = $0000000E	*Eurocode with Romanian national annex*
ndcEuroCode_HU = $0000000F	*Eurocode with Hungarian national annex*
ndcEuroCode_CZ = $00000010	*Eurocode with Czech national annex*
ndcEuroCode_B = $00000011	*Eurocode with Belgian national annex*
ndcEuroCode_PL = $00000012	*Eurocode with Polish national annex*
ndcEuroCode_DK = $00000013	*not implemented(Eurocode,Danish national annex)*
ndcEuroCode_S = $00000014	*not implemented(Eurocode, Swedish national annex)*
ndcUS = $00000015	*US not implemented yet*
ndcCA_NBCC = $00000016	*CA_NBCC not implemented yet*
ndcCA_Ontario = $00000017	*CA_ONTARIO not implemented yet*
ndcCA_Bridge = $00000018	*CA_Bridge not implemented yet*
ndcEuroCode_SK = $00000019	*Eurocode with Slovak national annex*
ndcEuroCode_LV = $0000001A	*Eurocode with Latvian national annex*
ndcEuroCode_NO = $0000001B	*Eurocode with Norwegian national annex*
ndcEuroCode_GR = $0000001C	*Eurocode with Greek national annex*
ndcEuroCode_NP = $0000001D	*Eurocode with Portuguese national annex*
ndcEuroCode_UNE = $0000001E	*Eurocode with Spanish national annex*

}

enum **ECrossSectionRegion** = *{*

cssr_Unknown = $00000000
cssr_EU = $00000001
cssr_HU = $00000002
cssr_RO = $00000003
cssr_US = $00000004
cssr_NL = $00000005
cssr_CN= $00000006
cssr_SK= $00000007
cssr_CA= $00000008
cssr_SE= $00000009
cssr_BR= $0000000A
cssr_PL= $0000000B

cssr_RU= $0000000C
cssr_CH= $0000000D
cssr_CZ= $0000000E
cssr_DE= $0000000F
cssr_FR= $00000010
cssr_ES= $00000011
cssr_IT= $00000012
cssr_AR= $00000013
cssr_AT= $00000014
cssr_BE= $00000015
cssr_BG= $00000016
cssr_DK= $00000017
cssr_EE= $00000018
cs**sr_FI**= $00000019
cssr_GR= $0000001A
cssr_IN = $0000001B
cssr_JP = $0000001C
cssr_LV = $0000001D
cssr_LT = $0000001E
cssr_NO = $0000001F
cssr_RS= $00000020
cssr_SI= $00000021
cssr_ZA= $00000022
cssr_TR= $00000023
cssr_UK= $00000024
cssr_AU= $00000025
cssr_NZ= $00000026
cssr_PT= $00000027
cssr_HR= $00000028
cssr_KR= $00000029
cssr_IL= $0000002A
}

Records / structures

RTableCrossSectionID = (

long	**TableID**	*table identifier*
long	**CrossSectionID**	*cross section identifier (in the table)*

)

Warning : TableID and CrossSectionID are valid only for as long as the table structure or the cross section structure inside a table is unchanged. Adding/deleting cross section tables, adding/deleting cross sections from a table will render this IDs corrupted.

RCrossSectionTable = (

ECrossSectionShapeEx	**CrossSectionShape**	*type of the cross section shape*
long	**ProviderID**	*1 if the table was provided by InterCAD*
ECrossSectionRegion	**CrossSectionRegion**	*region for the table*
long	**Id**	*table identifier to be used in GetTableCrossSections*

)

Warning : Id is valid only for as long as the table structure is unchanged. Adding/deleting cross section tables will render this Ids corrupted.

Functions

long **GetTableCrossSections** ([in] long **TableId,** [out] IAxisVMCrossSectionTable ** **CrossSectionTabe**)

TableId *Table ID. It can be picked from the returned data from GetAllTables, it is in the Id field*

CrossSectionTable *an interface for the cross section table. Its Item property can be used in GetCrossSection_V154, to get the parameters of that specific cross section.*

Queries all cross section names from the table with TableId. The Item property of CrossSectionTable can be used in GetCrossSection_V154 to access the full cross section information. If successful, result is the number of cross-sections in the table.Warning : GetAllTables must be called at some point prior to this function. Possible error codes: errIndexOutOfBounds (if a TableID is out of bounds), errInternalException, errCOMServerInternalError.

---

long **GetAllTables** ([out] IAxisVMCrossSectionTables ** **CrossSectionTables**)

CrossSectionTables *interface containing the list of all cross section tables. It is the starting point for the use of GetTableCrossSections.*

Queries all cross section tables. The caller later can filter it further, for example based on the CrossSectionShape field of the IAxisVMCrossSectionTables.Item.property. If successful, result is the number of tables. Possible error codes: errInternalException, errCOMServerInternalError.

---

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**)

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 **GetCrossSectionEx** ([in] ECrossSectionShapeEx **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 **GetCrossSectionNamesEx** ([in] ECrossSectionShapeEx **CrossSectionShape,** [in] BSTR **TableName,** [out] SAFEARRAY(BSTR)* **Names**)

CrossSectionShape *cross-section shape*

TableName *name of the cross-section table*

Names *list of cross-section names*

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

---

long **GetCrossSectionTableNames** ([in] ECrossSectionShape **CrossSectionShape,**
[out] SAFEARRAY(BSTR)* **TableNames**)

CrossSectionShape *cross-section shape*

TableNames *list of cross-section table names*

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

---

long **GetCrossSectionTableNamesEx** ([in] ECrossSectionShapeEx **CrossSectionShape,**
[out] SAFEARRAY(BSTR)* **TableNames**)

CrossSectionShape *cross-section shape*

TableNames *list of cross-section table names*

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

---

long **GetCrossSection_V154** ([in] RTableCrossSectionID **CrossSectionNameId,**
[out] AxisVMCrossSection* **CrossSection**)

CrossSectionNameId *a record corresponding to a cross-section from IAxisVMCrossSectionTable, for which the AxisVMCrossSection will be returned*

CrossSection *the returned detailed cross section information*

Queries the detailed cross section information for a given CrossSectionNameId. If successful, result is the value of CrossSectionID field from CrossSectionNameId. Warning : GetAllTables must be called at some point prior to this function. Possible error codes : errIndexOutOfBounds (if TableID or CrossSectionID is out of bounds), errInternalException.

---

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

DesignCode *national design code of the material*

Name *material name*

Material *an instance of the material object*

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

---

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

DesignCode *national design code of the material*

Names *list of material names*

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)

IAxisVMCrossSectionTables

Support interface for querying cross-sections tables.

Properties

long	**Count** *Number of tables*
RCrossSectionTable	**Item [**long **Index]** *Non string properties of a table. Its Id field is used for further queries.*
BSTR	**PrgTableName [**long **Index]** *Name of a table in the language of the user interface*
BSTR	**DocTableName [**long **Index]** *Name of a table in the language of the documentation*
BSTR	**FileName [**long **Index]** *Name of the file, where this table is stored (without path)*
BSTR	**Manufacturer [**long **Index]** *Name of the manufacturer, if available*

IAxisVMCrossSectionTable

Support interface for querying the cross-sections in a table.

Properties

long	**Count** *Number of cross-sections in the table*
RTableCrossSectionID	**Item [**long **Index]** *Descriptor record for a cross-section in a table. It is the input parameter for GetCrossSection_V154.*
long	**TableID** *The Tableid, through which this interface was obtained*
BSTR	**DocTableName** *Name of a table in the language of the documentation*
BSTR	**CrossSectionName [**long **Index]** *Name of the cross-section*

IAxisVMCrossSectionEditor

AxisVM interface for editing cross-sections.

Error codes

enum **ECrossSectionEditorError {**

cseeEditorNotOpened = –100001,	*Editor dialog window not opened*
cseeEditorModeIsNotSolidCrossSection = –100002 }	*Editor is not in solid cross-section mode*

Records / structures

RCrossSectionUserParams = (

long	**lParam**	*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(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	*User parameters*

Returns 1 if successful, otherwise returns an error code (errIndexOutOfBounds, cseeEditorModeIsNotSolidCrossSection, 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 **lParam,** [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*
lParam	*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 (errIndexOutOfBounds, cseeEditorModeIsNotSolidCrossSection, 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 (errIndexOutOfBounds, cseeEditorModeIsNotSolidCrossSection, 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.

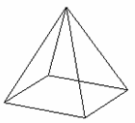
IAxisVMModel

The opened AxisVM Model

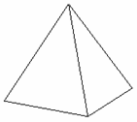
Enumerated types

enum **EDisplay** = {

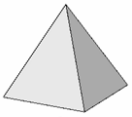
dWireframe = 0x0, *Wireframe*



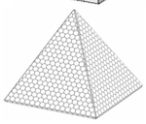
dHidden = 0x1, *wireframe with hidden lines removed*



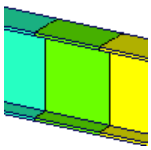
dRendered = 0x2, *rendered view*



dTextured = 0x3, *rendered view with textures*



dSolidResult = 0x4} *results on solid model (not always applicable)*



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,	*IFC 2.0*
ifc2x = 0x1,	*IFC 2x*
ifc2x2 = 0x2,	*IFC 2x2*
ifc2x3 = 0x3 }	*IFC 2x3*

IFC versions

enum **ELanguage** = {

lngEnglish = 0x00,	*English*
lngHungarian = 0x01,	*Hungarian*
lngGerman = 0x02,	*German*
lngRomanian = 0x03,	*Romanian*
lngSpanish = 0x04,	*Spanish*
lngItalian = 0x05,	*Italian*
lngRussian = 0x06,	*Not used*
lngPortuguese = 0x07,	*Portuguese*
lngSerbian = 0x08,	*Serbian*
lngDutch = 0x09,	*Dutch*
lngFrench = 0x0A,	*French*
lngHebrew = 0x0B,	*Not used*
lngArabic = 0x0C,	*Not used*
lngCzech = 0x0D,	*Czech*
lngSlovakian = 0x0E,	*Slovakian*
lngBrasilianPortuguese = 0x0F,	*Brazilian Portuguese*
lngGreek = 0x0F,	*Greek*
lngCroatian = 0x10,	*Croatian*
lngPolish = 0x11	*Polish*
lngBulgarian = 0x12 }	*Bulgarian*

Program or report languages

enum **ELengthUnit** = {

lu_mm = 0x0,	*mm*
lu_cm = 0x1,	*cm*
lu_dm = 0x2,	*dm*
lu_m = 0x3,	*m*
lu_inch = 0x4,	*inch*
lu_foot = 0x5,	*foot*
lu_yard = 0x6 }	*yard*

Length units

enum **ESelectionType** = {

seltNode = 0x01,	*nodes*
seltMidPoint = 0x02,	*line midpoints*
seltAllLines = 0x03,	*all line elements(beams, ribs, trusses) and lines*
seltTruss = 0x04,	*trusses*
seltBeam = 0x05,	*beams*
seltRib = 0x06,	*ribs*
seltSurfaceEdge = 0x07,	*surface edges*
seltRigidBody = 0x08,	*rigid bodies*
seltDiaphragm = 0x09,	*diaphragms*
seltGap = 0x0A,	*gap elements*
seltSpring = 0x0B,	*springs*
seltLink = 0x0C,	*link elements*
seltAllSurfaces = 0x0D,	*surfaces*
seltSurfaceMembrane = 0x0E,	*membrane elements*
seltSurfacePlate = 0x0F,	*plate elements*
seltSurfaceShell = 0x10,	*shell elements*

seltAllDomains = 0x11, *domains*
seltDomainMembrane = 0x12, *membrane domains*
seltDomainPlate = 0x13, *membrane domains*
seltDomainShell = 0x14, *shell domains*
seltHole = 0x15, *hole*
seltAllSupports = 0x16, *all supports*
seltNodalSupport = 0x17, *nodal supports*
seltLineSupport = 0x18, *line supports*
seltSurfaceSupport = 0x19, *surface supports*
seltReference = 0x1A, *references*
seltAllLoads = 0x1B, *all loads*
seltLoadDomainConcentrated = 0x1C, *concentrated loads on domains*
seltLoadDomainPoly = 0x1D, *polygonal loads on domains*
seltLoadNodalConcentrated = 0x1E, *nodal loads*
seltLoadBeamConcentrated = 0x1F, *concentrated loads on beams*
seltLoadBeamDistributed = 0x20, *distributed loads on beams*
seltLoadDomainDistributed = 0x21, *distributed loads on domains*
seltLoadDomainFluid = 0x22 } *fluid loads on domains*
seltLoadMoving = 0x23, *moving loads*
seltLoadDynamic = 0x24, *dynamic loads*
seltArchColumn = 0x25, *architectuctural elements - columns*
seltArchBeam = 0x26, *architectuctural elements - beams*
seltArchWall = 0x27, *architectuctural elements - walls*
seltArchSlab = 0x28, *architectuctural elements - slabs*
seltArchRamp = 0x29, *architectuctural elements - ramps*
seltLoadPanel = 0x2A, *load panels*
seltLoadPanelEdge = 0x2B *edge of load panels*
seltVirtualBeam = 0x2C, *virtual beams*
seltLinesOnly = 0x2D *non-member(undefined) lines*
seltNN_Link = 46, *node-node link element*
seltLL_Link = 47, *line-line link element*
seltEdgeHinge = 48, *edge hinge*
seltDimension = 49, *dimension (any type)*
seltOrthoDimension = 50, *orthogonal dimension line*
seltAlignedDimension = 51, *aligned dimesnion line*
seltAngleDimension = 52, *angle dimension*
seltLevelDimension = 53, *level mark dimension*
seltElevDimension = 54, *elevation mark dimension*
seltTextBoxDimension = 55, *text box*
seltResultBoxDimension = 56, *result box*
seltIsoLineDimension = 57, *isoline label*
seltInfoBoxDimension = 58, *info text box*
seltArcLengthDimension = 59, *arc length*
seltRadiusDimension = 60, *radius dimension*
seltLoadDynamicNSupportDisplacement = 61, *dynamic nodal support displacement load*
seltLayerLine = 62, *layer line*
seltLayerArc = 63, *layer arc*
seltLayerPolygon = 64, *layer polygon*
seltLayerFilledPolygon = 65, *layer filled polygon*
seltLayerDXFText = 66, *layer DXF text*
seltLayerDXFDimension = 67, *layer DXF dimension*
seltLoadPanelEdge_Snow = 68, *load panel edge for snow loads*
seltLoadPanelEdge_Wind = 69, *load panel edge for wind loads*

seltSteelMember = 70,	*steel structural member*
seltWoodMember = 71,	*wood structural member*
seltRC_Column = 72,	*reinforced concrete column*
seltRC_Beam = 73,	*reinforced concrete beam*
seltDomainSolid = 74,	*solid domain*
seltDomainRibbed = 75,	*ribbed domain*
seltDomainXLAM = 76,	*XLAM domain*
seltDomainHollowCore = 77,	*hollow core domain*
seltDomainComposite = 78,	*composite ribbed domain*
seltDomainTrapezoidalSteel = 79,	*trapezoidal steel deck domain*
seltDomainCustomStiffnessMatrix = 80,	*domain with custom stiffness matrix*
selt7DOFBeam = 81,	*7 DOF beam*

}

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,	*see AxisVM manual for more info*
idoAnchored = 0x1,	*see AxisVM manual for more info*
idoOverRun = 0x2 }	*see AxisVM manual for more info*

How to deal with overlapping domains.

enum **EIFCimportMethod** = {

iimStaticModel = 0x0,	*structural elements, see AxisVM manual for more info*
iimArchitecturalModelObjects = 0x1 }	*architectural elements, see AxisVM manual for more info*

What type of elements should be used for IFC import

enum **EIFCopeningsAlignedToDomainEdge** = {

ioatdeImportAsOpenings = 0x0,	*openings imprted as elements*
ioatdeAdjustTheDomain = 0x1 }	*contour of domain adjusted for openings*

How to import openings

enum **ECompanyLogoPosition** = {

clpNoLogo = 0x0,	*no logo*
clpLeft = 0x1,	*logo on the left*
clpRight = 0x2,	*logo on the right*
clpTopLeft = 0x3,	*logo on the top left*
clpTopCenter = 0x4,	*logo on the top center*
clpTopRight = 0x5 }	*logo on the top right*

enum **ECompanyLogoSizeOption** = {

clsoAuto = 0x0,	*automatic*
clsoWidth = 0x1,	*width*
clsoHeight = 0x2 }	*height*

enum **EGeneralAlignmentHorizontal** = {

gahLeft = 0x0,	*justified to the left horizontally*
gahRight = 0x1,	*justified to the right horizontally*
gahCenter = 0x2 }	*justified to the center horizontally*

enum **EGeneralAlignmentVertical** = {

gavTop = 0x0,	*justified to the top vertically*
gavBottom = 0x1,	*justified to the bottom vertically*
gavCenter = 0x2 }	*justified to the center vertically*

enum **EaxsImportCustomParts** = {

aicpToActiveCustomParts = 0x0,	*all custom parts to active part*
aicpPreserveNames = 0x1,	*custom part names of imported model remain same*
aicpNoCustomParts = 0x2 }	*ignore custom parts of imported model*

Records / structures

RWindowPosition = (

long	**Top**	Top
long	**Left**	Left
long	**Width**	Width
long	**Height**	Height

)

RIFCExportReinforcementParams = (

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*
EIFCDomainReinforcementLinkType	**DomainReinforcementLinkType**	*Link type for IFC export, see manual*
EIFCQuestionableDomainLink	**QuestionableDomainLink**	*type of link export, see manual*
EIFCDomainOverlap	**DomainOverlap_Larger**	*see manual*
EIFCDomainOverlap	**DomainOverlap_Smaller**	*see manual*
ElongBoolean	**ModifyClosedLinks**	*see manual*
double	**ClosedLinkRatio**	*see manual*

)

RIFCimportParameters = (

EFileImportMode	**ImportMode**	*import mode of files*
EIFCimportMethod	**ImportMethod**	*type of elements used for ifc import*
double	**MaxDeviation**	*max. deviation of the arc, ignored if zero*
double	**ByAngle**	*angle division of the arc, ignored if zero*
double	**JoinIfObjectsAreCloserThan**	*join closer objects*
EFileImportAs	**ImportAs**	*how to import lines and nodes.*
EIFCopeningsAlignedToDomainEdge	**OpeningsAlignedToDomainEdge**	*how to import openings*

)

RDXFimportParameters = (

ELengthUnit	**CoordinateUnit**	*type units used in file*
double	**MaxDeviation**	*max. deviation of the arc, ignored if zero*
double	**GeometryCheckTolerance**	*used for geometry check*
double	**CoordinateScaleFactor**	*scale factor*
EFileImportAs	**ImportAs**	*how to import lines and nodes.*
EFileImportMode	**ImportMode**	*import mode of files*
EIFCimportMethod	**ImportMethod**	*type of elements used for ifc import*
EFileImportPlane	**BasePlane**	*base plane*
Rpoint3D	**PlaceOffset**	*Offset of the origin*
ElongBoolean	**VisibleLayersOnly**	*see AxisVM manual*
ElongBoolean	**ImportHatch**	*see AxisVM manual*
ElongBoolean	**ActivateDXFonAllDrawings**	*see AxisVM manual*
long	**WorkPlaneIndex**	*index of the workplane*

)

RPDFimportParameters = (

long	**PageNumber**	*page number , first is 1*
double	**MaxDeviation**	*max. deviation of the arc, ignored if zero*
double	**GeometryCheckTolerance**	*used for geometry check*
double	**Scale**	*scale factor*
ElongBoolean	**ImportLineWidth**	*line width from file will used*
ElongBoolean	**ImportText**	*if lbTrue, then text will be also imported*
EFileImportAs	**ImportAs**	*how to import lines and nodes.*
EFileImportMode	**ImportMode**	*import mode of files*
EIFCimportMethod	**ImportMethod**	*type of elements used for ifc import*
EFileImportPlane	**BasePlane**	*base plane*
Rpoint3D	**PlaceOffset**	*Offset of the origin*
long	**WorkPlaneIndex**	*index of the workplane*

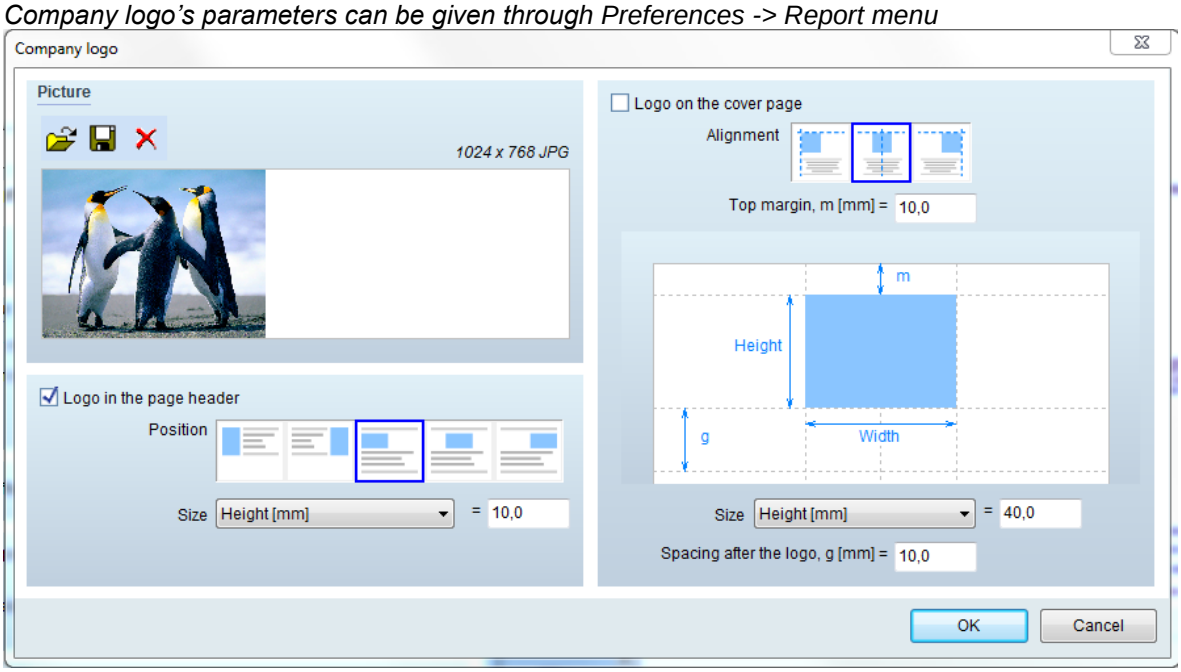
)

RCompanyLogoParameters = (

ElongBoolean	**ShowInHeader**	*show in header*
ECompanyLogoPosition	**HeaderPosition**	*position of the header*
ECompanyLogoSizeOption	**HeaderSizeOption**	*size option of the logo in header*
double	**HeaderSize**	*size of the logo in header in mm*
ElongBoolean	**ShowOnCover**	*show on cover*
EGeneralAlignment	**CoverAlignmentHorizontal**	*logo's horizontal alignement on the cover page*
double	**TopMargin**	*top margin's size*
double	**SpacingAfter**	*spacing after the logo*
ECompanyLogoSizeOption	**CoverSizeOption**	*size option of the logo on the cover page*
double	**CoverSize**	*size of the logo on the cover page in mm*

)

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



RAXSimportParameters = (

RPoint3d	**Place**	*origin of imported model*
double	**CheckTolerance**	*geometry tolerance*
ELongBoolean	**MergeLoadCases**	*merge load cases with the same name, otherwise new load cases are created*
ELongBoolean	**UserInteraction**	*If lbTrue dialog messages will be shown*
ELongBoolean	**IgnoreDesignCode**	*Proceed even if national design code of imported model is different*
ELongBoolean	**MergeLayers**	*merge layers with the same name, otherwise new layers are created*
EaxsImportCustomParts	**CustomParts**	*how to import openings*

)

JSON string properties

BSTR	**jsonFileName**	*file name of json string*
double	**rAura**	*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***
double	**rTessDistance**	*Maximal deviation from the arc for tessellation. Default:* ***0,05***
double	**rEps**	*Tolerance used in geometric calcluations. Default:* ***1E-05***
long	**rTessDegree**	*Tesselation degree Default:* ***5***
ElongBoolean	**rTessByDegree**	*Certain geometric calcluations require arcs to be tesselated. This can happen either by an angle parameter or by maximal deviation from the arc. Default:* ***True***

ElongBoolean	**rModStatModell**	*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. Default:* ***True***
ElongBoolean	**rClearBefore**	*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***
long	**rDefMatType**	*National design code:* ENationalDesignCode *, default:* ***ndcEuroCode*** *as long*
string	**rDefMatName**	*Name of default material, default*: ***25/30***
ElongBoolean	**rSuppressDialogs**	*Hide import dialogs, default:* ***True***

Error codes

enum **EModelError** = {

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

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

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**)

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)*

If successful it returns 1, otherwise it returns error (meDXFmoduleNotAvailable , meCannotExport, errDatabaseNotReady).

---

long **ExportToIFC** ([in] BSTR **FileName,** [in] EIFCVersion **IFCVersion,**
[in] ELongBoolean **ArchitectModel,** [in] ELongBoolean **SelectedOnly,**
[in] ELongBoolean **ExportReinforcement,**
[in] RIFCExportReinforcementParams **ExportReinforcement**)

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

If successful it returns 1, otherwise it returns error (meIFCmoduleNotAvailable, meCannotExport, errDatabaseNotReady).

---

long **ExportToPIA** ([in] BSTR **FileName,** [in] ELongBoolean **SelectedOnly**)

FileName	*file name*
SelectedOnly	*if True and the selection is not empty only the selected elements are exported, otherwise the entire model is exported*

Exports the model or a selection to a PIA (PianoCA interface) file. Possible error codes are meCannotExport, errDatabaseNotReady.

FitInView

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

---

long **GetAPIGlobalData** ([in] BSTR **DataName**, [out] SAFEARRAY (byte) * **APIGlobalData**)

DataName	*name of the saved API data, recommended to use also the API name (max 32 chars)*
APIGlobalData	*custom API data*

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).

---

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

DataName	*name of the saved API data, recommended to use also the API name (max 32 chars)*

Import DXF file. If successful it returns 1, otherwise an error code.

---

long **GetCompanyLogoParameters** ([i/o] RCompanyLogoParameters **Parameters,**
[out] BSTR **LogoFileName**)

Parameters *parameters of company's logo*
LogoFileName *logo's filename*

Get filename and parameters of the company's logo. If succesfull it retruns 1.0, otherwise it returns error (e.g. errDatabaseNotReady).

*Note: Supported extensions for LogoFileName are: *.png; *.jpg; *.jpeg; *.bmp; *.tif; *.tiff; *.ico; *.emf; *.wmf.*

For further details see the picture by the description of RCompanyLogoParameters.

---

long **SetCompanyLogoParameters** ([i/o] RCompanyLogoParameters **Parameters,**
[in] BSTR **LogoFileName**)

Parameters *parameters of company's logo*
LogoFileName *logo's filename*

Set filename and parameters of the company's logo. If succesfull it retruns 1.0, otherwise it returns error (e.g. errDatabaseNotReady).

*Note: Supported extensions for LogoFileName are: *.png; *.jpg; *.jpeg; *.bmp; *.tif; *.tiff; *.ico; *.emf; *.wmf.*

For further details see the picture by the description of RCompanyLogoParameters.

---

long **GetIFCExportReinfParams** ([i/o] RIFCExportReinforcementParams **Value**)

Value *IFC reinforcement export* parameters

Get actual reinforcement parametrs for IFC export. If successful it returns 1, otherwise an error code.

---

long **GetSeismicParams** ([i/o] RSeismicParams **Value**)

Warning! *This function has become obsolete, was superseded by GetSeismicParams_V153*

Duplicate of function GetSeismicParams. Retained for compatibility but will be removed in later version.

---

long **ImportDXF** ([in] BSTR **FileName,** [i/o] RDXFimportParameters **Parameters**)

FileName *file name*
Parameters *import* parameters

Import DXF file. If successful it returns 1, otherwise an error code.

---

long **ImportIFC** ([in] BSTR **FileName,** [i/o] RIFCimportParameters **Parameters**)

FileName *file name*
Parameters *import* parameters

Import IFC file. If successful it returns 1, otherwise an error code.

---

long **ImportRAE** ([in] BSTR **raeFileName,** [in] BSTR **jsonFileName**)

raeFileName *file name*
jsonFileName *file name of json string*

Import RAE file. If successful it returns 1, otherwise an error code (errDatabaseNotReady, meRevitImportTessDegreeOutOfRange, errJSONpropertyMissing).

---

long **ImportPDF** ([in] BSTR **FileName,** [i/o] RPDFimportParameters **Parameters**)

FileName *file name*
Parameters *import* parameters

Import PDF file. If successful it returns 1, otherwise an error code.

---

long **ImportAXS** ([in] BSTR **FileName,** [i/o] RAXSimportParameters **Parameters**)

FileName *file name*
Parameters *import* parameters

Import AXS file. If successful it returns 1, otherwise an error code.

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 **PickCoordinate** ([in] BSTR **HintMessage,** [i/o] RPoint3d **Coord**)

HintMessage *hint message to be displayed at the bottom bar of AxisVM*

Coord *the picked up coordinate*

Starts a coordinate picking task in AxisVM. The user can input a coordinate either through a mouse click in the active view, or through typing in a value in the coordinate window. If there is a successfull point acquisition, the return value is 1. If the point selection is canceled by the user, returns 0. In case of an error returns a negative error code(errDatabaseNotReady)

---

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. This is a non blocking function call, your program will continue with the next statement after this function call. If you are looking for a blocking function, use StartModalSelection instead. Allowed selection types can be set by constructing a double-byte Unicode string of characters encoding ESelectionType constants (0x01-0x22).). This encoding is language dependent, here is an example in Delphi for selecting nodes and domains : WideChar(seltNode) + WideChar(seltAllDomains).

This function will trigger the IAxisVMModelsEvents.SelectionProcessingChanged event with NewStatus = True when the selection task has started and with NewStatus = False when the selection has ended (either with OK or Cancel).

A negative return value indicates an error, which prevented the selection to start. The only such return value is errDatabaseNotReady.

ELongBoolean **StartModalSelection** ([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. It blocks your code until it returns, either due to an error or a user interaction. Allowed selection types can be set by constructing a double-byte Unicode string of characters encoding ESelectionType constants (0x01-0x22). This encoding is language dependent, here is an example in Delphi for selecting nodes and domains : WideChar(seltNode) + WideChar(seltAllDomains). If the return value is 1, the user terminated the selection with Ok. If the return value is 0, the user terminated the selection with Cancel. A negative return value indicates an error, which prevented the selection to start. The only such return value is errDatabaseNotReady.

---

long **SetSeismicParams** ([i/o] RSeismicParams **Value**)

Warning! *This function has become obsolete, was superseded by SetSeismicParams_V153*

Duplicate of function SetSeismicParams. Retained for compatibility but will be removed in later version.

Undo

Revert to an older state of the model when SaveUndo function was called

---

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**
Access to the Actual Reinforcement of surfaces(domains) (extension module RC1 is required)

BSTR* **AnalysisBy**
Get or set name of the project designer (engineer).

AxisVMCalculation* **Calculation**
Access to the solver of AxisVM.

ElongBoolean	**CallProgress** • *If set to lbTrue, any progress bar movements on main AxisVM form will call COM event IAxisVMModelsEvents.MainProgress* *Read and write property.*
AxisVMColumnRebars*	**ColumnRebars** *Access to the column rebars in the RC column (extension module RC2 is required)*
BSTR*	**Comment** *Get or set comment about (description of) the current model (project).*
AxisVMCriticalGroupCombinations*	**CriticalGroupCombinations** *Access to the* critical group combinations *of the model.*
AxisVMCrossSections*	**CrossSections** *Access to the cross-sections of the model.*
AxisVMCustomParts*	**CustomParts** *Access to the* custom parts *of the model.*
AxisVMDiaphragm*	**Diaphragm** *Access to the diaphragms of the model*
AxisVMDimensions*	**Dimensions** *Access the dimensions*
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*
AxisVMEnvelopes*	**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 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.

ELongBoolean **NeedsSaving**
True, if the model has been modified so it needs saving. Read only property.

AxisVMNodalSupports* **NodalSupports**
Access to the node supports of the model.

AxisVMNodesSupports* **NodesSupports**
Access to the nodal supports of the model.

AxisVMNodes* **Nodes**
Access to the nodes of the model.

EAxisVMPlatform **Platform**
Get whether it is 32 or 64 bit version of AxisVM

BSTR* **ProjectName**
Name (title) of the current project. Read and write property.

AxisVMPushoverHingeFunctions* **PushoverHingeFunctions**
Access to the hinge functions for pushover (extension module SE2 is required)

AxisVMRCBeamDesign* **RCBeamDesign**
Access to the RC beam design (extension module RC2 is required)

AxisVMRCColumnChecking* **RCColumnChecking**
Access to the RC column checking (extension module RC2 is required)

AxisVMReferences* **References**
Access to the references of the model.

AxisVMReports* **Reports**
Access to the reports of the model.

AxisVMResults* **Results**
Access to the results of the model.

AxisVMRebarSteelGrades* **RebarSteelGrades**
Access to the rebar steel grades

AxisVMRigidBodies* **RigidBodies**
Access to the rigid bodies of the model.

ELongBoolean **SelectionProcessing**
True, if AxisVM waits for the user to complete the selection. Read only property.

ELongBoolean **SelectionResult**
True, if the user clicked the OK button of the selection toolbar. Read only property.

AxisVMSettings* **Settings -** *Get model settings.*

AxisVMSpectrum* **SpectrumH**
Warning! *This property has become obsolete, was superseded by SeismicSpectrumH*
Access to the horizontal spectrum of the first seismic group

AxisVMSpectrum* **SpectrumV**
Warning! *This property has become obsolete, was superseded by SeismicSpectrumV*
Access to the vertical spectrum of the first seismic group

AxisVMSpectrum* **SpectrumPushOver**
Access to the pushover spectrum of the model

AxisVMSeismicStoreys* **SeismicStoreys**
Access to the seismic storeys of the model

AxisVMCrossSectionOptimization* **SteelCrossSectionOptimization**
Access to the steel cross-section optimization (extension module SD9 is required)

AxisVMSteelDesignMembers* **SteelDesignMembers**
Access to the steel design (extension module SD1 is required)

AxisVMStructuralGrids* **StructuralGrids**
Access to structural grids

AxisVMSurfaces* **Surfaces**
Access to the surface elements of the model.

AxisVMSurfaceSupports* **SurfaceSupports**
Access to the surface supports of the model.

AxisVMSections* **Sections**
Access to the sections of the model.

AxisVMStoreys* **Storeys**
Access to storeys of the model.

AxisVMTask* **Task**
Run various tasks in AxisVM

AxisVMCrossSectionOptimization* **TimberCrossSectionOptimization**
Access to the timber cross-section optimization (extension module TD9 is required)

AxisVMTimberDesignMembers* **TimberDesignMembers**
Access to timber design (extension module TD1 is required)

AxisVMTimeIncrementFunctions* **TimeIncrementFunctions**
Access to time increment functions of the model (extension module DYN is required)

EView **View •**
Get or set view of the active window also available in AxisVMWindows and AxisVMWindow interface

AxisVMVirtualBeams* **VirtualBeams**
Access to virtual beams/strips of the model.

AxisVMWindows* **Windows**
Access to all windows, their view settings and displayed load case.

AxisVMWorkplanes* **Workplanes**
Access to workplanes of the model.

AxisVMXLAMpanels* **XLAMpanels**
Access to XLAM panels of the model.

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**= {

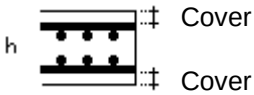
areCOMError = -100001	*COM server error*
areDomainIdOutOfBounds = -100002	*IAxisVMActualReinforcement.AddDomainReinforcement or AxisVMActualReinforcement.IndexOfDomainReinforcement can return this error when the DomainId parameter is invalid*
areErrorAddingPolygonReinforcement = -100003	*IAxisVMActualReinforcement.AddPolygonReinforcement can return this when reinforcement parameters are missing*
areDomainReinforcementNotFound = -100004	*IAxisVMActualReinforcement.IndexOfDomainReinforcement could not find reinforcement for the domain*
areSurfaceIdOutOfBounds = -100005	*SurfaceId is invalid.*
areSurfaceVertexIndexOutOfBounds = -100006	*SurfaceVertexIndex is invalid.*
areDiameterIsInvalid = -100007	*Diameter is invalid.*
areSpacingIsInvalid = -100008	*Spacing is invalid.*
areCoverIsInvalid = -100009	*Cover is invalid.*
areReinfParamMissing = -1000010 }	*Domain has no reinforcement parameters*

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, errIndexOutOfBounds)

---

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 ≤ Index ≤ Count*
ActualReinforcementType	*artDomain: Polygon variable won't be filled ; artPolygon: DomainId variable won't be filled*
DomainId	*index of the domain, 1 ≤ Index ≤ Count*
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 **SurfaceId,** [in] ESurfaceVertexIndex
SurfaceVertexIndex, [out] SAFEARRAY(RActualReinforcement)*** X_Bottom,** [out]
SAFEARRAY(RActualReinforcement)*** X_Top,** [out] SAFEARRAY(RActualReinforcement)*****
Y_Bottom, [out] SAFEARRAY(RActualReinforcement)*** Y_Top**)

SurfaceId	*index of the surface, 1 ≤ Index ≤ 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 SurfaceId, otherwise returns an error code (areSurfaceIdOutOfBounds).

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

SurfaceId *index of the surface, 1 ≤ Index ≤ 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 SurfaceId, otherwise returns an error code (areSurfaceIdOutOfBounds).

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 x N elements for surface of quad mesh and surface of triangular mesh will have also 9 x 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] ELongBoolean **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

ELongBoolean **Selected** [long Index] ***

Index *index of the actual reinforcement, 1 ≤ Index ≤ 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 ≤ Index ≤ Count*

IAxisVMAttachments

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

Error codes

enum **EAttachmentsError** = {

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

Functions

long **AddData** ([in] BSTR **Name**, [in] long **ItemIndex**, [in] SAFEARRAY (byte) * **ItemData**)

Name *Name of the saved attachment data, recommended to include also the API name (max. 32 chars) e.g "attachment A"*
ItemIndex *Index of the element/item of the parent interface (Nodes, lines, etc.)*
ItemData *attachment data,this data will be assigned to an item/elemt with index ItemIndex*

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.

---

long **AddData_vb** ([in] BSTR **Name**, [in] long **ItemIndex**, [i/o] SAFEARRAY (byte) * **ItemData**)

Visual Basic compatible function of AddData.

---

long **GetData** ([in] BSTR **Name**, [in] long **ItemIndex**, [out] SAFEARRAY (byte) * **ItemData**)

Name *Name of the saved attachment data*
ItemIndex *Index of the element/item of the parent interface (Nodes, lines, etc.)*
ItemData *attachment data*

Get attachment data from the axs file. If successful it returns attachment size, otherwise it returns error.

---

long **GetSize** ([in] BSTR **Name**, [in] long **ItemIndex**)

Name *Name of the saved attachment data*
ItemIndex *Index of the element/item of the parent interface (Nodes, lines, etc.)*

Get size of attachment data from the axs file. If successful it returns attachment data size, otherwise it returns error.

---

long **Delete** ([in] BSTR **Name**, [in] long **ItemIndex**)

Name *Name of the saved attachment data*
ItemIndex *Index of the element/item of the parent interface (Nodes, lines, etc.)*

Delete attachment data from the axs file. If successful it returns ItemIndex, otherwise it returns error.

Properties

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

IAxisVMAttributes

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

Functions

long **AddDefault** ([in] BSTR **Name**, [in] SAFEARRAY (byte) * **ItemData**)

Name *Attribute name of the saved attribute data, recommended to include also the API name (max. 32 chars) e.g "AttributeA"*

ItemData *default attribute data,this data will be assigned to all items and also to new ones created from now on.*

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).

---

long **AddDefault_vb** ([in] BSTR **Name**, [i/o] SAFEARRAY (byte) * **ItemData**)

Visual Basic compatible function of AddDefault.

---

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

Name *Attribute name of the saved attribute data*

Delete attribute data by name. If successful it returns 1, otherwise it returns error (atteCannotDeleteAttribute, atteAttributeNotFound, atteInvalidName errDatabaseNotReady).

---

long **DeleteByIndex** ([in] long **AttributeIndex**)

AttributeIndex *Attribute index of the saved attribute data*

Delete attribute data by attribute index. If successful it returns 1, otherwise it returns error (atteCannotDeleteAttribute, atteAttributeNotFound, errIndexOutOfBounds, errDatabaseNotReady).

---

long **GetDefault** ([in] BSTR **Name**, [out] SAFEARRAY (byte) * **ItemData**)

Name *Attribute name of the saved attribute data*

ItemData *default attribute data*

Get default attribute data from the axs file. If successful it returns attribute size, otherwise it returns error (atteAttributeNotFound, errIndexOutOfBounds, errDatabaseNotReady).

---

long **GetName** ([in] long **AttributeIndex,** [out] BSTR **Name**)

AttributeIndex *Attribute index of the saved attribute data*

Name *Attribute name of the saved attribute data*

Get attribute name. If successful it returns attribute size, otherwise it returns error (atteAttributeNotFound, errIndexOutOfBounds, errDatabaseNotReady).

---

long **GetItemByIndex** ([in] long **AttributeIndex**, [in] long **ItemIndex**,
[out] SAFEARRAY (byte) * **ItemData**)

AttributeIndex *Attribute index of the saved attribute data*
ItemIndex *item (element) index , 1 to ParentInteface.count*
ItemData *custom attribute data of the item*
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).

---

long **GetItemByName** ([in] BSTR **Name**, [in] long **ItemIndex**,
[out] SAFEARRAY (byte) * **ItemData**)

Name *Attribute name of the saved attribute data*
ItemIndex *item (element) index , 1 to ParentInteface.count*
ItemData *custom attribute data of the item*
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).

---

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

Name *Attribute name of the saved attribute data*
Get attribute data size by name. If successful it returns attribute size, otherwise it returns error (atteAttributeNotFound, atteInvalidName, errDatabaseNotReady).

---

long **GetSizeByIndex** ([in] long **AttributeIndex**)

AttributeIndex *Attribute index of the saved attribute data*
Get attribute data size by index. If successful it returns attribute size, otherwise it returns error (atteAttributeNotFound, errIndexOutOfBounds, errDatabaseNotReady).

---

long **FillItemByIndex** ([in] long **AttributeIndex**, [in] long **ItemIndex**,
[in] SAFEARRAY (byte) * **ItemData**)

AttributeIndex *Attribute index of the saved attribute data*
ItemIndex *item (element) index , 1 to ParentInteface.count*
ItemData *custom attribute data, length must match length of default attribute data*
Owerwrite attribute data of one item (element) with new data. If successful it returns attribute index, otherwise it returns error (atteCannotSetAttribute, atteInvalidOrEmptyItemData, atteAttributeSizeMustMatch, atteItemIndexOutOfBounds, errIndexOutOfBounds, errDatabaseNotReady).

---

long **FillItemByName** ([in] BSTR **Name**, [in] long **ItemIndex**,
[in] SAFEARRAY (byte) * **ItemData**)

Name *Attribute name of the saved attribute data*
ItemIndex *item (element) index , 1 to ParentInteface.count*
ItemData *custom attribute data, length must match length of default attribute data*
Owerwrite attribute data of one item (element) with new data. If successful it returns attribute index, otherwise it returns error (atteCannotSetAttribute, atteInvalidOrEmptyItemData, atteAttributeSizeMustMatch, atteItemIndexOutOfBounds, atteAttributeNotFound, atteInvalidName, errDatabaseNotReady).

---

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**)

AttributeIndex *Attribute index of the saved attribute data*
ItemData *custom attribute data, length must match length of default attribute data*
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).

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*

Owerwrite 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 ParentInteface.count*
ItemData *custom attribute data, length must match length of default attribute data*

Owerwrite 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 ParentInteface.count*
ItemData *custom attribute data, length must match length of default attribute data*

Owerwrite 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).

ElongBoolean **IsDefaultItemByIndex** ([in] long **AttributeIndex**, [in] long **ItemIndex**)

AttributeIndex *Attribute index of the saved attribute data*
ItemIndex *item (element) index , 1 to ParentInteface.count*

If returns lbTrue then it is the default attribute.

ElongBoolean **IsDefaultItemByName** ([in] BSTR **Name**, [in] long **ItemIndex**)

Name *Attribute name of the saved attribute data*
ItemIndex *item (element) index , 1 to ParentInteface.count*

If returns lbTrue 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)*
Owerwrite attribute data of all items (elements) with new data. If successful it returns attribute index, otherwise it returns error (atteCannotSetAttribute, atteInvalidOrEmptyItemData, atteItemsDataMustContainAllItems, errIndexOutOfBounds, 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)*
Owerwrite 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 tha parent interface elements*

ELongBoolean **InheritCopiedElements** **[**long **AttributeIndex]** • *Get or set wheteher copied elements should copy their custom data to the new copied elements or use default data*

ELongBoolean **InheritDividedElements** **[**long **AttributeIndex]** • *Get or set wheteher 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*
ceInvalidResultCase = –100005	*for ReinforcementCalculation = lbTrue and ReinforcementCalculationType = rcCalculated ReinforcementCase isn't a valid result case*
ceReinfPreConditionError = –100006	*for ReinforcementCalculation = lbTrue the model doesn't have the required data (applied reinforcement somewhere in case of ReinforcementCalculationType = rcActual or the calculated ReinforcementCase in case of ReinforcementCalculationType = rcCalculated)*
ceNodeIndexOutOfBounds = –100007	*node index is out of range*

}

Enumerated types

enum **EAnalysisCaseType** = {

actCase = 1,	*actual load case or load combination.The load combination start from the last load case, if there are 3 load cases then the 2nd load combination will have the index 3+2 = 5*
actEnvelopeMin = 2,	*envelope min*
actEnvelopeMax = 3,	*envelope max*
actEnvelopeMinMax = 4,	*envelope min/max*
actCriticalMin = 5,	*critical min*
actCriticalMax = 6,	*critical max*
actCriticalMinMax = 7 }	*critical min/max*

Analysis case type, as seen in the load case tree.

enum **EDynAnalDLI** = {

dadli_Linear = 1,	*linear interpolation*
dadli_WhittakerShannon = 2 }	*Whittaker-Shannon interpolation*

Dynamic load interpolation method.

enum **ECalculationUserInteraction** = {

cuiUserInteraction = 0x0,	*user has to interact with the program to answer questions arising during the analysis (e.g. domains are not meshed, degress of freedom not set correctly)*
cuiNoUserInteractionWithAutoCorrect = 0x1,	*AxisVM tries to correct problems automatically during the analysis*
cuiNoUserInteractionWithoutAutoCorrect = 0x2,	*AxisVM stops if a problem is detected during the analysis*
cuiNoUserInteractionWithAutoCorrectNoShow = 0x3,	*same as cuiNoUserInteractionWithAutoCorrect, but the calculation window won't be visible*
cuiNoUserInteractionWithoutAutoCorrectNoShow = 0x4,	*same as cuiNoUserInteractionWithoutAutoCorrect, but the calculation window won't be visible*

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

Note:

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

enum **EMassControl** = {

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

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

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

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

enum **EMassesTakenIntoAccount** = {

mtiaAll = 0x0,	*consider all masses*
mtiaAboveHeightZ = 0x1	*consider only masses above specific height*
mtiaAboveSelectedStory = 0x2 }	*consider only masses above specific story*

Setting used in vibration analysis regarding considered masses.

Records / structures

RNonLinearAnalysis = (

Warning! *This record was superseded by RNonLinearAnalysis_V162*

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

)

These settings correspond to settings on Nonlinear Static Analysis form.

RNonLinearAnalysis_V162 = (

long	**LoadCase**	*load case or load combination to analyse* ***IMPORTANT NOTE:*** *If it is the index of the load combination then: Load combination index =* **LoadCase** *– IAxisVMLoadcases.count*
long	**Iterations**	*maximum number of iterations*
ESolutionControl	**SolutionControl**	*way of solution control*
long	**NodeId**	*index of the node for displacement control. In case of SolutionControl in [scDisplacement,scArcLength,scPushOver], it must be a valid node index. In case of SolutionControl = scForce, 0 indicates that there is no tracked node. A greater than 0 value should be a valid node index for tracking*
EAxis	**Direction**	*direction of displacement for displacement control*
ELongBoolean	**MaterialNonLinearity**	*Enable/disable material non-linearity*
double	**MaxDisplacement**	*maximum displacement for displacement control [m]*
long	**Increments**	*number of increments*
double	**DisplacementConvergenceValue**	*convergence criterion for displacement*
double	**ForceConvergenceValue**	*convergence criterion for force*
double	**WorkConvergenceValue**	*convergence criterion for work*
ELongBoolean	**EnableDisplacementConvergence**	*DisplacementConvergenceValue enabled*
ELongBoolean	**EnableForceConvergence**	*ForceConvergenceValue enabled*
ELongBoolean	**EnableWorkConvergence**	*WorkConvergenceValue enabled*
ELongBoolean	**GeometricNonlinearity**	*taking into account the geometric nonlinearity*
ELongBoolean	**ReinforcementCalculation**	*taking into account the reinforcement*
ELongBoolean	**StoreLastIncrementOnly**	*only the last increment is stored*
EReinforcementCalculation	**ReinforcementCalculationType**	*use actual or calculated reinforcement if ReinforcementCalculation = True*
ELongBoolean	**ContinueWithoutConvergence**	*If true then analysis continues without convergence*
long	**IncrementFunctionId**	*Index of the increment function, see here. If this value is set to 0, equal increments will be used*

ELongBoolean	**ConsiderCreep**	*consider the effect of creep on the stiffness of RC structures More here…*
ELongBoolean	**ConsiderShrinkage**	*consider shrinkage strain by the calculation of RC structures*
ELongBoolean	**SecantStiffness**	*secant stiffness*
ELongBoolean	**StrainIncAutoConstraint**	*iteration control : automatic constraint for strain increments*
ELongBoolean	**DisplacemetIncConstraint**	*iteration control : constraint for displacement increments*
double	**DIC_Displacement**	*if DisplacemetIncConstraint is lbTrue, the value of displacement [m]*
double	**DIC_Rotation**	*if DisplacemetIncConstraint is lbTrue, the value of rotation [rad]*
RResultCase	**ReinforcementCase**	*if reinforcement calculation is taken into account and its type is rcCalculated, then this is the source case of the calculation*

)

These settings correspond to settings on Nonlinear Static Analysis form.

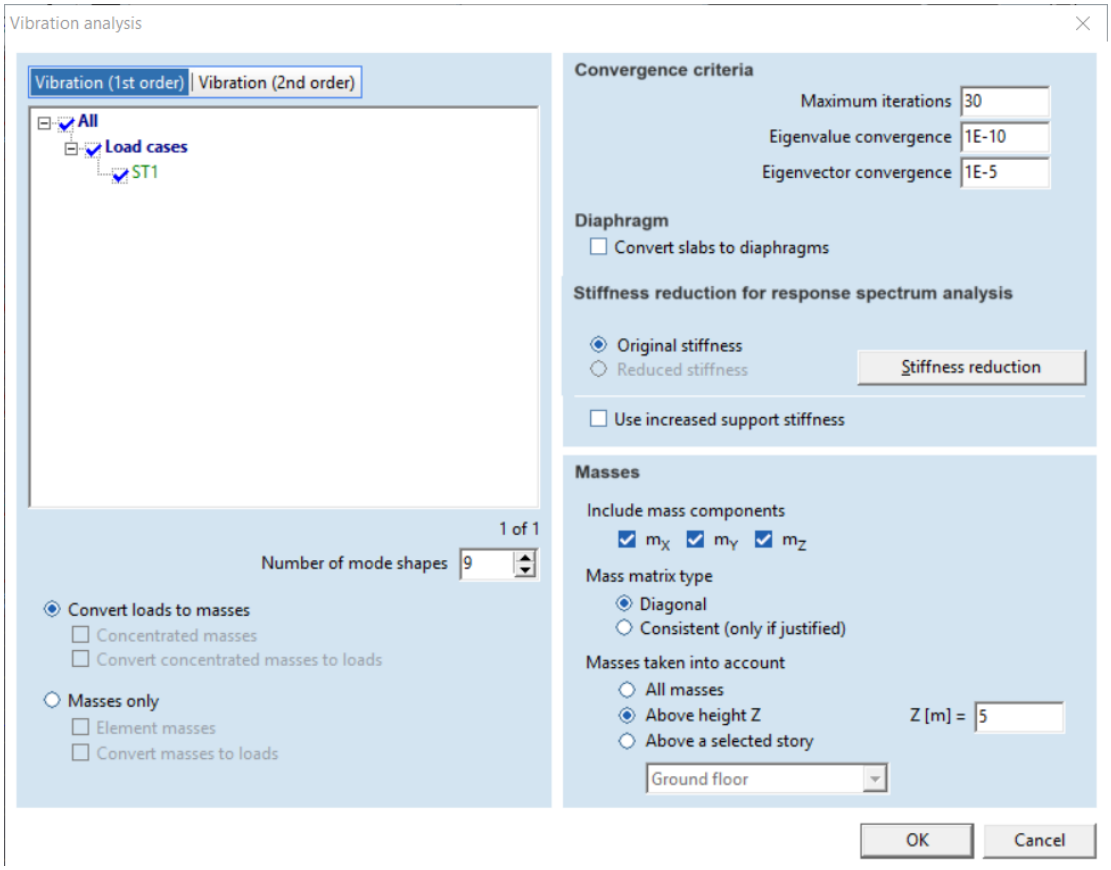
RVibration = (

Warning! *This record was superseded by RVibration_V162*

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

)

These settings correspond to settings on Vibration Analysis form.



RVibration_V162 = (

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

)

These settings correspond to settings on Vibration Analysis form.

RBuckling = (

Warning! *This record was superseded by RBuckling_V162*

long	**LoadCase**	*load case or load combination to analyse* ***IMPORTANT NOTE:*** *If it is the index of the load combination then:* *Load combination index* = **LoadCase** – *IAxisVMLoadcases.count*
long	**Iterations**	*maximum number of iterations*

long	**ModeShapes**	*number of buckling shapes to determine*
double	**EigenValueConvergence**	*eigenvalue convergence criterion*
double	**EigenVectorConvergence**	*eigenvector convergence criterion*

)

These settings correspond to settings on Buckling analysis form.

RBuckling_V162 = (

long	**LoadCase**	*load case or load combination to analyse* ***IMPORTANT NOTE:*** *If it is the index of the load combination then: Load combination index =* **LoadCase** *– IAxisVMLoadcases.count*
long	**Iterations**	*maximum number of iterations*
long	**ModeShapes**	*number of buckling shapes to determine*
double	**EigenValueConvergence**	*eigenvalue convergence criterion*
double	**EigenVectorConvergence**	*eigenvector convergence criterion*
ELongBoolean	**OnlyPositiveEigenvalues**	*keep only positive eigenvalues*

)

These settings correspond to settings on Buckling analysis form.

RDynamicAnalysis = (

Warning! *This record was superseded by RDynamicAnalysis_V162*

long	**LoadCase**	*dynamic load case index to analyse*
long	**StaticLoadCase**	*load case or load combination to analyse* ***IMPORTANT NOTE:*** *If it is the index of the load combination then: Load combination index =* **StaticLoadCase** *– IAxisVMLoadcases.count*
long	**LoadCombination**	*optional static load combinationID which can be enabled in analysis*
long	**TimeIncrementFunctionId**	*Index of the time increment function, see here*
double	**TimeIncrement**	*Time increment [s]*
long	**NumberOfIncrements**	*Number of increments*
double	**a**	*Rayleigh damping coefficient a*
double	**b**	*Rayleigh damping coefficient b*
EMassMatrixType	**MassMatrixType**	*type of mass matrix*
ELongBoolean	**ConvertLoadsToMasses**	*Convert loads to masses*
ELongBoolean	**ConcentratedMasses**	*Concentrated masses used also*
ELongBoolean	**ConvertConcentratedMassesToLoads**	*Convert concentrated masses to loads*
EMassControl	**MassControl**	*type of mass control*
ELongBoolean	**ElementMasses**	*taking into account the element masses*
ELongBoolean	**MaterialNonLinearity**	*use material nonLinearity*
ELongBoolean	**GeometricNonLinearity**	*use geometric nonLinearity*
ELongBoolean	**PerformIterations**	*Iterate or not*
long	**Iterations**	*number of iterations*
double	**DisplacementConvergenceValue**	*displacement convergence value*
double	**ForceConvergenceValue**	*force convergence value*
double	**WorkConvergenceValue**	*work convergence value*
ELongBoolean	**EnableDisplacementConvergence**	
ELongBoolean	**EnableForceConvergence**	
ELongBoolean	**EnableWorkConvergence**	
ELongBoolean	**ContinueWithoutConvergence**	*Analysis continues without convergence*
double	**SavingInterval**	*interval of saving results [s]*
ELongBoolean	**LoadsAndNodalMassesForDamping**	*both loads and masses are taken into account for damping (used with Rayleigh damping coefficient a)*

)

These settings correspond to settings on dynamic analysis form.

RDynamicAnalysis_V162 = (

long	**LoadCase**	*dynamic load case index to analyse*
long	**StaticLoadCase**	*load case or load combination to analyse* ***IMPORTANT NOTE:*** *If it is the index of the load combination then: Load combination index =* **StaticLoadCase** *– IAxisVMLoadcases.count*
long	**LoadCombination**	*optional static load combinationID which can be enabled in analysis*
long	**TimeIncrementFunctionId**	*Index of the time increment function, see here*
double	**TimeIncrement**	*Time increment [s]*
long	**NumberOfIncrements**	*Number of increments*
double	**a**	*Rayleigh damping coefficient a*
double	**b**	*Rayleigh damping coefficient b*
EMassMatrixType	**MassMatrixType**	*type of mass matrix*
ELongBoolean	**ConvertLoadsToMasses**	*Convert loads to masses*
ELongBoolean	**ConcentratedMasses**	*Concentrated masses used also*
ELongBoolean	**ConvertConcentratedMassesToLoads**	*Convert concentrated masses to loads*
EMassControl	**MassControl**	*type of mass control*
ELongBoolean	**ElementMasses**	*taking into account the element masses*
ELongBoolean	**MaterialNonLinearity**	*use material nonLinearity*
ELongBoolean	**GeometricNonLinearity**	*use geometric nonLinearity*

ELongBoolean	**PerformIterations**	*Iterate or not*
long	**Iterations**	*number of iterations*
double	**DisplacementConvergenceValue**	*displacement convergence value*
double	**ForceConvergenceValue**	*force convergence value*
double	**WorkConvergenceValue**	*work convergence value*
ELongBoolean	**EnableDisplacementConvergence**	
ELongBoolean	**EnableForceConvergence**	
ELongBoolean	**EnableWorkConvergence**	
ELongBoolean	**ContinueWithoutConvergence**	*Analysis continues without convergence*
double	**SavingInterval**	*interval of saving results [s]*
ELongBoolean	**LoadsAndNodalMassesForDamping**	*both loads and masses are taken into account for damping (used with Rayleigh damping coefficient a)*
ELongBoolean	**LoadsToMasses**	*in case of static load case convert load to masses*
long	**NodeID**	*tracked node*
EAxis	**Axis**	*direction*
ELongBoolean	**StrainIncAutoConstraint**	*iteration control : automatic constraint for strain increments*
ELongBoolean	**DisplacemetIncConstraint**	*iteration control : constraint for iteration increments*
double	**DIC_Displacement**	*if DisplacemetIncConstraint is lbTrue, the value of displacement [m]*
double	**DIC_Rotation**	*if DisplacemetIncConstraint is lbTrue, the value of rotation [rad]*
EDynAnalDLI	**DynLoadInterpolation**	*dynamic load interpolation method*
ELongBoolean	**ZeroFinalSpeed**	*set zero final speed*
ELongBoolean	**ZeroFinalDisplacement**	*set zero final displacement*

)

These settings correspond to settings on dynamic analysis form.

RPushOverAnalysis = (

Warning! *This record was superseded by RPushOverAnalysis_V162*

long	**LoadCase**	*load case or load combination to analyse* ***IMPORTANT NOTE:*** *If it is the index of the load combination then: Load combination index =* **LoadCase** *–IAxisVMLoadcases.count*
long	**ConstantLoadCase**	*optional static load case ID which can be enabled in analysis*
long	**NodeId**	*index of the node for displacement control*
EAxis	**Direction**	*direction of displacement for displacement control*
ELongBoolean	**MaterialNonLinearity**	*Enable/disable material non-linearity*
double	**MaxDisplacement**	*maximum displacement for displacement control [m]*
long	**Increments**	*number of increments*
long	**Iterations**	*Number of iterations*
double	**DisplacementConvergenceValue**	*convergence criterion for displacement*
double	**ForceConvergenceValue**	*convergence criterion for force*
double	**WorkConvergenceValue**	*convergence criterion for work*
ELongBoolean	**EnableDisplacementConvergence**	*DisplacementConvergenceValue enabled*
ELongBoolean	**EnableForceConvergence**	*ForceConvergenceValue enabled*
ELongBoolean	**EnableWorkConvergence**	*WorkConvergenceValue enabled*
ELongBoolean	**GeometricNonlinearity**	*taking into account the geometric nonlinearity*
ELongBoolean	**ContinueWithoutConvergence**	*If true then analysis continues without convergence*

)

These settings correspond to settings on Nonlinear Static Analysis form.

RPushOverAnalysis_V162 = (

long	**LoadCase**	*load case or load combination to analyse* ***IMPORTANT NOTE:*** *If it is the index of the load combination then: Load combination index =* **LoadCase** *–IAxisVMLoadcases.count*
long	**ConstantLoadCase**	*optional static load case ID which can be enabled in analysis*
long	**NodeId**	*index of the node for displacement control*
EAxis	**Direction**	*direction of displacement for displacement control*
ELongBoolean	**MaterialNonLinearity**	*Enable/disable material non-linearity*
double	**MaxDisplacement**	*maximum displacement for displacement control [m]*
long	**Increments**	*number of increments*
long	**Iterations**	*Number of iterations*
double	**DisplacementConvergenceValue**	*convergence criterion for displacement*
double	**ForceConvergenceValue**	*convergence criterion for force*
double	**WorkConvergenceValue**	*convergence criterion for work*
ELongBoolean	**EnableDisplacementConvergence**	*DisplacementConvergenceValue enabled*
ELongBoolean	**EnableForceConvergence**	*ForceConvergenceValue enabled*
ELongBoolean	**EnableWorkConvergence**	*WorkConvergenceValue enabled*
ELongBoolean	**GeometricNonlinearity**	*taking into account the geometric nonlinearity*
ELongBoolean	**ContinueWithoutConvergence**	*If true then analysis continues without convergence*
long	**IncrementFunctionId**	*increment function ID (IAxisVMIncrementFunctions)*
ELongBoolean	**SecantStiffness**	*secant stiffness*
ELongBoolean	**StrainIncAutoConstraint**	*iteration control : automatic constraint for strain increments*
ELongBoolean	**DisplacemetIncConstraint**	*iteration control : constraint for iteration increments*
double	**DIC_Displacement**	*if DisplacemetIncConstraint is lbTrue, the value of displacement [m]*
double	**DIC_Rotation**	*if DisplacemetIncConstraint is lbTrue, the value of rotation [rad]*

These settings correspond to settings on Nonlinear Static Analysis form.

RResultCase = (

EAnalysisType	**AnalysisType**	*analysis type*
EAnalysisCaseType	**CaseType**	*case type*
long	**CaseId**	*case ID, only for CaseType = actCase*
long	**CaseSubIndex**	*sub index, only for CaseType = actCase and specific AnalysisTypes.*
long	**EnvelopeUID**	*envelope UID, only for envelope CaseTypes*
long	**CriticalIndex**	*critical combination type, only for critical CaseTypes. The value 9 can be used for automatic selection. For specific values for a given national code, the CriticalIndex for a given state of the case selector drop down menu can be queried with IAxisVMWindows.GetResultCase*
ELongBoolean	**Creep**	*creep, only for specific cases*
ELongBoolean	**Imperfection**)	*imperfection, only for specific cases*

Result case selector

Functions

ELongBoolean **Buckling** ([i/o] RBuckling **AnalysisParameters,**
[in] ECalculationUserInteraction **UserInteraction**)

Warning! *This function has become obsolete, was superseded by Buckling_V162*

AnalysisParameters *analysis parameters*
UserInteraction *type of interaction during calculation*

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

ELongBoolean **Buckling2** ([i/o] RBuckling **AnalysisParameters,**
[in] ECalculationUserInteraction **UserInteraction,**
[in] **ELongBoolean ShowError,** [out] BSTR* **ErrorList**)

Warning! *This function has become obsolete, was superseded by Buckling2_V162*

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*

Buckling analysis of the model. Same as Buckling, but with extended options in the call interface. Returns True if the analysis was completed.

ELongBoolean **Buckling_V162** ([i/o] RBuckling_V162 **AnalysisParameters,**
[in] ECalculationUserInteraction **UserInteraction**)

AnalysisParameters *analysis parameters*
UserInteraction *type of interaction during calculation*

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

ELongBoolean **Buckling2_V162** ([i/o] RBuckling_V162 **AnalysisParameters,**
[in] ECalculationUserInteraction **UserInteraction,**
[in] **ELongBoolean ShowError,** [out] BSTR* **ErrorList**)

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*

Buckling analysis of the model. Same as Buckling_V162, but with extended options in the call interface. Returns True if the analysis was completed.

ELongBoolean **DynamicAnalysis** ([i/o] RDynamicAnalysis* **AnalysisParameters,**
[in] ECalculationUserInteraction **UserInteraction,**
[in] ELongBoolean **ShowError,** [out] BSTR* **ErrorList**)

Warning! *This function has become obsolete, was superseded by DynamicAnalysis_V162*

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*

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

ELongBoolean **DynamicAnalysis_V162** ([i/o] RDynamicAnalysis_V162* **AnalysisParameters,**
[in] ECalculationUserInteraction **UserInteraction,**
[in] ELongBoolean **ShowError,** [out] BSTR* **ErrorList**)

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*

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

ELongBoolean **LinearAnalysis** ([in] ECalculationUserInteraction **UserInteraction**)

UserInteraction	*type of interaction during calculation*

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

ELongBoolean **LinearAnalysis2** ([in] ECalculationUserInteraction **UserInteraction,**
[in] ELongBoolean **ShowError**, [out] BSTR* **ErrorList**)

UserInteraction *type of interaction during calculation*

ShowError *show error message in AxisVM, ErrorList is always returned*

ErrorList *message with errors after analysis*

Linear analysis of the model. Same as LinearAnalysis, but with extended options in the call interface. Returns True if the analysis was completed.

ELongBoolean **NonLinearAnalysis** ([i/o] RNonLinearAnalysis **AnalysisParameters,**
[in] ECalculationUserInteraction **UserInteraction**)

Warning! *This function has become obsolete, was superseded by NonLinearAnalysis_V162*

AnalysisParameters *analysis parameters*

UserInteraction *type of interaction during calculation*

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

ELongBoolean **NonLinearAnalysis2** ([i/o] RNonLinearAnalysis **AnalysisParameters,**
[in] ECalculationUserInteraction **UserInteraction,**
[in] ELongBoolean **ShowError**, [out] BSTR* **ErrorList**)

Warning! *This function has become obsolete, was superseded by NonLinearAnalysis2_V162*

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*

Nonlinear analysis of the model. Same as NonLinearAnalysis, but with extended options in the call interface. Returns True if the analysis was completed.

ELongBoolean **NonLinearAnalysis_V162** ([i/o] RNonLinearAnalysis_V162 **AnalysisParameters,** [in] ECalculationUserInteraction **UserInteraction**)

AnalysisParameters *analysis parameters*

UserInteraction *type of interaction during calculation*

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

ELongBoolean **NonLinearAnalysis2_V162** ([i/o] RNonLinearAnalysis_V162 **AnalysisParameters,**
[in] ECalculationUserInteraction **UserInteraction,**
[in] ELongBoolean **ShowError**, [out] BSTR* **ErrorList**)

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*

Nonlinear analysis of the model.Same as NonLinearAnalysis_V162, but with extended options in the call interface. Returns True if the analysis was completed.

ELongBoolean **NonLinearVibration** ([i/o] RVibration **AnalysisParameters,**
[in] ECalculationUserInteraction **UserInteraction**)

Warning! *This function has become obsolete, was superseded by NonLinearVibration_V162*

AnalysisParameters *analysis parameters*

UserInteraction *type of interaction during calculation*

Second order vibration analysis of the model. Returns True if the analysis was completed.

ELongBoolean **NonLinearVibration2** ([i/o] RVibration **AnalysisParameters,**
[in] ECalculationUserInteraction* **UserInteraction,** [in] ELongBoolean **ShowError**,
[out] BSTR* **ErrorList**)

Warning! *This function has become obsolete, was superseded by NonLinearVibration2_V162*

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*

Second order vibration analysis of the model. Same as NonLinearVibration, but with extended options in the call interface. Returns True if the analysis was completed.

ELongBoolean **NonLinearVibration_V162** ([i/o] RVibration_V162 **AnalysisParameters,**
[in] ECalculationUserInteraction **UserInteraction**)

AnalysisParameters *analysis parameters*

UserInteraction *type of interaction during calculation*

Second order vibration analysis of the model. Returns True if the analysis was completed.

ELongBoolean **NonLinearVibration2_V162** ([i/o] RVibration_V162 **AnalysisParameters,**
[in] ECalculationUserInteraction* **UserInteraction,** [in] ELongBoolean **ShowError**,
[out] BSTR* **ErrorList**)

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*

Second order vibration analysis of the model. Same as NonLinearVibration, but with extended options in the call interface. Returns True if the analysis was completed.

ELongBoolean **Vibration** ([i/o] RVibration **AnalysisParameters,**
[in] ECalculationUserInteraction **UserInteraction**)

Warning! *This function has become obsolete, was superseded by Vibration_V162*

AnalysisParameters *analysis parameters*

UserInteraction *type of interaction during calculation*

First order vibration analysis of the model. Returns True if the analysis was completed.

ELongBoolean **Vibration2** ([i/o] RVibration **AnalysisParameters,**
[in] ECalculationUserInteraction **UserInteraction,** [in] ELongBoolean **ShowError**,
[out] BSTR* **ErrorList**)

Warning! *This function has become obsolete, was superseded by Vibration2_V162*

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*

First order vibration analysis of the model. Same as Vibration, but with extended options in the call interface. Returns True if the analysis was completed.

ELongBoolean **Vibration_V162** ([i/o] RVibration_V162 **AnalysisParameters,**
[in] ECalculationUserInteraction **UserInteraction**)

AnalysisParameters	*analysis parameters*
UserInteraction	*type of interaction during calculation*

First order vibration analysis of the model. Returns True if the analysis was completed.

ELongBoolean **Vibration2_V162** ([i/o] RVibration_V162 **AnalysisParameters,**
[in] ECalculationUserInteraction **UserInteraction,** [in] ELongBoolean **ShowError**,
[out] BSTR* **ErrorList**)

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*

First order vibration analysis of the model. Same as Vibration_V162, but with extended options in the call interface. Returns True if the analysis was completed.

ELongBoolean **PushOverAnalysis** ([i/o] RPushOverAnalysis **AnalysisParameters,**
[in] ECalculationUserInteraction **UserInteraction,** [in] ELongBoolean **ShowError**,
[out] BSTR* **ErrorList**)

Warning! *This function has become obsolete, was superseded by PushOverAnalysis_V162*

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*

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

ELongBoolean **PushOverAnalysis_V162** ([i/o] RPushOverAnalysis_V162 **AnalysisParameters,** [in] ECalculationUserInteraction **UserInteraction,** [in] ELongBoolean **ShowError**,
[out] BSTR* **ErrorList**)

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*

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

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*

IAxisVMColorLegend

Color legend interface of AxisVM.

Enumerated types

enum	**EColorLegendDirection** = {	enum
	cld_BlueRed = 0x1,	*bottom blue, top red*
	cld_RedBlue = 0x2 }	*bottom red, top blue*
enum	**EColorLegendType** = {	enum
	clt_AutoFull = 0x1,	*absolute max. of model*
	clt_AutoPart = 0x2,	*absolute max. of parts*
	clt_Manual = 0x3,	*custom values*
	clt_MinMaxFull = 0x4,	*min, max of model*
	clt_MinMaxPart = 0x5 }	*min, max. of parts*

Functions

long **GetDirection** ([out] EColorLegendDirection **Direction**)

Direction *placement direction of blue red excess*

Returns a positive number on success.

---

long **GetForcedPozNeg** ([out] ELongBoolean **ForcedPozNeg**)

ForcedPozNeg *state of the forced PozNeg mode. When lbTrue, the color legend only has a positive and negative color (and is fixed to 3 levels)*

Returns a positive number on success.

---

long **GetLegendType** ([out] EColorLegendType **LegendType**)

LegendType *source of values (custom values, full model, parts)*

Returns a positive number on success.

---

long **GetLevels** ([out] long **Levels**)

Levels *number of values. Automatic values will be recalculated in steps with accordance to it.*

Returns a positive number on success.

---

long **GetValues** ([i/o] SAFEARRAY(double) * **Values**)

Values *values of the color legend. While it should be an out only parameter, for compatibility with various languages it is and i/o variable, meaning it has to be initialized prior to calling (an empty/null/nil value is enough).*

Returns a positive number on success.

---

long **SetDirection** ([in] EColorLegendDirection **Direction**)

Direction *placement direction of blue red excess*

Returns a positive number on success.

---

long **SetForcedPozNeg** ([in] ELongBoolean **ForcedPozNeg**)

ForcedPozNeg *state of the forced PozNeg mode. When lbTrue, the color legend only has a positive and negative color (and is fixed to 3 levels)*

Returns a positive number on success.

---

long **SetLegendType** ([in] EColorLegendType **LegendType**)

LegendType *source of values (custom values, full model, parts)*

Returns a positive number on success.

---

long **SetLevels** ([in] long **Levels**)

Levels *number of values. Automatic values will be recalculated in steps with accordance to it.*

Returns a positive number on success. If Levels is outside of the valid range for levels, errIndexOutOfBounds is returned.

---

long **SetManualValues** ([i/o] SAFEARRAY(double) * **Values**)

Values *values of the color legend.*

Calling this function will set the legend type to clt_Manual, and the number of levels to the length of Values. Returns a positive number on success. If the length of Values is outside of the valid range for levels, errIndexOutOfBounds is returned.

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** = {
crbeInvalidCrossSectionId = –100001 } *invalid cross section index*

Enumerated types

enum **ESeismicDuctilityClass** = { enum
sdc_DCM = 0x0, *medium ductility class*
sdc_DCH = 0x1 } *high ductility class*

Records / structures

RColumnRebarPos = (

double	**x**	*local x coordinate of the rebar from bottom left corner of the cross section*
double	**y**	*local y coordinate of the rebar from bottom left corner of the cross section*
double	**d**)	*diameter of the rebar*

RColumnCheckingParameters = (

ELongBoolean	**CalcEccIncz**	*calculate eccentricity in local z direction*
ELongBoolean	**CalcEccIncy**	*calculate eccentricity in local y direction*
ELongBoolean	**CalcEcc2Y**	*calculate second order eccentricity in local y direction*
ELongBoolean	**CalcEcc2Z**	*calculate second order eccentricity in local z direction*
double	**Ky**	*buckling length factor in local x-z plane of the column*
double	**Kz**	*buckling length factor in local x-y plane of the column*
ELongBoolean	**IsCustomLength**	*over write length of the column*
double	**ColumnIength**	*length of the column (only for custom length)*
double	**fieff_EC_ITA**	φ_{eff} *effective creep ratio (only for EuroCode and Italian DesignCode), see 5.8.4 in EN 1992-1-1:2004(E)*
ELongBoolean	**SwayImp**	consider sway geometric imperfections (see 5.2 (7) in EN 1992-1-1)
long	**BucklingModeY**	*buckling mode in local x-z plane of the column*
long	**BucklingModeZ**	*buckling mode in local x-y plane of the column*
ELongBoolean	**ApproximateCurvatureSIA**	*consider approximate curvature in calculation of second order eccentricities (used only if SIA standard is selected)* *lbTrue:* $\chi_d = \frac{2f_{sd}}{E_s(d-d')}$ *lbFalse:* $\chi_d = \frac{\varepsilon_{sd}-\varepsilon'_{sd}}{d-d'} + \frac{\lvert\varepsilon_{c,\infty}\rvert}{d}$
double	**ShrinkCreepEpsSIA**	*shrinkage + creep strain (used only if SIA standard is selected)*
ELongBoolean	**Sum2ndEccentricities**	*consider both 2nd eccentricities calculated in local y and z directions simultaneously or separately*

)

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

1  **6** 

2

3

4

5

7

8

9

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*

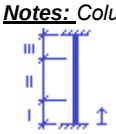
)

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 trirrup 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	**CheckingParameters**	*column checking parameters*
double	**fse**	*load factor for seismic forces (default is 1 =>no change)*
ELongBoolean	**TakeConcTensileStrength**	*take tensile strength of the concrete into account in nonlinear analysis*
ELongBoolean	**Usefctmfl**	*consider flexural tensile strength in nonlinear analysis instead of tensile strength (only for EC and ITA)*
double	**ShrinkageEps**	*shrinkage strain considered in nonlinear analysis*
ELongBoolean	**SpiralStirrup**	*use spiral stirrup (only for circular sections)*
RColumnStirrupSpacing	**StirrupSpacing**	*spacing between stirrups*
RColumnStirrupDiameters	**StirrupDiameters**	*diameter of stirrups*
RColumnStirrupZones	**StirrupZones**	*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	**CapacityDesignParams**	*parameters for capacity design*
ELongBoolean	**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 = (

ELongBoolean	**PlasticHinges**	*consider plastic hinges in the column*
ESeismicDuctilityClass	**DuctilityClass**	*ductility class*
ELongBoolean	**PlasticHingeYY_Top**	*plastic hinge can be formed due to yy bending moment at the top of the column*
ELongBoolean	**PlasticHingeYY_Bottom**	*plastic hinge can be formed due to yy bending moment at the bottom of the column*
ELongBoolean	**PlasticHingeZZ_Top**	*plastic hinge can be formed due to zz bending moment at the top of the column*
ELongBoolean	**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, crbeInvalidCrossSectionId).

---

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 ≤ Index ≤ 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 ≤ Index ≤ 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**

Adds a new group combination. It will be without linking of groups. All the possible load groups will be set to used. Returns critical group combination index if successful, otherwise returns an error code (errDatabaseNotReady).

---

long **Clear**

Delete all critical group combinations. A default new critical group combination will be automatically created after the clear. 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*

Deletes a critical group combination. Returns critical group combination index if successful, otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBounds).

---

long **Validate** ( )

Deletes duplicate entries, and merges linked groups if they have at least one LinkingEditable = lbTrue element in common. Returns a positive number if successful, otherwise returns an error code (errDatabaseNotReady).

Properties

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

ELongBoolean **Editable** [long **LoadGroupId**] *Returns lbTrue if can be switched on/off for consideration in critical results when linking of groups is off.*

LoadGroupId *index of the load group, 1 ≤ LoadGroupId ≤ IAxisVMLoadGroups.Count*

ELongBoolean **Linking** [long **Index**] • *Get or set the linking of groups property of a critical group combination. Warning! Changing the value of this property will result in filling with default values of the Used fields of the given critical group combination, according to the new value of linking.*

Index *index of critical group combination, 1 ≤ index ≤ Count*

ELongBoolean **LinkingEditable** [long **LoadGroupId**] *Returns lbTrue if can be switched on/off for consideration in critical results, when linking of groups is on.*

LoadGroupId *index of the load group, 1 ≤ LoadGroupId ≤ IAxisVMLoadGroups.Count*

ELongBoolean **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 ≤ index ≤ Count*

LoadGroupId *index of the load group, 1 ≤ LoadGroupId ≤ IAxisVMLoadGroups.Count*

IAxisVMCrossSections

Cross-sections of the model.

Error codes

enum **ECrossSectionError** = {

cseNotAllowedProcessForShape = –100001	*process information is not compatible with the shape*
cseNotAllowedProcessForParameters = –100002	*process information is not compatible with the parameters*
cseNonPositive_b = –100003	*b ≤ 0*
cseNonPositive_h = –100004	*h ≤ 0*
cseNonPositive_tw = –100005	*tw ≤ 0*
cseNonPositive_tf = –100006	*tf ≤ 0*
cseNonPositive_d = –100007	*d ≤ 0*
cseNonPositive_v = –100008	*v ≤ 0*
cseNonPositive_b1 = –100009	*b1 ≤ 0*
cseNonPositive_h1 = –100010	*h1 ≤ 0*
cseNonPositive_b2 = –100011	*b2 ≤ 0*
cseNonPositive_h2 = –100012	*h2 ≤ 0*
cseNonPositive_tw1 = –100013	*tw1 ≤ 0*
cseNonPositive_tf1 = –100014	*tf1 ≤ 0*
cseNonPositive_tw2 = –100015	*tw2 ≤ 0*
cseNonPositive_tf2 = –100016	*tf2 ≤ 0*
cseNonPositive_R = –100017	*R ≤ 0*
cseNegative_r1 = –100018	*r1 < 0*
cseNegative_r2 = –100019	*r2 < 0*
cseNegative_r3 = –100020	*r3 < 0*
cseNegative_e = –100021	*e < 0*
cseNegative_a = –100022	*a < 0*
cseTooHigh_h = –100023	*h too large*
cseTooHigh_tw = –100024	*tw too large*
cseTooHigh_tf = –100025	*tf too large*
cseTooHigh_r1 = –100026	*r1 too large*
cseTooHigh_r2 = –100027	*r2 too large*
cseTooHigh_r3 = –100028	*r3 too large*
cseTooHigh_v = –100029	*v too large*
cseTooLow_e = –100030	*e too small*
cseTooHigh_tw1 = –100031	*tw1 too large*
cseTooHigh_tf1 = –100032	*tf1 too large*
cseTooHigh_tw2 = –100033	*tw2 too large*
cseTooHigh_tf2 = –100034	*tf2 too large*
cseTooLow_h = –100035	*h too small*
cseTooLow_r1 = –100036	*r1 too small*
cseTooLow_N = –100037	*N too small*
cseDifferentThicknesses = –100038	*v ≠ t*
cseDifferentWidthAndHeight = –100039	*b ≠ h*
cseIncompatibleWidthAndHegiht = –100040	*b ≠ 2h*
cseNonPositiveAx = –100041	*Ax ≤ 0*
cseNegativeAy = –100042	*Ay < 0*
cseAyIsHigherThanAx = –100043	*Ay > Ax*
cseNegativeAz = –100044	*Az < 0*
cseAzIsHigherThanAx = –100045	*Ay > Ax*
cseNonPositiveIx = –100046	*Ix ≤ 0*
cseNonPositiveIy = –100047	*Iy ≤ 0*
cseNonPositiveIz = –100048	*Iz ≤ 0*
cseNonPositiveHy = –100049	*Hy ≤ 0*
cseNonPositiveHz = –100050	*Hz ≤ 0*
cseNegativeIw = –100051	*Iw < 0*
cseNegativeW1t = –100052	*W1t < 0*
cseNegativeW1b = –100053	*W1b < 0*
cseNegativeW2t = –100054	*W2t < 0*
cseNegativeW2b = –100055	*W2b < 0*
cseNegativeW1pl = –100056	*W1pl < 0*
cseNegativeW2pl = –100057	*W2pl < 0*
cseEmptyName = –100058	*cross-section name is empty*
cseNameAlreadyExists = –100059	*cross-section name already exists*
cseExtParams = –100060	*error in extended parameters*
cseErrorAdding = –100061	*error while adding or Cancel pressed*
cseErrorEditing = –100062	*error while editing, invalid cross-section or Cancel pressed*
cseInvalidCrossSectionType = -100063	*invalid cross section type (solid sections accepted)*
cseDifferentCrossSectionShape = -100064	*cross section shape does not match the section's shape*
cseTooHigh_e = –100065	*e too large*
cseTooLargeInnerCrossSection = –100066	*dous not fit into boundaries*
cseInvalidMaterials = –100067	*inner section should be metal*
cseNonPositive_b3 = –100068	*b3 ≤ 0*
cseNonPositive_h3 = –100069	*h3 ≤ 0*
cseNonPositive_tw3 = –100070	*tw3 ≤ 0*

cseNonPositive_tf3 = –100071	*tf3 ≤ 0*
cseTooHigh_tw3 = -100072	*tw3 > b3*
cseTooHigh_tf3 = -100073	*tf3 > h3*
cseTooLow_b = –100074	*b1 too small*
cseNonPositive_c = –100075	*c ≤ 0*
cseTooLow_c = –100076	*c < tw*
cseTooHigh_h2 = –100077	*h2 > b2*
cseTooLow_b2= –100078	*b2 < tw, b2 < b/2 (SFBA)*
cseTooLow_b1= –100079	*bu1 < b/2 (SFBA)*
cseTooLow_r2= –100080	*r2 is too small compared to r1 (Box)*
cseException = –101000 }	*other error*

Cross-section error codes.

Enumerated types

enum **ECrossSectionShape** = {

cssAll = 0x00000000,	*all shapes*
cssCustom = 0x00000001,	*custom shape*
cssRectangular = 0x00000002,	*rectangular shape*
cssI = 0x00000004,	*I shape*
cssDoubleI = 0x00000008,	*double I shape*
cssWedgedI = 0x00000010,	*wedged I shape*
cssAsymmetricI = 0x00000020,	*asymmetric I shape*
cssPipe = 0x00000040	*pipe*
cssRegularPolygon = 0x00000080,	*regular polygon*
cssBox = 0x00000100,	*box shape*
cssDoubleIBox = 0x00000200,	*double I box shape*
cssU = 0x00000400,	*U shape*
cssDoubleUOpened = 0x00000800,	*opened double U shape*
cssDoubleUClosed = 0x00001000,	*closed double U shape*
cssL = 0x00002000,	*unequal angle*
cssDoubleL = 0x00004000,	*double angle*
cssDoubleLFlange = 0x00008000,	*double angles connected at their flanges*
cssT = 0x00010000,	*T shape*
cssZ = 0x00020000,	*Z shape*
cssC = 0x00040000,	*C shape*
cssS = 0x00080000,	*S shape*
cssJ = 0x00100000,	*J shape*
cssCircle = 0x00200000 }	*circle shape*
cssRectangleRounded = 0x00400000,	*rectangle shape with rounded corners*
cssRectangleHollow = 0x00800000,	*rectangle shape with hollow inside*
cssIHaunched = 0x01000000,	*I shape with haunched flanges*
cssTWallHaunched = 0x02000000,	*T shape with haunched web*
cssTTopHaunched = 0x04000000,	*I shape with haunched top flange*
cssCircleHollow = 0x08000000,	*circle shape with hollow inside*
cssTrapezoid = 0x10000000,	*Symmetric trapezoid shape*
css2LX = 0x20000000,	*2 No. L-shapes*
css4L = 0x40000000,	*4 No. L-shapes (X cross)*

Cross-section shapes, supersceded with EcrossSectionShapeEx

enum **ECrossSectionShapeEx** = {

csseAll = 0x00000000,	*all shapes*
csseCustom = 0x00000001,	*custom shape*
csseRectangular = 0x00000002,	*rectangular shape*
csseI = 0x00000003,	*I shape*
csseDoubleI = 0x00000004,	*double I shape*
csseWedgedI = 0x00000005,	*wedged I shape*
csseAsymmetricI = 0x00000006,	*asymmetric I shape*
cssePipe = 0x00000007	*pipe*
csseRegularPolygon = 0x00000008,	*regular polygon*
csseBox = 0x00000009,	*box shape*
csseDoubleIBox = 0x00000010,	*double I box shape*

csseU = 0x00000011, *U shape*
csseDoubleUOpened = 0x00000012, *opened double U shape*
csseDoubleUClosed = 0x00000013, *closed double U shape*
csseL = 0x00000014, *unequal angle*
csseDoubleL = 0x00000015, *double angle*
csseDoubleLFlange = 0x00000016, *double angles connected at their flanges*
csseT = 0x00000017, *T shape*
csseZ = 0x00000018, *Z shape*
csseC = 0x00000019, *C shape*
csseS = 0x00000020, *S shape*
csseJ = 0x00000021, *J shape*
csseCircle = 0x00000022, *circle shape*
csseRectangleRounded = 0x00000023, *rectangle shape with rounded corners*
csseRectangleHollow = 0x00000024, *rectangle shape with hollow inside*
csseIHaunched = 0x00000025, *I shape with haunched flanges*
csseTWallHaunched = 0x00000026, *T shape with haunched web*
csseTTopHaunched = 0x00000027, *I shape with haunched top flange*
csseCircleHollow = 0x00000028, *circle shape with hollow inside*
csseTrapezoid = 0x00000029, *symmetric trapezoid shape*
csse2LX = 0x00000030, *2 No. L-shapes*
csse4L = 0x00000031, *4 No. L-shapes (X cross)*
csseCross = 0x00000032, *Cross shape*
csseCompositePipe = 0x00000033, *Composite section with outter steel pipe*
csseCompositeBox = 0x00000034, *Composite section with outter steel box*
csseCompositeRound = 0x00000035, *Round composite section with inner cross-section*
csseCompositeRectangle = 0x00000036, *Rectangular composite section with inner cross-section*
csseDoubleWedgedI = 0x00000037, *double wedged I shape*
csseHSQ = 0x00000038, *HSQ cross-section*
csseHSQA = 0x00000039 *HSQA cross-section*
csse2IX = 40, *crossed I shape*
csseComposite2IX = 41, *concrete encased crossed I shape*
csseIFB = 42, *IFB shape*
csseSFB = 43 *SFB shape*
csseDoubleLClosed = 44 *double L closed cross-section*

Cross-section shapes, replaces EcrossSectionShape

enum **ECrossSectionBasicType** = {
csbt_All = 0x0, *all basic types*
csbt_Solid = 0x1, *solid cross sections*
csbt_ThinWalled = 0x2, *thin walled cross sections*
csbtCompositeThinWalled = 0x4, *composite with inner and outter cross section*
csbtCompositeSolid = 0x8 } *composite with inner cross section*

Basic type of cross section

enum **ECrossSectionProcess** = {
cspOther = 0x0, *other*
cspRolled = 0x1, *rolled*
cspWelded = 0x2, *welded*
cspColdFormed = 0x3 } *cold-formed*

How the cross-section was manufactured.

enum **ECrossSectionDoubleUType** = { **duOpened** = 0x0, **duClosed** = 0x1}

Tells two types of double U-shapes apart (cstDoubleUOpened, cstDoubleUClosed).

enum **ECrossSectionImageExportOptions**= {

csieoNone= 0x0,	*none*
csieoFilled= 0x1,	*filled*
csieoStressPointsMarks= 0x2,	*marks of stress points*
csieoStressPointsLabels= 0x4	*labels of stress points*
csieoMainAxis= 0x8,	*main axis*
csieoMainAxisLabels= 0x16}	*main axis labels*

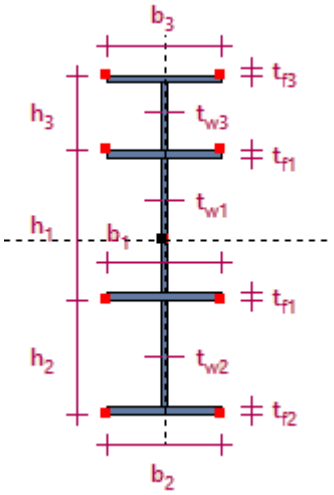
enum **ECompositeInnerCSalign** = { **cicsa_CentreGravity** = 0x0, **cicsa_Centre** = 0x1}

Not used yet, actual default: cicsa_CentreGravity

Records / structures

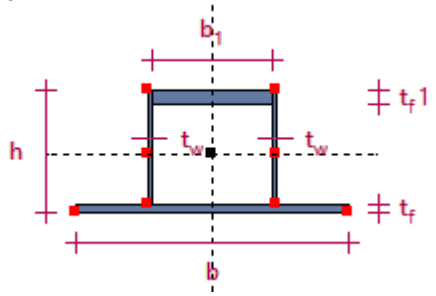
RDoubleWedgedI = (

ECrossSectionProcess	**Process**	*manufacturing process (can only be rolled or welded)*
double	**h1**	*middle I-shape height [m]*
double	**b1**	*middle I-shape width [m]*
double	**tw1**	*middle I-shape web thickness [m]*
double	**tf1**	*middle I-shape flange thickness [m]*
double	**r**	*fillet radius (for rolled process only) [m]*
double	**h2**	*lower I-shape height [m]*
double	**b2**	*lower I-shape width [m]*
double	**tw2**	*lower I-shape web thickness [m]*
double	**tf2**	*lower I-shape flange thickness [m]*
double	**h3**	*upper I-shape height [m]*
double	**b3**	*upper I-shape width [m]*
double	**tw3**	*upper I-shape web thickness [m]*
double	**tf3**)	*upper I-shape flange thickness [m]*



RCrossSectionHSQ = (

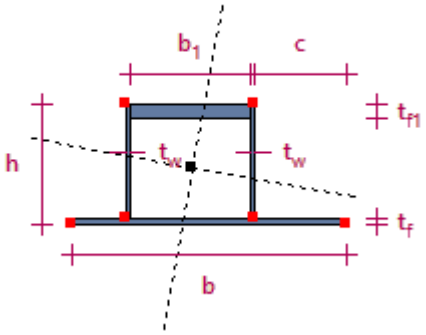
double	**h**	*total height [m]*
double	**b**	*base width [m]*
double	**b1**	*top width [m]*
double	**tw**	*web thickness [m]*
double	**tf1**	*top flange thickness [m]*
double	**tf)**	*base flange thickness [m]*



RCrossSectionHSQA = (

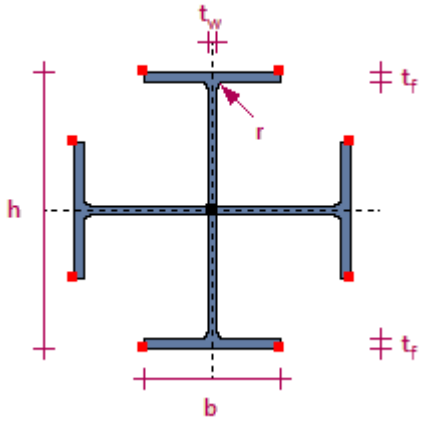
double	**h**	*total height [m]*

double	**b**	*base width [m]*
double	**b1**	*top width [m]*
double	**tw**	*web thickness [m]*
double	**tf1**	*top flange thickness [m]*
double	**tf**	*base flange thickness [m]*
double	**c)**	*offset from right [m]*



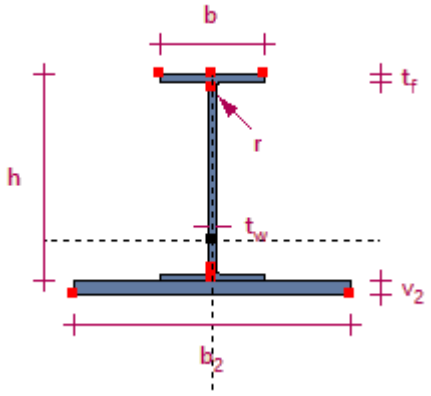
RCrossSection2IX = (

double	**h**	*total height [m]*
double	**b**	*flange width [m]*
double	**tw**	*web thickness [m]*
double	**tf**	*flange thickness [m]*
double	**r**	*fillet radius (for rolled process only) [m]*
ECrossSectionProcess	**Process)**	*manufacturing process (can be only rolled or welded)*



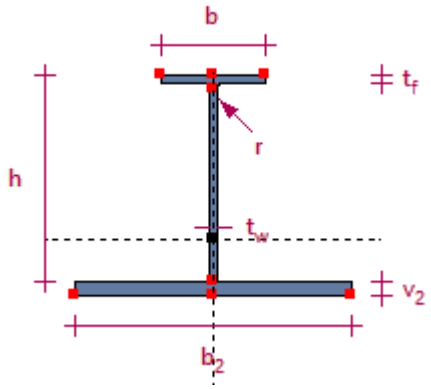
RCrossSectionSFB = (

double	**h**	*total height [m]*
double	**b**	*top flange width [m]*
double	**tw**	*web thickness [m]*
double	**tf**	*base flange thickness [m]*
double	**r**	*fillet radius (for rolled process only) [m]*
double	**b2**	*bottom flange width [m]*
double	**v2**	*bottom flange thickness [m]*
ECrossSectionProcess	**Process)**	*manufacturing process (can be only rolled or welded)*



RCrossSectionIFB = (

double	**h**	*total height [m]*
double	**b**	*top flange width [m]*
double	**tw**	*web thickness [m]*
double	**tf**	*base flange thickness [m]*
double	**r**	*fillet radius (for rolled process only) [m]*
double	**b2**	*bottom flange width [m]*
double	**v2**	*bottom flange thickness [m]*
ECrossSectionProcess	**Process)**	*manufacturing process (can be only rolled or welded)*

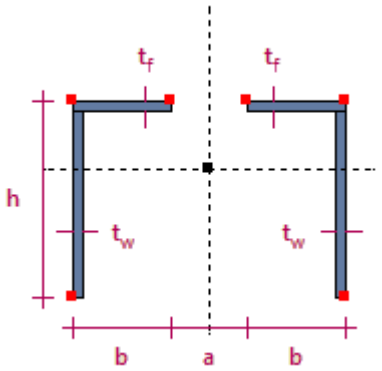


RCrossSectionComposite = (

long	**CrossSectionIndex**	*index of the inner cross-section. Must be of the correct type for that particular composite section. 1 <= Index <= AxisVMCrossSections.Count*
long	**OuterMaterialIndex**	*index of the outer material. 1 <= Index <= AxisVMMaterials.Count*
long	**CrossSectionMaterialIndex**	*index of the inner cross-section material. 1 <= Index <= AxisVMMaterials.Count*
long	**CrossSectionFillMaterialIndex**	*index of the fill material (if required for that particular composite section) 0 <= Index <= AxisVMMaterials.Count*
ECompositeInnerCSalign	**CrossSectionAlign)**	*not used (present for potential future compatibility)*

RDoubleLClosed = (

double	**h**	*total height [m]*
double	**b**	*flange width [m]*
double	**tw**	*web thickness [m]*
double	**tf**	*flange thickness [m]*
double	**r**	*fillet radius (for rolled process only) [m]*
double	**a**	*gap [m]*
ECrossSectionProcess	**Process)**	*manufacturing process (can be only rolled or welded)*



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 **Add2IX** ([in] BSTR **Name**, [in] RCrossSection2IX **Params**)

Name *name of the new cross-section*

Params *parameters of the new cross-section*

Adds a crossed I-shape to the model.

---

long **AddComposite2IX** ([in] BSTR **Name**, [in] RCrossSectionComposite **Params**)

Name *name of the new cross-section*

Params *parameters of the new cross-section*

Adds a concrete encased crossed I shape to the model.

---

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*

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).

---

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*

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 **AddFromEditor** ([in] long **CrossSectionBasicTypes**, [in] long **CrossSectionShapes**)

CrossSectionBasicTypes *cross-section types (ECrossSectionBasicType value or sum of values)*

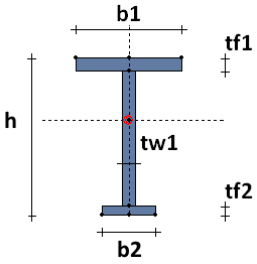
CrossSectionShapes *cross-section shapes (ECrossSectionShape value or sum of values)*

Adds a cross-section from a cross section editor. If successful, returns index of the cross section, otherwise an error (cseErrorAdding).

---

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*

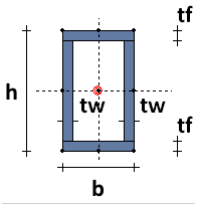


Adds an asymmetic I-shape to the model.

---

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*

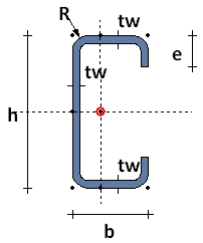


Adds a box shape to the model.

---

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*

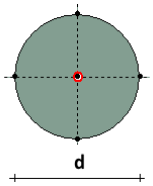


Adds a C-shape to the model.

---

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

Name	*name of the new cross-section*
d	*circle diameter*

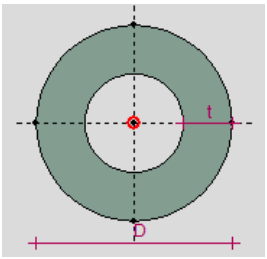


Adds a circle shape to the model.

---

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 **OutterMaterial_ID,** [in] long **InnerMaterial_ID,** [in] long **CS_MaterialID,** [in] long **CS_ID,** [in] ECompositeInnerCSalign **InnerCSalign**)

Name	*name of the new cross-section*
Diameter	*outter diameter of the pipe cross-section*
t	*thickness of the pipe cross-section*
OutterMaterial_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 **OutterMaterial_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*
OutterMaterial_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	*outter 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*
CrossSectionUserParams	*User parameters*

Returns cross-section index if successful. If not successful, then returns an error code (errIndexOutOfBounds)

---

long **AddCustomWithUserParamsAsArray** ([in] BSTR **Name,** [in] AxisVMPolygon2dList* **ShapePolygonList,** [in] ECrossSectionProcess **Process,** [in] long **lParam,** [in] SAFEARRAY(double) **CrossSectionUserParams**)

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

Process *the manufacturing process*
lParam *User parameter*
CrossSectionUserParams *Array with user parameters (only first 10 elements of the array are used)*

Returns cross-section index if successful. If not successful, then returns an error code (errIndexOutOfBounds)

---

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

---

long **AddCustomWithUserParamsAsByteArray** ([in] BSTR **Name,** [in] AxisVMPolygon2dList* **ShapePolygonList,** [in] ECrossSectionProcess **Process,** [in] SAFEARRAY(double) **CrossSectionUserParams**)

Name *Name of the cross-section*
ShapePolygonList *Polygon list with shape of the cross-section*
lParam *User parameter*
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. If not successful, then returns an error code (errIndexOutOfBounds)

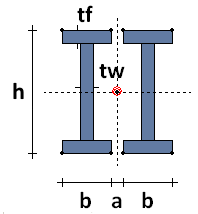
---

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**)

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

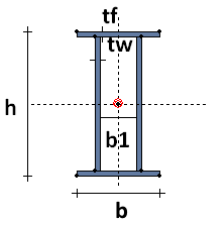


Adds a double I-shape to the model.

---

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

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

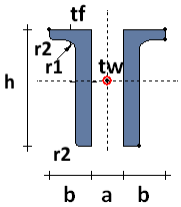


Adds a double I box shape to the model.

---

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

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



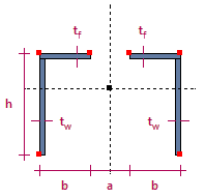
Process *the manufacturing process*

Adds a double angle shape to the model.

---

long **AddDoubleLClosed** ([in] BSTR **Name,** [in] RDoubleLClosed **Params**)

Name *name of the new cross-section*
Params *parameters of the double L closed cross-section*

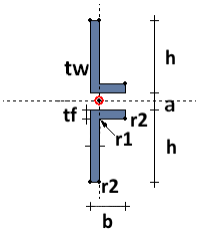


Adds a double L closed cross section to the model. A positive return value indicates success and it is the index of the newly created cross-section. A negative value means an error has occurred. The possible returned error codes are : errDatabaseNotReady, cseNotAllowedProcessForShape.

---

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*

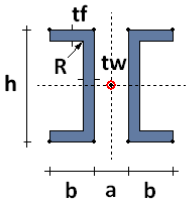


Adds a double angle shape connected at the flanges to the model.

---

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**)

Warning! *This function has become obsolete, was superseded by AddFromCatalog_V161*

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 **AddFromCatalog_V161** ([in] EcrossSectionShapeEx **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\AxisVM9\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 **AddFromCatalogTable** ([in] long **TableId,** [in] BSTR **Name**)

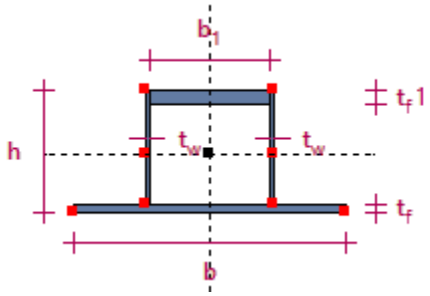
TableId *identificator for a catalog table. Obtained through IAxisVMCrossSectionTables.Item[ ].Id. To obtain IAxisVMCrossSectionTables, use AxisVMCatalog. GetAllTables*

Name *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 **AddHSQ** ([in] BSTR **Name,** [in] RCrossSectionHSQ Params)



Name *name of the new cross-section*

Params *parameters of the HSQ cross-section*

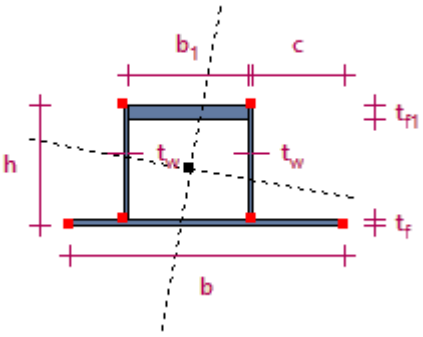
Adds a HSQ cross-section to the model. A positive return value indicates success and it is the index of the newly created cross-section. A negative value means an error has occurred. The possible returned error codes are : errDatabaseNotReady, cseNotAllowedProcessForShape.

It can trigger the following events in case of an error : ,

IAxisVMCrossSectionsEvents.Error, with the following errorcodes (errDatabaseNotReady)

---

long **AddHSQA** ([in] BSTR **Name,** [in] RCrossSectionHSQA Params)



Name *name of the new cross-section*

Params *parameters of the HSQ cross-section*

Adds a HSQA cross-section to the model. A positive return value indicates success and it is the index of the newly created cross-section. A negative value means an error has occurred. The possible returned error codes are : errDatabaseNotReady, cseNotAllowedProcessForShape.

It can trigger the following events in case of an error : ,

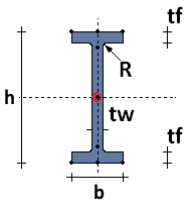
IAxisVMCrossSectionsEvents.Error, with the following errorcodes (errDatabaseNotReady)

---

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 **AddIFB** ([in] BSTR **Name,** [in] RCrossSectionIFB **Params**)

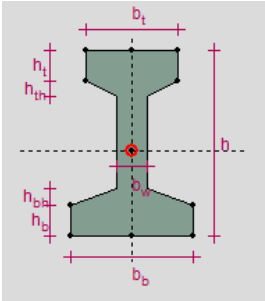
Name *name of the new cross-section*
Params *parameters of the new cross-section*

Adds an IFB 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*

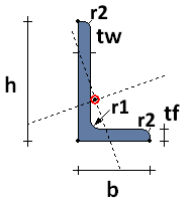


Adds a haunched I-shape to the model.

---

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*

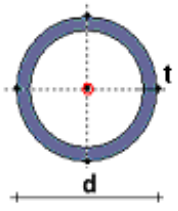


Adds an unequal angle shape to the model.

---

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*

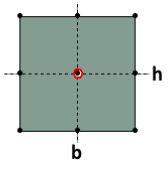


Adds a pipe to the model.

---

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*

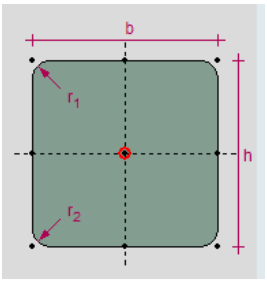


Adds a rectangular shape to the model.

---

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*

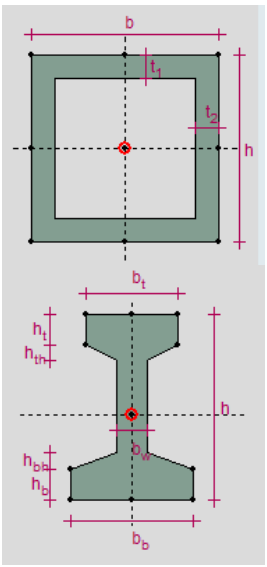


Adds a rectangular shape with rounded corners to the model.

---

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*

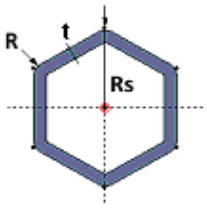


Adds a hollow rectangular shape to the model.

---

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*

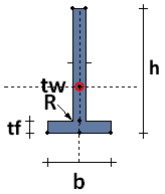


Adds a regular polygon shape to the model.

---

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 **AddSFB** ([in] BSTR **Name,** [in] RCrossSectionSFB **Params**)

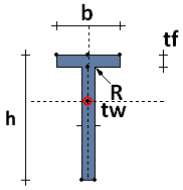
Name	*name of the new cross-section*
Params	*parameters of the new cross-section*

Adds a SFB 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*

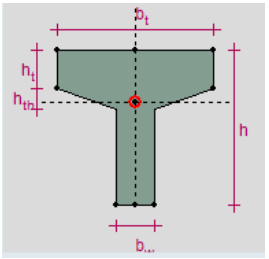


Adds a T-shape to the model.

---

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*

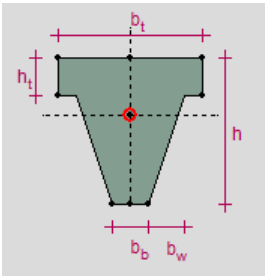


Adds a T-shape with haunched top flange to the model.

---

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*

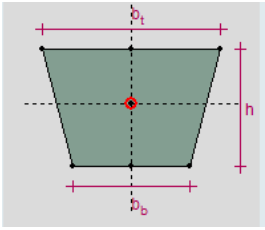


Adds a T-shape with haunched wall to the model.

---

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*



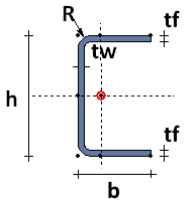
Adds a trapezoid shape to the model.

---

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*

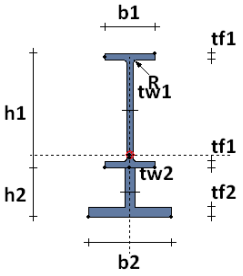


Adds a U-shape to the model.

---

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*

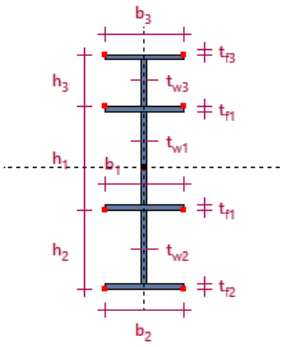


Adds a wedged I-shape to the model.

---

long **AddDoubleWedgedI** ([in] BSTR **Name,** [in] RDoubleWedgedI Params)

Name *name of the new cross-section*
Params *parameters of the double wedged I*



Adds a double wedged I-shape to the model. A positive return value indicates success and it is the index of the newly created cross-section. A negative value means an error has occurred. The possible returned error codes are : errDatabaseNotReady, cseNotAllowedProcessForShape.

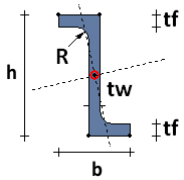
It can trigger the following events in case of an error : ,

IAxisVMCrossSectionsEvents.Error, with the following errorcodes (errDatabaseNotReady, cseNotAllowedProcessForShape)

---

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 ≤ Index ≤ 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.

---

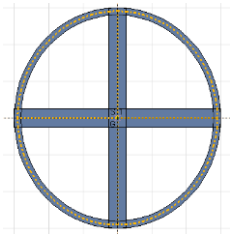
long **IndexOf** ([in] BSTR **Name**)

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).

---

long **Merge** ([in] long **IntoItem,** [in] long **FromItem,** [in] double **OffsetY,**
[in] double **OffsetZ,** [in] double **Rotation,** [in] ELongBoolean **MergeIntoNew**)



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.*

*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.*

---

long **ReplaceFromCatalog** ([in] long **Index** ,[in] ECrossSectionShape **CrossSectionShape,**
[in] BSTR **CrossSectionName**)

Index	*cross-section index to replace*
CrossSectionShape	*shape of the new cross-section*
CrossSectionName	*name of the new cross-section*

If successful, returns index, otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBounds, errNotFound, errInternalException).

---

long **ReplaceFromCatalogEx** ([in] long **Index** ,[in] EcrossSectionShapeEx **CrossSectionShape,**
[in] BSTR **CrossSectionName**)

Index	*cross-section index to replace*
CrossSectionShape	*shape of the new cross-section*
CrossSectionName	*name of the new cross-section*

If successful, returns index, otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBounds, errNotFound, errInternalException).

---

long **ReplaceFromCatalogFile** ([in] long **Index,** [in] BSTR **CatalogFileName,**
[in] BSTR **CrossSectionName**)

Index	*cross-section index to replace*
CatalogFileName	*name of the catalog file (e.g.: 'c:\Program Files\AxisVM9\EU_LEQ.SEC')*
CrossSectionName	*name of the new cross-section*

If successful, returns index, otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBounds, errNotFound, errInternalException).

---

long **ReplaceWith2IX** ([in] long **Index,** [in] BSTR **Name**, [in] RCrossSection2IX **Params**)

Index	*cross-section index to replace*
Name	*name of the new cross-section*
Params	*parameters of the new cross-section*

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

---

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

Index	*cross-section 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*

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

---

long **ReplaceWithBox** ([in] long **Index,** [in] BSTR **Name,** [in] double **h,** [in] double **b,** [in] double **tw,** [in] double **tf,** [in] double **R,** [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 wall thickness at the web*
tf *cross-section wall thickness at the flange*
R *fillet radius*
Process *the manufacturing process*

Replaces the specified cross-section with a box shape.

---

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

Index *cross-section 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*
Process *the manufacturing process*

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

---

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

Index *cross-section index to replace*
Name *name of the new cross-section*
d *diameter of the circle*

Replaces the specified cross-section with a circle shape.

---

long **ReplaceWithComposite2IX** ([in] long **Index,** [in] BSTR **Name,** [in] RCrossSectionComposite **Params**)

Index *cross-section index to replace*
Name *name of the new cross-section*
Params *parameters of the new cross-section*

Replaces the specified cross-section with a concrete encased crossed I shape.

---

long **ReplaceWithCustom** ([in] long **Index,** [in] BSTR **Name,** [in] AxisVMPolygon2dList* **ShapePolygonList,** [in] ECrossSectionProcess **Process**)

Index *cross-section index to replace*
Name *name of the new cross-section*
ShapePolygonList *polygon list describing the shape*
Process *the manufacturing process*

Replaces the cross-section with a custom shape.

---

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

Index *cross-section index to replace*
Name *name of the new cross-section*
d *circle diameter*
t *thickness*

Replaces the cross-section with a hollow circle shape to the model.

---

long **ReplaceWithCustomAndUserParams** ([in] long **Index**, [in] BSTR **Name,** [in] AxisVMPolygon2dList * **ShapePolygonList,** [in] ECrossSectionProcess **Process,**

[i/o] RCrossSectionUserParams * **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*
CrossSectionUserParams	*User parameters*

Returns index if successful. If not successful, then returns an error code (errIndexOutOfBounds)

---

long **ReplaceWithCustomAndUserParamsAsArray** ([in] long **Index**, [in] BSTR **Name,**
[in] AxisVMPolygon2dList * **ShapePolygonList,** [in] ECrossSectionProcess **Process,**
[in] long **lParam,** [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*
lParam	*User parameter*
CrossSectionUserParams	*Array with user parameters (only first 10 elements of the array are used)*

Returns index if successful. If not successful, then returns an error code (errIndexOutOfBounds)

---

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*
CrossSectionUserParams	*Array with user parameters (first element is 32bit long other 10 elements of the array are 64bit doubles)*

Returns index if successful. If not successful, then returns an error code (errIndexOutOfBounds)

---

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*

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

---

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

Index	*cross-section 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*

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

---

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

Index *cross-section 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 **ReplaceWithDoubleLClosed** ([in] long **Index,** [in] BSTR **Name,** [in] RDoubleLClosed **Params**)

Index *cross-section index to replace*
Name *name of the new cross-section*
Params *parameters of the cross-section*

Replaces the specified cross-section with a double L closed cross-section.

---

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 **ReplaceWithDoubleWedgedI** ([in] long **Index,** [in] BSTR **Name,** [in] RDoubleWedgedI Params)

Index *cross-section index to replace*
Name *name of the new cross-section*
Params *parameters of the double wedged I*

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

---

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 **ReplaceWithIFB** ([in] long **Index,** [in] BSTR **Name,** [in] RCrossSectionIFB **Params**)

Index	*cross-section index to replace*
Name	*name of the new cross-section*
Params	*parameters of the new cross-section*

Replaces the specified cross-section with an IFB shape to the model.

---

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 **hbh,** [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*
hbh	*height of the bottom haunch*
hb	*bottom flange thickness*

Replaces the specified cross-section with a haunched I-shape to the model.

---

long **ReplaceWithHSQ** ([in] long **Index,** [in] BSTR **Name,** [in] RCrossSectionHSQ Params)

Index	*cross-section index to replace*
Name	*name of the new cross-section*
Params	*parameters of the HSQ cross-section*

Replaces the specified cross-section with a HSQ cross-section.

---

long **ReplaceWithHSQA** ([in] long **Index,** [in] BSTR **Name,** [in] RCrossSectionHSQA Params)

Index	*cross-section index to replace*
Name	*name of the new cross-section*
Params	*parameters of the HSQA cross-section*

Replaces the specified cross-section with a HSQA cross-section.

---

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 **ReplaceWithSFB** ([in] long **Index,** [in] BSTR **Name,** [in] RCrossSectionSFB **Params**)

Index	*cross-section index to replace*
Name	*name of the new cross-section*
Params	*parameters of the new cross-section*

Replaces the specified cross-section with a SFB shape to the model.

---

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 ≤ Index ≤ 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

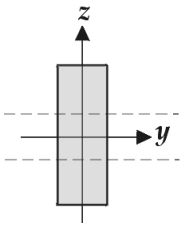
A cross-section in the model or in the catalog.

Records / structures

RStressPoint = (

double	**y**	*y coordinate of the stress point*
double	**z**	*z coordinate of the stress point*

)



RStressPointParams = (

double	**Tx_y**	$C_{Tx,y}$ *stress calculation factor for shear stress along y axis due to Saint-Venant torsion*
double	**Tx_z**	$C_{Tx,z}$ *stress calculation factor for shear stress along z axis due to Saint-Venant torsion*
double	**Vy_y**	$C_{Vy,y}$ *stress calculation factor for shear stress along y axis due to shear force along y axis*
double	**Vy_z**	$C_{Vy,z}$ *stress calculation factor for shear stress along z axis due to shear force along y axis*
double	**Vz_y**	$C_{Vz,y}$ *stress calculation factor for shear stress along y axis due to shear force along z axis*
double	**Vz_z**	$C_{Vz,z}$ *stress calculation factor for shear stress along z axis due to shear force along z axis*
double	**w**	C_{ω} *stress calculation factor for normal stress due to restrained warping*
double	**Tw_y**	$C_{T\omega,y}$ *stress calculation factor for shear stress along y axis due to restrained warping*
double	**Tw_z**	$C_{T\omega,z}$ *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 ≤ Index ≤ 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 < Index ≤ StressPointCount)*
Point *stress point coordinates*

Get stress point coordinates by index (0 < Index ≤ 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 < Index ≤ 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*

If successful, returns cross-section index, otherwise an error code (errDatabaseNotReady, errIndexOutOfBounds).

---

long **GetUserParamsAsArray** ([out] long **lParam,**
[out] SAFEARRAY(double) * **Value**)

lParam *User parameter (long)*

Value *Array with user parameters (only first 10 elements are used)*

If successful, returns cross-section index , otherwise an error code (errDatabaseNotReady, errIndexOutOfBounds).

---

long **GetUserParamsAsByteArray** ([out] SAFEARRAY(byte) * **Value**)

Value *Array with user parameters (first element is 32bit integer and another 10 elements are 64bit double)*

If successful, returns cross-section index , otherwise an error code (errDatabaseNotReady, errIndexOutOfBounds).

---

long **Mirror** ([in] double **y1,** [in] double **z1,** [in] double **y2,** [in] double **z2**)

y1 *y coordinate of the 1st point of the mirroring axis*
z1 *z coordinate of the 1st point of the mirroring axis*
y2 *y coordinate of the 2nd point of the mirroring axis*
z2 *z coordinate of the 2nd point of the mirroring axis*

Mirrors the cross-section to an axis specified by its two points.
If successful, returns the cross-section index in the list (AxisVMCrossSections), otherwise returns an error code (errReadOnly, if the cross-section is from the catalog or errDatabaseNotReady, errIndexOutOfBounds).

---

long **Move** ([in] double **dy,** [in] double **dz**)

dy *shift in y direction*
dz *shift in z direction*

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

---

long **Rotate** ([in] double **oy,** [in] double **oz,** [in] double **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*

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

---

long **SetStressPoint** ([in] long **Index,** [i/o] RStressPoint **Point**)

Index *stress point index (0 < Index ≤ StressPointCount)*
Point *stress point coordinates*

Set stress point coordinates by index (0 < Index ≤ StressPointCount). If successful, returns cross-section index, otherwise an error code (errIndexOutOfBounds).

---

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

Index *stress point index (0 < Index ≤ 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, errReadOnly).

---

long **SetUserParams** ([i/o] RCrossSectionUserParams **Value**)

Value *user parameters*

If successful, returns cross-section index, otherwise an error code (errReadOnly).

---

long **SetUserParamsAsArray** ([in] long **lParam** , [in] SAFEARRAY(double)* **Value**)

lParam *User parameter (long)*

Value *Array with user parameters (only first 10 elements are used)*

If successful, returns cross-section index, otherwise an error code (errReadOnly).

---

long **SetUserParamsAsArray_vb** (Visual Basic compatible function of **SetUserParamsAsArray**)

---

long **SetUserParamsAsByteArray** ([in] SAFEARRAY(byte) * **Value**)

Value *Array with user parameters (first element is 32bit integer and another 10 elements are 64bit double)*

If successful, returns cross-section index, otherwise an error code (errReadOnly).

---

long **SetUserParamsAsByteArray_vb** (Visual Basic compatible function of **SetUserParamsAsByteArray**)

---

long **VerifyProperties**

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

Properties

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

double **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]*

RCrossSection2IX **CrossSection2IX** • *crossed I-shape query/set field. Should be read only if CrossSectionShapeEx = csse2IX. When written, it will change the cross-section to a crossed I-shape*

RCrossSectionHSQ **CrossSectionHSQ** • *HSQ cross-section query/set field. Should be read only if CrossSectionShapeEx = csseHSQ. When written, it will change the cross-section to a HSQ cross-section*

RCrossSectionHSQA **CrossSectionHSQA** • *HSQA cross-section query/set field. Should be read only if CrossSectionShapeEx = csseHSQA. When written, it will change the cross-section to a HSQA cross-section*

RCrossSectionIFB **CrossSectionIFB** • *IFB shape query/set field. Should be read only if CrossSectionShapeEx = csseIFB. When written, it will change the cross-section to an IFB shape*

RCrossSectionSFB	**CrossSectionSFB** • *SFB shape query/set field. Should be read only if CrossSectionShapeEx = csseSFB. When written, it will change the cross-section to an SFB shape*
ECrossSectionShape	**CrossSectionShape** *cross-section shape (will be deprecated)*
ECrossSectionShapeEx	**CrossSectionShapeEx** *cross-section shape (introduced in x5)*
RDoubleWedgedI	**DoubleWedgedI** • *double wedged I query/set field. Should be read only if CrossSectionShapeEx = csseDoubleWedgedI. When written, it will change the cross-section to a DoubleWedgedI*
RDoubleLClosed	**DoubleLClosed** • *double L closed cross-section query/set field. Should be read only if CrossSectionShapeEx = csseDoubleLClosed. When written, it will change the cross-section to a double L closed one*
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]*
ELongBoolean	**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*
ECrossSectionProcess	**Process** • *manufacturing process*
double	**r1** • *1. fillet radius [m]*
double	**r2** • *2. fillet radius [m]*
double	**r3** • *3. fillet radius [m]*
AxisVMPolygon2dList	**ShapePolygonList** • *polygon list describing the shape 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
csoGroupIsForPredefinedShapes = -100002
csoGroupIsForParametricOptimization = -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
csoInvalisOptimizationType = -100032
csoNoneOfTheCrossSectionsIsValid = -100033
}

Enumerated types

enum **ECrossSectionObjectiveOfOptimization** = {

csooMinimumWeight = 0	*optimize for min. weight*
csooMinimumHeight = 1	*optimize for min. height*
csooMinimumWidth = 2}	*optimize for min. width*

Objective of optimization

enum **ECrossSectionOptimizationType** = {

csotPreDefinedShapes = 0	*optimize using cross sections in the group*
csotParametric = 1	*parametric optimization*

}
Type of optimization

Records / structures

RCSOptimizationChecks_V171 = (

ElongBoolean	**Strength**	*consider strength (currently it must always be lbTrue. If you set it to lbFalse, still lbTrue will be taken into account)*
ElongBoolean	**FlexuralBuckling**	*consider flexural buckling*
ElongBoolean	**LateralTorsionalBuckling**	*consider lateral torsional buckling*
ElongBoolean	**WebBuckling**	*consider web buckling*
ElongBoolean	**SLS**	*consider SLS*

)

RCSOptimizationParamsGeneral = (

ECrossSectionObjectiveOfOptimization	**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*

double	**MaximumEfficiency**	*max. efficiency*
ElongBoolean	**Custom**	*if lbTrue then max. number of iterations is considered*
long	**NumberOfIterations**	*max. number of iterations*
long	**Beams**	*Item index in Beams combobox, see AxisVM manual*

)

RCSParametricOptimizationParams = (

Warning! This record has become obsolete, it was superseded by RCSParametricOptimizationParams_V171

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

)

RCSParametricOptimizationParams_V171 = (

RCSOptimizationParamsGeneral	**General**	*general parameters of optimization*
ElongBoolean	**fixed_h**	*if lbTrue then height is fixed*
double	**delta_h**	*increments in height of the cross-section [m]*
ElongBoolean	**fixed_b**	*if lbTrue then width is fixed*
double	**delta_b**	*increments in width of the cross-section [m]*
ElongBoolean	**fixed_tw**	*if lbTrue then wall thickness is fixed*
double	**delta_tw**	*increments in wall thickness of the cross-section [m]*
ElongBoolean	**fixed_tf**	*if lbTrue then flange thickness is fixed*
double	**delta_tf**	*increments in flange thickness of the cross-section [m]*
double	**tw_min**	*minimum value of the web thickness (only if not fixed_tw) [m]*
double	**tw_max**	*maximum value of the web thickness (only if not fixed_tw) [m]*
double	**tf_min**	*minimum value of the flange thickness (only if not fixed_tw) [m]*
double	**tf_max**	*maximum value of the flange thickness (only if not fixed_tw) [m]*
ElongBoolean	**fixed_b2**	*if lbTrue then width b2 is fixed , only for non-symetric I*
double	**b2_min**	*min. width b2 constraint [m]*
double	**b2_max**	*max. width b2 constraint [m]*
double	**delta_b2**	*increments in width b2 of the cross-section, only for non-symetric I [m]*
ElongBoolean	**fixed_tf2**	*if lbTrue then flange thickness tf2 is fixed , only for non-symetric I*
double	**tf2_min**	*min. flange thickness tf2 constraint [m]*
double	**tf2_max**	*max. flange thickness tf2 constraint [m]*
double	**delta_tf2**	*increments in flange thickness tf2, only for non-symetric I [m]*
ElongBoolean	**fixed_a**	*if lbTrue then distance of two cross-sections is fixed only for double U*
double	**a_min**	*min. distance of two cross-sections constraint only for double U [m]*
double	**a_max**	*max. distance of two cross-sections constraint only for double U [m]*
double	**delta_a**	*increments in distance of two cross-sections, only for double U [m]*

)

RCSOptimizationResultsParametric = (

double	**OptimizationEfficiency**	*Optimization efficiency []*
double	**Efficiency**	*Efficiency []*
double	**M**	*Weight per length [kg/m]*
double	**DeltaM**	*Weight difference [%]*
double	**b**	*Width [m]*
double	**h**	*Height [m]*
double	**tw**	*Web thickness [m]*
double	**tf**	*Flange thickness [m]*
double	**b2**	*Width 2 (only for non-symetric I) [m]*
double	**tf2**	*Flange thickness 2 (only for non-symetric I) [m]*
double	**a**	*Distance between the two cross-sections (only for double U) [m]*

)

RCSOptimizationResultsPredefinedShapes = (

double	**OptimizationEfficiency**	*Optimization efficiency []*

double	**Efficiency**	*Efficiency []*
double	**M**	*Weight per length [kg/m]*
double	**DeltaM**	*Weight difference [%]*
long	**GroupCrossSection**	*Group index*
	)	

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* <= **CrossSectionIndex** <= *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	*index of the group*
CrossSectionName	*name of the cross-section used in catalog (must be the same shape as shape used in the group)*

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.*

---

long **AddCrossSectionFromDialog** ([in] long **GroupIndex**)

GroupIndex	*index of the group*

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.*

---

long **DeleteGroup** ([in] long **GroupIndex**)

GroupIndex	*index of the group*

Delete a optimization group. If successful, returns GroupIndex*, otherwise an error code.*

---

long **DeleteGroupCrossSection** ([in] long **GroupIndex**, [in] long **GroupCrossSectionIndex**)

GroupIndex	*index of the group*
GroupCrossSectionIndex	index of the cross-section in the group

Delete a group cross-section from optimization group. If successful, returns GroupCrossSectionIndex*, otherwise an error code.*

---

long **GetGroupCrossSections** ([in] long **GroupIndex**, [out] SAFEARRAY(long) * **ModelCrossSectionIDs**, [out] SAFEARRAY(BSTR) * **TableNames**, [out] SAFEARRAY(BSTR) * **CrossSectionNames**)

GroupIndex	*index of the group*
ModelCrossSectionIDs	*index of cross-section in the model*
TableNames	*table name of cross-section in the group*
CrossSectionNames	*name of the cross-section in the group*

Get group cross sections of the optimization group. If successful, returns length of each array, *otherwise an error code.*

---

long **GetOptimizationChecks** ([in] long **GroupIndex,** [out] ElongBoolean **Strength,**
[out] ElongBoolean **FlexuralBuckling,** [out] ElongBoolean **LateralTorsionalBuckling,**
[out] ElongBoolean **WebBuckling**)

Warning! *This function has become obsolete, was superseded by GetOptimizationChecks_V171*

GroupIndex	*index of the group*
Strength	strength considered in optimization
FlexuralBuckling	flexural buckling considered in optimization
LateralTorsionalBuckling	lateral torsional buckling considered in o optimization ptimisation
WebBuckling	web buckling considered in optimization

Get optimization checks of the optimization group. If successful, returns GroupIndex, *otherwise an error code.*

---

long **GetOptimizationChecks_V171** ([in] long **GroupIndex,** [out] RCSOptimizationChecks_V171 **Checks**)

Checks	*check description record*

Get optimization checks of the optimization group. If successful, returns GroupIndex, *otherwise an error code.*

---

long **GetParametersForPredefinedShapes** ([in] long **GroupIndex,**
[i/o] RCSOptimizationParamsGeneral **GeneralParams**)

GroupIndex	*index of the group*
GeneralParams	general parameters for optimization of pre-defined shapes

Get optimization parameters of the optimization group. If successful, returns GroupIndex, *otherwise an error code.*

---

long **GetParametersForParametricOptimization** ([in] long **GroupIndex,**
[i/o] RCSParametricOptimizationParams **ParametricParams**)

GroupIndex	*index of the group*
ParametricParams	parameters for parametric optimization

Get optimization parameters of the optimization group. If successful, returns GroupIndex, *otherwise an error code.*

---

long **GetParametersForParametricOptimization_V171** ([in] long **GroupIndex,**
[i/o] RCSParametricOptimizationParams_V171 **ParametricParams**)

GroupIndex	*index of the group*
ParametricParams	parameters for parametric optimization

Get optimization parameters of the optimization group. If successful, returns GroupIndex, *otherwise an error code.*

---

long **GetMemberDesignIDs** ([in] long **GroupIndex,** [out] SAFEARRAY(long) * **MemberDesignIDs**)

GroupIndex	*index of the group*
MemberDesignIDs	*indexes of SteelDesignMembers or TimberDesignMembers depends on the used property of IAxisVMModel*

Get design members indexes of the optimization group. If successful, returns GroupIndex, *otherwise an error code.*

---

long **GetParametricShapeOptimizationResultsByLoadCaseId** ([in] long **GroupIndex,**
[in] long **LoadCaseID,** [in] long **LoadLevel**, [in] EAnalysisType **AnalysisType**,
[i/o] RCSOptimizationResultsParametric **ParametricResults**)

GroupIndex	*index of the group*
LoadCaseID	*index of the load case*
LoadLevel	*load level*

AnalysisType *type of analysis*
ParametricResults *results of parametric optimization*

Get parametric optimization results of the optimization group. If successful, returns GroupIndex*, otherwise an error code.*

---

long **GetParametricShapeOptimizationResultsByLoadCombinationId** ([in] long **GroupIndex,** [in] long **LoadCombinationId,** [in] long **LoadLevel**, [in] EAnalysisType **AnalysisType**, [i/o] RCSOptimizationResultsParametric **ParametricResults**)

GroupIndex *index of the group*
LoadCombinationId *index of the load combination*
LoadLevel *load level*
AnalysisType *type of analysis*
ParametricResults results of parametric optimization

Get parametric optimization results of the optimization group. If successful, returns GroupIndex*, otherwise an error code.*

---

long **GetEnvelopeParametricShapeOptimizationResults** ([in] long **GroupIndex,** [in] long **EnvelopeUID,** [in] EAnalysisType **AnalysisType**, [i/o] RCSOptimizationResultsParametric **ParametricResults**)

GroupIndex *index of the group*
EnvelopeUID *unique index of the envelope*
AnalysisType *type of analysis*
ParametricResults results of parametric optimization

Get parametric optimization results of the optimization group. If successful, returns GroupIndex*, otherwise an error code.*

---

long **GetCriticalParametricShapeOptimizationResults** ([in] long **GroupIndex,** [in] ECombinationType **CombinationType,** [in] EAnalysisType **AnalysisType**, [i/o] RCSOptimizationResultsParametric **ParametricResults**)

GroupIndex *index of the group*
CombinationType *combination type*
AnalysisType *type of analysis*
ParametricResults results of parametric optimization

Get parametric optimization results of the optimization group. If successful, returns GroupIndex*, otherwise an error code.*

---

long **GetPredefinedShapesOptimizationResultsByLoadCaseId** ([in] long **GroupIndex,** [in] long **LoadCaseID,** [in] long **LoadLevel**, [in] EAnalysisType **AnalysisType**, [out] SAFEARRAY(RCSOptimizationResultsPredefinedShapes) * **PredefinedShapesResults**)

GroupIndex *index of the group*
LoadCaseID *index of the load case*
LoadLevel *load level*
AnalysisType *type of analysis*
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 **GetPredefinedShapesOptimizationResultsByLoadCombinationId** ([in] long **GroupIndex,** [in] long **LoadCombinationId,** [in] long **LoadLevel**, [in] EAnalysisType **AnalysisType**, [out] SAFEARRAY(RCSOptimizationResultsPredefinedShapes) * **PredefinedShapesResults**)

GroupIndex *index of the group*
LoadCombinationId *index of the load combination*
LoadLevel *load level*

AnalysisType *type of analysis*

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 **GetEnvelopePredefinedShapesOptimizationResults** ([in] long **GroupIndex,**
[in] long **EnvelopeUID,** [in] EAnalysisType **AnalysisType**,
[out] SAFEARRAY(RCSOptimizationResultsPredefinedShapes) * **PredefinedShapesResults**)

GroupIndex *index of the group*

EnvelopeUID *unique index of the envelope*

AnalysisType *type of analysis*

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 **GetCriticalPredefinedShapesOptimizationResults** ([in] long **GroupIndex,**
[in] ECombinationType **CombinationType,** [in] EAnalysisType **AnalysisType**,
[out] SAFEARRAY(RCSOptimizationResultsPredefinedShapes) * **PredefinedShapesResults**)

GroupIndex *index of the group*

CombinationType *combination type*

AnalysisType *type of analysis*

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 **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 *index of the group*

GroupCrossSectionIndex index of the cross-section in the group

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.

---

long **ReplaceCrossSectionToModelCrossSection** ([in] long **GroupIndex,**
[in] long **ModelCrossSectionIndex**)

GroupIndex *index of the group*

ModelCrossSectionIndex index of the cross-section in the model

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.

---

long **SetOptimizationChecks** ([in] long **GroupIndex,** [in] ElongBoolean **Strength,**
[in] ElongBoolean **FlexuralBuckling,** [in] ElongBoolean **LateralTorsionalBuckling,**
[in] ElongBoolean **WebBuckling**)

Warning! *This function has become obsolete, was superseded by SetOptimizationChecks_V171*

GroupIndex *index of the group*

Strength *strength considered in optimization*

FlexuralBuckling *flexural buckling considered in optimization*

LateralTorsionalBuckling *lateral torsional buckling considered in o optimization ptimisation*

WebBuckling *web buckling considered in optimization*

Set optimization checks of the optimization group. If successful, returns GroupIndex*, otherwise an error code.*

---

long **SetOptimizationChecks_V171** ([in] long **GroupIndex,** [i/o] RCSOptimizationChecks_V171 **Checks**)

Checks *check description record*

Set optimization checks of the optimization group. If successful, returns GroupIndex, *otherwise an error code.*

---

long **SetParametersForPredefinedShapes** ([in] long **GroupIndex,**
[i/o] RCSOptimizationParamsGeneral **GeneralParams**)

GroupIndex *index of the group*

GeneralParams *general parameters for optimization of pre-defined shapes*

Set optimization parameters of the optimization group. If successful, returns GroupIndex, *otherwise an error code.*

---

long **SetParametersForParametricOptimization** ([in] long **GroupIndex,**
[i/o] RCSParametricOptimizationParams **ParametricParams**)

GroupIndex *index of the group*

ParametricParams parameters for parametric optimization

Get optimization parameters of the optimization group. If successful, returns GroupIndex, *otherwise an error code.*

---

long **SetMemberDesignIDs** ([in] long **GroupIndex,** [in] SAFEARRAY(long) * **MemberDesignIDs**)

GroupIndex *index of the group*

SteelMemberDesignIDs *indexes of SteelDesignMembers or TimberDesignMembers depends on the used property of IAxisVMModel*

Set steel design members indexes of the optimization group. If successful, returns GroupIndex, *otherwise an error code.*

---

long **SetMemberDesignIDs_vb** ([in] long **GroupIndex,** [in] SAFEARRAY(long) * **MemberDesignIDs**)

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

Properties

EAnalysisType	**AnalysisType** • *Get or set the type of analysis*
ELongBoolean	**CallMainProgress** • *Get or set whether MainProgress event should be called*
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	**GroupCount** *Get number of optimization groups*
long	**GroupCrossSectionCount** [long **Index**] *Get number of cross-sections in the group*
ECrossSectionShape	**GroupCrossSectionShape** [long **Index**] *Get cross-section shape of the group (will be deprcated with CrossSectionShapeEx*
ECrossSectionShapeEx	**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 t load case index* *(0 < LoadCaseId ≤ AxisVMLoadCases.Count)*
long	**LoadCombinationId** • *Get or set the load combination index* *(0 < LoadCaseId ≤ AxisVMLoadCombinations.Count)*
long	**LoadLevel** • *Get or set the load level (increment) index*
ECrossSectionOptimizationType	**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 created in IAxisVMCustomPartFolder

Enumerated types

enum **ESelectMode** = {

smSelect = 0x0,	*select elements of the part*
smDeselect = 0x1,	*unselect elements of the part*
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 **ECustomPartsError** = {

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

RPartItem = (

EPartItemType	**ItemType**	*Type of the element in the part*
long	**Id**	*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, errIndexOutOfBounds)

---

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, errIndexOutOfBounds)

---

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

PartID *index of the custom part*

Delete custom part. Returns PartID if successful, otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBounds)

IAxisVMCustomPartFolder **GetCustomPartFolderByPath** ([in] BSTR **FullPath**)

FullPath *Full path of the custom part folder*

Get custom part folder from the full path. Returns custom part folder if successful, otherwise null.

---

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

PartID *index of the custom part*
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 **GetPartItemsByUID** ([in] long **PartUID,** [i/o] SAFEARRAY(RPartItem)*** PartItems**)

PartUID *unique index of the custom part*
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 **ModifyPart** ([in] long **PartID,** [in] SAFEARRAY(RPartItem)*** PartItems**)

PartID *index of the custom part*
PartItems *custom part items*

Modify part items in the existing custom part. Returns number of part items if successful, otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBounds)

---

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

PartID *index of the custom part*

Replace part items in the existing custom part with selected element. Returns PartID if successful, otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBounds)

---

long **SelectPartItems** ([in] long **PartID,** [in] ESelectMode **SelectMode**)

PartID *index of the custom part*
SelectMode *Selection mode*

Select part items of the existing custom part bassed on selection mode. Returns PartID if successful, otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBounds)

**** Call Refresh function afterwards if not called between functions BeginUpdate and EndUpdate*

Properties

AxisVMAttachments* **Attachments** *Get the attachments interface*

long **Count**
Get total number of custom parts in the model.

ELongBoolean **IsCustomPart** [long **PartUID**] • *Is lbTrue if it is a custom part*
PartUID *Unique index of the custom part*

long **IndexOfUID** [long **PartUID**] • *Get index of the custom part*
PartUID *Unique index of the custom part*

BSTR **Name** [long **PartID**] • *Get or set name of the custom part*
PartID *index of the custom part*

BSTR **FullName** [long **PartID**] • *Get or set full name of the custom part*
PartID *index of the custom part*

IAxisVMCustomPartFolder **RootFolder** *Name of the custom part root folder*

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,	*delete parts in folder*
pcMoveToUpperFolder = 0x1,	*move parts to the upper folder*
pcMoveToRootFolder = 0x2 }	*move parts to the root folder*

Part control after subfolder delete

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 *index of the custom part subfolder (0 < SubFolderIndex ≤ SubFolderCount)*

PartControl *Part control after subfolder delete*

Deletes custom part subfolder. Returns SubFolderIndex if successful, otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBounds)

---

long **ModifyPart** ([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*

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)

---

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

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

Replace part items of the existing custom part with selected elements. Returns custom part index if successful, otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBounds)

---

long **RenamePart** ([in] long **PartIndex,** [in] BSTR **Name**)

PartIndex *index of the custom part in the current custom* part folder

Name *name of the custom part*

Rename custom part. Returns custom part index if successful, otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBounds)

---

long **RenameSubFolder** ([in] long **SubFolderIndex,** [in] BSTR **Name**)

SubFolderIndex *index of the custom part subfolder (0 < SubFolderIndex ≤ SubFolderCount)*

Name *name of the custom part subfolder*

Rename custom part subfolder. Returns SubFolderIndex if successful, otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBounds)

---

long **SelectPartItems** ([in] long **PartIndex,** [in] ESelectMode **SelectMode**)

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

SelectMode *Selection mode*

Select part items of the existing custom part bassed on selection mode. Returns index of the custom part if successful, otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBounds)

NOTE: Call Refresh function afterwards if not called between functions BeginUpdate and EndUpdate

---

Properties

ELongBoolean **IsRootFolder** *Is lbTrue if the custom part folder is the root folder*

BSTR **Name** • *Get or set name of the custom part folder*

IAxisVMCustomPartFolder **ParentFolder** *Get the custom part folder's parent folder*

long **PartCount**
Get number of custom parts in the current custom part folder

long **PartId** [long **PartIndex**] *Get PartID of the custom part, used in IAxisVMCustomParts*

PartIndex *index of the custom part in the current custom part folder*

long **PartUID** [long **PartIndex**]
Get unique index of the custom part which remains the same while exists in the model

PartIndex *index of the custom part in the current custom part folder*

BSTR **Path** *Path of the custom part folder, folders separated with TAB character*

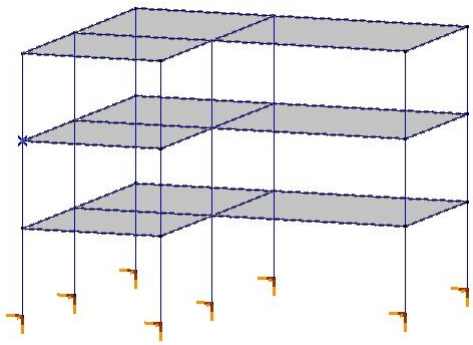
IAxisVMCustomPartFolder **SubFolder** [long **SubFolderIndex**] *Get the custom part subfolder*

SubFolderIndex *index of the custom part subfolder (0 < SubFolderIndex ≤ SubFolderCount)*

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*

If successful returns number of diaphragms, otherwise returns an error code (errDatabaseNotReady, dpheIllegalDOFValue, dpheLineListIsEmpty, dpheLineIndexOutOfBounds)

---

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

---

long **AddSelectedLines** ([in] long **Dof**)

Dof *Degree of freedom see: EDegreeOfFreedom*

If successful returns number of diaphragms, otherwise returns an error code (errDatabaseNotReady, dpheIllegalDOFValue, dpheNoLinesAreSelected)

---

long **Clear**

If successful returns number of diaphragms before delete, otherwise returns an error code (errDatabaseNotReady)

---

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

Index *index of the diaphragm, 1 ≤ Index ≤ Count*

If successful returns Index, otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBounds)

---

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

Index *index of the diaphragm, 1 ≤ Index ≤ Count*

LineIds *Array with LineIDs*

If successful returns number of lines in diaphragm with index Index, otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBounds)

---

long **GetDOF** ([in] long **Index,** [out] long **Dof**)

Index *index of the diaphragm, 1 ≤ Index ≤ Count*

Dof *Degree of freedom see: EDegreeOfFreedom*

If successful returns Index, otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBounds)

---

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,  (NOT SUPPORTED YET)

dsLevelCircle= 0x10,  (NOT SUPPORTED YET)

dsLevelCircleExt= 0x11,  (NOT SUPPORTED YET)

dsLevelCross= 0x12,  (NOT SUPPORTED YET)

dsLevelCrossExt= 0x13,  (NOT SUPPORTED YET)

dsElevCorner= 0x14,  (NOT SUPPORTED YET)

dsElevTriangleExt= 0x15,  (NOT SUPPORTED YET)

dsElevTriangleRight= 0x16, (NOT SUPPORTED YET)

dsElevTriangleTop= 0x17,  (NOT SUPPORTED YET)

dsElevTriangleTopRight= 0x18,  (NOT SUPPORTED YET)

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

RDimensionLineParameters= (

ELongBoolean	**Orthogonal**	*orthogonal*
EGlobalWorkplaneType	**Plane**	*plane*
EAxis	**OrthoDirection**	*orthogonal direction*
long	**NodeID1**	*start node's index*
long	**NodeID2**	*endnode's index*
long	**LayerID**	*layer's index*
double	**LabelDistance**	*distance*
ELongBoolean	**LabelInside**	*label is inside/outside*
EDimensionLabelOrientation	**LabelOrientation**	*label's orientation*
EDimensionStyle	**TickMarkStyle**	*dimension's style*
ELongBoolean	**DisplayUnit**	*display unit*

)

RTextBoxParameters= (

long	**NodeID**	*node's index*
long	**LayerID**	*layer's index*
RPoint3d	**Position**	*position*
EDimensionLabelOrientation	**LabelOrientation**	*orientation*
EDimensionStyle	**TextBoxStyle**	*style*

)

Functions

long **AddDimensionLine** ([in] BSTR **Text,**
[i/o] RDimensionLineParameters **DimensionLineParameters**)

Text *text*
Note: the text must be started with MeasurementCharacter (‡).
Example: "‡ wall" → it gives e.g. "4.0 wall"

DimensionLineParameters *dimension line's parameters*

Adds a new dimension as dimension line. If successful, returns dimension's index, otherwise an error code (EDimensionsError).

---

long **AddTextBox** ([in] BSTR **Text,** [i/o] RTextBoxParameters **TextBoxParameters**)

Text *text*

TextBoxParameters *text box's parameters*

Adds a new dimension as a text box. If successful, returns dimension's index, otherwise an error code (EDimensionsError).

---

long **GetDimensionLine** ([in] long **Index,** [out] BSTR **Text,**
[i/o] RDimensionLineParameters **DimensionLineParameters**)

Index *dimension line's index*

Text *text*

DimensionLineParameters *dimension line's parameters*

Gets dimension line's text and parameters. If successful, returns dimension's index, otherwise an error code (EDimensionsError).

---

long **GetParams** ([in] long **Index,** [out] SAFEARRAY(byte) **Params**)

Index *dimension's index*

Params *parameters*

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!

---

long **SetParams** ([in] long **Index,** [in] SAFEARRAY(byte) **Params**)

Index *dimension line's index*

Params *parameters*

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!

---

long **GetText** ([in] long **Index,** [out] BSTR **Text**)

Index *dimension's index*

Text *text*

Sets dimension's text. If successful, returns dimension's index, otherwise an error code (EDimensionsError).

---

long **SetText** ([in] long **Index,** [in] BSTR **Text**)

Index *dimension's index*

Params *text*

Sets dimension's text. If successful, returns dimension's index, otherwise an error code (EDimensionsError).

---

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

Index *dimension's index*

Deletes dimension. If successful, returns dimension's index, otherwise an error code (EDimensionsError).

Properties

long	**Count** *Get number of dimensions*
EDimensionType	**DimensionType** [long **Index**] *Get dimension's type by index*
	Index *dimension's index*
BSTR	**MeasurementCharacter** *(‡).*

IAxisVMDomains

Domains of the model

Enumerated types

enum **ELineNonlinearity** = {

lnlTensionAndCompression = 0x0,	*linear behaviour*
lnlTensionOnly = 0x1,	*tension only*
lnlCompressionOnly = 0x2 }	*compression only*

Types of nonlinear behaviour.

enum **EMeshType** = {

mtAdaptive = 0x0,	*adaptive mesh*
mtUniform = 0x1 }	*unofrm mesh*

Types of finite element mesh.

enum **EMeshGeometryType** = {

mgtTriangle = 0,	*triangle mesh*
mgtQuad = 1,	*quad mesh*
mgtMixedQuadTriangle = 2 }	*mixed triang and quad mesh*

Geometry type of finite element mesh.

enum **ESurfaceCharacteristics** = {

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

*Nonlinear surface characteristics **(not used)**.*

enum **ESurfaceType** = {

stHole = 0x0,	*hole*
stMembraneStress = 0x1,	*membrane (plane stress)*
stMembraneStrain = 0x2,	*membrane (plane strain)*
stPlate = 0x3,	*plate*
stShell = 0x4 }	*shell*

Types of surface elements.

enum **EDomainCompRibEccType** = {

dcret_top = 1,	*top alignement*
dcret_midplane = 2,	*midplane alignement*
dcret_bottom = 3,	*bottom alignement*
dcret_custom = 4 }	*user specified offset*

rib alignement to the domain

enum **EDomainVariableThicknessType** = {

dvtvNone = 0x0,	*constant thickness*
dvtvOneDirection = 0x1,	*varies in one direction*
dvtvTwoDirections = 0x2 }	*varies in two directions*

Type of variable thickness

enum **EDomainExcentricityType** = {

detNone = 0x0,	*no eccentricity*
detConstant = 0x1,	*constant eccentricity*
detOneDirection = 0x2,	*eccentricity in one direction*

detTwoDirections = 0x3,	*eccentricity in two directions*
detTopSurface = 0x4,	*eccentricity by alignment to top surfacce*
detBottomSurface = 0x5 }	*eccentricity by alignment to bottom surfacce*

Types of eccentricity

enum **EHollowHoleType**= {

hht_Circular = 1,	*circular hole*
hht_Rectangular = 2,	*rectangular hole*

hole geometry for hollow domains

enum **EXLAMTopLayerDirection** = {

xtldLocalX = 0x0,	*in local x direction of the domain*
xtldLocalY = 0x1,	*in local y direction of the domain*

Grain direction of the XLAM panel's top layer

enum **EXLAMServiceClass** = *{*

xlscClass1 = 1,	*service class 1*
xlscClass2 = 2,	*service class 2*

XLAM service class

enum **EXYDirection** = *{*

xyd_x = 1,	*axis in x direction*
xyd_y = 2,	*axis in y direction*

Axis direction

Error codes

enum **EDomainsError** = {

deEmptyContour = = –100001,	*line list is empty*
deMoreThanOneContourFound = –100002,	*more than one contour can be identified in the line list*
deEmptyHole = = –100003,	*line list is empty (when defining a hole)*
deMoreThanOneHoleFound = = –100004,	*more than one hole contour can be identified in the line list*
doeCOMError = -100005,	*internal COM server error*
deConcreteIdIndexOutOfBounds = -100006,	*Concrete's material index is out of bounds*
deRebarSteelGradeIdIndexOutOfBounds = -100007,	*rebar's material index index is out of bounds*
deThicknessMustBePositive = -100008,	*domain thickness must be a positive number*
deRebarPosMustBePositive = -100009,	*rebar position must be a positive number*
dePhiMustBePositiveOrZero = -100010,	*Phi must be a positive number*
deNuMustBePositiveOrZero = -100011,	*Nu must be a positive number*
deTauaMustBePositiveOrZero = -100012,	*Tau_a must be a positive number*
deAggregateSizeMustBePositive = -100013,	*aggregate size must be a positive number*
deStoreyIdOutOfBounds = -100014,	*storey index is out of bounds*
deReinforcementParametersNotExists = -100015**,**	*reinforcement parameters do not exists*
deHoleNotInDomainPlane= -100016**,**	*hole is not in the domain's plane*
dePropertyNotValidForThisDomainSurfaceType = -100017,	*StiffnessReduction property can return this error*
defseMustBePositive = -100018 ,	*seismic load factor fse must be a positive value*
deParametersRecordNotValidForUsedDesignCode = -100019	*used parameter record type is not valid for current design code*
deConcreteCoverMustBePositive = -100020,	*concrete cover must be a positive value*
deRebarDiameterMustBePositive = -100021,	*Rebar diameter must be a positive value*
deInvalidGroupID = -100022 ,	*GroupID is not valid*
deXLMmoduleNotAvailable = -100023 ,	*XLM module is not available*
deEnvironmentClassNotValidForUsedDesignCode = -100024,	*Environment Class is not valid for used design code*
deDomainIsNotMeshed = - 100025 ,	*the domain is not meshed*
deAlphaVRdmaxIsInvalid = - 100026 ,	*the angle of shear reinforcement is invalid*
deThetaVRdmaxIsInvalid = - 100027 ,	*the angle of shear crack is invalid*
deShrinkageEpsMustBePositive = - 100028 ,	*shrinkage strain must be positive*
deRCNonlinearSurfTypeIsInvalid = - 100029	*nonlinear surface type is invalid*
deAlphaAngleIsInvalid = - 100030 ,	*the angle of ξ reinforcement direction is invalid*
deBetaAngleIsInvalid = - 100031,	*the angle between ξ and η reinf. directions is invalid*
de_k_torsionIsInvalid = - 100032 ,	*k_torsion should be <= 0.1 and < 1*
de_k_shearIsInvalid = - 100033 ,	*k_shear should be <= 0.1 and < 1*
de_k_bendingIsInvalid = - 100034 ,	*k_bending should be <= 0.1 and < 1*
deCustomStiffnessMatrixUndefined = - 100035 ,	*custom stiffness matrix is not defined*

deCustomStiffnessMatrixNonSymetric = - 100036 , *custom stiffness matrix is not symmetric*

deCustomStiffnessMatrixNonPositiveDefine = - 100037, *custom stiffness matrix is not positive define*

deLimitingCrackWidthIsInvalid = - 100038 , *the limiting value for crack width is invalid*

dePolyLineIsNotContinuous = - 100039, *the selected polyline is not continuous*

deLineDoesNotReachDomainEdge = - 100040 , *the line does not reach the edge of selected domain*

deNoSelectedLine = - 100041 , *no line is selected*

deNoSelectedLineAndDomain = - 100042 , *no line nor domain is selected*

deMaterialIndex = - 100043 , *MaterialIndex must be : 1 ≤ MaterialIndex ≤ AxisVMMaterials.Count*

deReferenceIndexOutOfBounds = - 100044 , *reference index must be : 0 ≤ ReferenceIndex ≤ AxisVMReferences.Count*

deDomainInvalidType = - 100045 , *an enumerated type hase violated the valid range for that type*

deElasticFoundationNegative = - 100046 , *Elastic foundation cannot be negative*

deInvalidCharacteristics = -100047 , *Characteristics is out of valid range*

deInvalidStiffnessReduction = -100048 , *stiffness reduction should be > 0.00 and <= 1.00*

deStiffnessReductionNotAllowed = -100049 , *design code doesn't allow for stiffness reduction*

deInvalidStiffnessReductionMat = -100050, *domain's material doesn't allow for stiffness reduction*

deInvalidReinfParamForTrapezoidal = -100051 , *reinforcement parameters are incompatible with the geometry of the trapezoidal domain*

deInvalidHollowCoreDirection = -100052 , *RDomainHollowCore.Direction must be xyd_x or xyd_y*

deInvalidHollowCoreHoletype = -100053 , *RDomainHollowCore.HoleType must be hht_Circular or hht_Rectangular*

deHollowCoreDMustBePositive = -100054 , *RDomainHollowCore.d must be positive*

deHollowCoreFiMustBePositive = -100055 , *RDomainHollowCore.fi must be positive*

deHollowCoreBMustBePositive = -100056 , *RDomainHollowCore.b must be positive*

deHollowCoreHMustBePositive = -100057 , *RDomainHollowCore.h must be positive*

deTrapezoidalInvalidDirection = -100058 , *RDomainTrapezoidal.Direction must be xyd_x or xyd_y*

deTrapezoidalInvalidMaterial = -100059 , *RDomainTrapezoidal.PlateMaterial must be 1 ≤ PlateMaterial ≤ AxisVMMaterials.Count, RDomainTrapezoidal.InfillMaterial must be 1 ≤ InfillMaterial ≤ AxisVMMaterials.Count, RDomainTrapezoidal.PlateMaterial must be steel, RDomainTrapezoidal. InfillMaterial must be concrete*

deTrapezoidal_h_MustBePositive = -100060 , *this error code is no longer valid, instead deTrapezoidal_h_MustNotBeNegative is returned (h = 0.00 is a valid value, it means that there is no infill)*

deTrapezoidal_t_MustBePositive = -100061 , *RDomainTrapezoidal.t must be positive*

deTrapezoidal_v_MustBePositive = -100062 , *RDomainTrapezoidal.v must be positive*

deTrapezoidal_d_MustBePositive = -100063 , *RDomainTrapezoidal.d must be positive*

deTrapezoidal_b_MustBePositive = -100064 , *RDomainTrapezoidal.b must be positive*

deTrapezoidal_w_MustBePositive = -100065 , *RDomainTrapezoidal.w must be positive*

deTrapezoidal_p_MustBePositive = -100066 , *RDomainTrapezoidal.p must be positive*

deTrapezoidal_eta_OutOfRange = -100067 , *RDomainTrapezoidal.eta must be >= 0.00 and <= 1.00*

deTrapezoidal_ht_Mismatch = -100068 , *conflicting RDomainTrapezoidal.h and RDomainTrapezoidal.t*

deTrapezoidal_dbw_Mismatch = -100069 *conflicting RDomainTrapezoidal.d, RDomainTrapezoidal.b and RDomainTrapezoidal.w*

deCompositeRibInvalidMaterial = -100070 , *RDomainCompositeRib. RibMaterialmust be 1 ≤ RibMaterial≤ AxisVMMaterials.Count*

deCompositeRibInvalidCrossSection = -100071 , *RDomainCompositeRib. CrossSection be 1 ≤ CrossSection ≤ AxisVMCrossSections.Count*

deCompositeRibInvalidDirection = -100072 , *RDomainCompositeRib.Direction must be a valid EXYDirection value*

deCompositeRib_d_TooSmall = -100073 , *distance between ribs is too small with respect to the cross-section parameters*

deCompositeRibInvalidEccType = -100074 , *RDomainCompositeRib.EccType must be a valid EdomainCompRibEccType value*

deNotARibbedDomain = -100075 *domain has to be a ribbed one already*

deRibbedNotUniformThickness = -100076 , *the domain must be of uniform thickness (i.e. cannot be of variable thickness)*

deRibbedRibHTooSmall = -100077 *rib height must be greater than domain thickness*

deRibbed_w_MustBePositive = -100078 , *rib width must be a positive number*

deRibbed_OverlappingRibs = -100079 , *this rib distance and rib width combination leads to overlapping*

deRibbed_ExcTooLarge = -100080 ,	*with this large excentricity the rib gets outside of the domain*
deRibbed_EC_ThicknessTooSmall = -100081 ,	*only for design code EC, according to EN 1992-1-1, 5.3.1. (6) the domain thickness is too small for a ribbed domain*
deRibbed_EC_h_TooLarge = -100082 ,	*only for design code EC, according to EN 1992-1-1, 5.3.1. (6) the height of the rib is too large*
deRibbed_EC_d_TooLarge = -100083 ,	*only for design code EC, according to EN 1992-1-1, 5.3.1. (6) the distance between ribs is too large*
deXLAM_NotTimber = -100084 ,	*only timber domains are allowed*
deXLAM_DesignCodeConflict = -100085 ,	*timber type is not from the active national code*
deXLAM_InvalidKDEF = -100086,	*kDef is not within the valid range (0.00 – 1.00)*
deTrapezoidal_h_MustNotBeNegative = -100087 }	*RDomainTrapezoidal.h cannot be negative (but 0.00 is a valid value)*

Errors during domain or hole definition

Records / structures

RDomainMeshParameters = (

Warning! *This record was superseded by RDomainMeshParameters_V171*

double	**MeshSize**	*average mesh size [m]*
EMeshType	**MeshType**	*type of the finite element mesh*
ELongBoolean	**IsFitToPointLoad**	*fit mesh to point loads*
double	**FitToPointLoad**	*minimum load to fit the mesh to the load point [kN]*
ELongBoolean	**IsFitToLineLoad**	*fit mesh to line loads*
double	**FitToLineLoad**	*minimum load to fit the mesh to the load line [kN/m]*
ELongBoolean	**IsFitToSurfaceLoad**	*fit mesh to surface loads*
double	**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*

)

RDomainMeshParameters_V171 = (

double	**MeshSize**	*average mesh size [m]*
EMeshType	**MeshType**	*type of the finite element mesh*
ELongBoolean	**IsFitToPointLoad**	*fit mesh to point loads*
double	**FitToPointLoad**	*minimum load to fit the mesh to the load point [kN]*
ELongBoolean	**IsFitToLineLoad**	*fit mesh to line loads*
double	**FitToLineLoad**	*minimum load to fit the mesh to the load line [kN/m]*
ELongBoolean	**IsFitToSurfaceLoad**	*fit mesh to surface loads*
double	**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*
ELongBoolean	**FitToColumnhead**	*fit mesh to column heads*
ELongBoolean	**MeshOnlyUnmeshed**	*taken into account only during mesh generation, the already meshed domains will be ignored (even if the mesh parameters are different)*
ELongBoolean	**GenDomainIntersection**	*taken into account only during mesh generation, the intersection lines of domains in different planes will be generated*
ELongBoolean	**KeepGuideLinesAfterFailure**	*taken into account only during mesh generation, if some kind of fitting is specified, if there is an error during mesh generation, the generated guide lines will be preserved. The default value is false, and as such these guidelines are automatically removed in case of failure. Keeping them helps in finding potential sources of errors.*

)

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

)

RDomainVariableThickness = (

EDomainVariableThicknessType	**VariableThicknessType**	*type of variability*
Rpoint3D	**P1**	*coordinates of reference point 1*
Rpoint3D	**P2**	*coordinates of reference point 2*

Rpoint3D	**P3**	*coordinates of reference point 3 (only for dvtvTwoDirections type)*
Double	**t1**	*thickness at reference point 1*
Double	**t2**	*thickness at reference point 2*
Double	**t3**	*thickness at reference point 3 (only for dvtvTwoDirections type)*

)

RDomainExcentricity = (

EDomainExcentricityType	**ExcentricityType**	*type of eccentricity*
Rpoint3D	**P1**	*coordinates of reference point 1*
Rpoint3D	**P2**	*coordinates of reference point 2*
Rpoint3D	**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
Double	**h**	*height of the rib (including slab)*
Double	**w**	*width of the rib*
Double	**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*

)

RDomainHollowCore = (

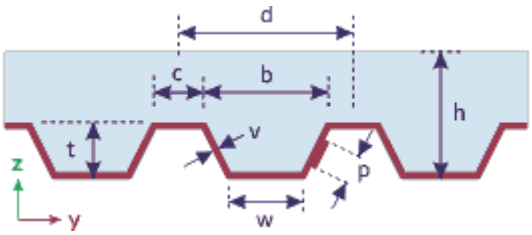
EXYDirection	**Direction**	*hole ax direction*
double	**d**	*hole interax distance [m]*
Rpoint3D	**Origin**	*origin of the grid*
EHollowHoleType	**HoleType**	*type of the hole*
double	**Fi**	*hole diameter [m] (for HoleType = hht_Circular)*
double	**b**	*hole width [m] (for HoleType = hht_Rectangular)*
double	**h**	*hole height [m] (for HoleType = hht_Rectangular)*

)

RXLAMParams = (

long	**XLAMindex**	*index in XLAM library, 1 ≤ XLAMIndex ≤ AxisVMXLAMpanels.Count*
EXLAMTopLayerDirection	**TopLayerDirection**	*top layer direction relative to the local system of the domain*
EXLAMServiceClass	**ServiceClass**	*Bending stiffness factor, must be >= 0.1 and < 1*
double	**k_def**	*deformation factor*
ELongBoolean	**k_sys**	*system strength factor is taken into account*
ELongBoolean	**k_fin**	*flexural strength is taken into account*

)



RDomainTrapezoidal = (

long	**PlateMaterial**	*index of the plate material, 1 ≤ PlateMaterial ≤ AxisVMMaterials.Count. The referenced material must be steel*
long	**InfillMaterial**	*index of the infill material, 1 ≤ InfillMaterial ≤ AxisVMMaterials.Count. The referenced material must be concrete*
EXYDirection	**Direction**	*trapezoidal axis direction*
Rpoint3D	**Origin**	*origin of the grid*
double	**h**	*total height of the slab [m]. If it is 0.00, that means that there is no infill.*
double	**t**	*height of the trapezoidal steel plate [m]*
double	**v**	*thickness of the trapezoidal steel plate [m]*
double	**d**	*distance between ridges [m]*

double	**b**	*top width of the trapezoidal steel plate [m]*
double	**w**	*bottom width of the trapezoidal steel plate [m]*
double	**p**	*width of the rolled mechanical embossments [m]*
double	**eta**	*effectiveness of shear connection []*

)

RDomainCompositeRib = (

long	**RibMaterial**	*index of the rib material, 1 ≤ RibMaterial ≤ AxisVMMaterials.Count*
long	**CrossSection**	*index of the rib cross-section, 1 ≤ CrossSection ≤ AxisVMCrossSections.Count*
EXYDirection	**Direction**	*axis direction*
Rpoint3D	**Origin**	*origin of the grid*
double	**d**	*distance between the ribs [m]*
EDomainCompRibEccType	**EccType**	*type of the rib eccentricity*
double	**Eccentricity**	*eccentricity value [m] (only for EccType = dcret_custom)*
ELongBoolean	**HasCustomShear**	*has costum shear*
double	**CustomShear**	*value of the costum shear [kN/m/m] (only in case of HasCustomShear = lbTrue)*
ELongBoolean	**ActualRibs**	*if lbTrue, actual rib elements will be created*

)

Functions

long **Add** ([in] SAFEARRAY(long) **LineIds,** [i/o] RSurfaceAttr **SurfaceAttr**)

LineIds *line indexes defining the domain's contour. Lines must be in the same plane.*

SurfaceAttr *domain properties*

Defines a new domain.
If successful, returns the new domain index, otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBounds, deEmptyContour [LineIds array is empty], deMoreThanOneContourFound [multiple contours], deThicknessMustBePositive, deMaterialIndex, deReferenceIndexOutOfBounds, deDomainInvalidType, deElasticFoundationNegative).

---

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

---

long **AddCompositeRib** ([in] SAFEARRAY(long) **LineIds,** [i/o] RSurfaceAttr **SurfaceAttr,** [i/o] RDomainCompositeRib **CompositeRibParams**)

LineIds *line indexes defining the domain's contour. Lines must be in the same plane.*

SurfaceAttr *general domain properties*

CompositeRibParams *parameters of the composite rib domain*

Defines a new composite rib domain.
If successful, returns the new domain index, otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBounds, deEmptyContour [LineIds array is empty], deMoreThanOneContourFound [multiple contours], deThicknessMustBePositive, deMaterialIndex, deReferenceIndexOutOfBounds, deDomainInvalidType, deElasticFoundationNegative, deCompositeRib).*

---

long **AddHollowCore** ([in] SAFEARRAY(long) **LineIds,** [i/o] RSurfaceAttr **SurfaceAttr,** [i/o] RDomainHollowCore **HollowCoreParams**)

LineIds *line indexes defining the domain's contour. Lines must be in the same plane.*

SurfaceAttr *general domain properties*

HollowCoreParams *hollow core parameters of the domain*

Defines a new hollow core domain.
If successful, returns the new domain index, otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBounds, deEmptyContour [LineIds array is empty], deMoreThanOneContourFound [multiple contours], deThicknessMustBePositive, deMaterialIndex, deReferenceIndexOutOfBounds, deDomainInvalidType, deElasticFoundationNegative).

---

long **AddTrapezoidal** ([in] SAFEARRAY(long) **LineIds,** [i/o] RSurfaceAttr **SurfaceAttr,** [i/o] RDomainTrapezoidal **TrapezoidalParams**)

LineIds *line indexes defining the domain's contour. Lines must be in the same plane.*

SurfaceAttr *general domain properties*

TrapezoidalParams *trapezoidal parameters of the domain*

Defines a new trapezoidal domain.
If successful, returns the new domain index, otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBounds, deEmptyContour [LineIds array is empty], deMoreThanOneContourFound [multiple contours], deThicknessMustBePositive, deMaterialIndex, deReferenceIndexOutOfBounds, deDomainInvalidType, deElasticFoundationNegative, deTrapezoidal).*

---

long **AddXLAM** ([in] SAFEARRAY(long) **LineIds,** [i/o] RSurfaceAttr **SurfaceAttr,** [i/o] RXLAMParams **XLAMParams**)

LineIds *line indexes defining the domain's contour. Lines must be in the same plane.*

SurfaceAttr *general domain properties*

XLAMParams *XLAM parameters of the domain. Warning : the thickness of the domain present in SurfaceAttr will be overriden by the thickness of the selected XLAM type.*

Defines a new XLAM domain.
If successful, returns the new domain index, otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBounds, deEmptyContour [LineIds array is empty], deMoreThanOneContourFound [multiple contours], deThicknessMustBePositive, deMaterialIndex, deReferenceIndexOutOfBounds, deDomainInvalidType, deElasticFoundationNegative).

---

long **BulkAdd** ([in] SAFEARRAY(long) **LineIdCounts,** [in] SAFEARRAY(long) **BulkLineIds,** [in] SAFEARRAY(RSurfaceAttr) **SurfaceAttrs,** [out] SAFEARRAY(long)* **DomainIds**)

LineIdCounts *list of contour line counts for each domain. There will be created as many domains as the count of LindIdCounts.*
BulkLineIds *appended list of domain contour lines. LineIdCounts controls its segmentation. If LineIdCounts 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 ≤ LineIndex ≤ AxisVMLines.Count*
SurfaceAttrs *domain properties for each domain*
DomainIds *the indexes of the newly created domains. If there was an error, it will be empty*

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).

---

long **BulkGetDomains** ([in] SAFEARRAY(long) **DomainIds,** [out] SAFEARRAY(long)* **LineIdCounts,** [out] SAFEARRAY(long)* **BulkLineIds,** [out] SAFEARRAY(RSurfaceAttr)* **SurfaceAttrs**)

DomainIds *list of domain indexes. Each index must satisfy : 1 ≤ DomainId ≤ AxisVMDomains.Count*
LineIdCounts *list of contour line counts for each domain.*
BulkLineIds *appended list of domain contour lines. LineIdCounts controls its segmentation. If LineIdCounts 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)*
SurfaceAttrs *domain properties of each domain*

Queries several domains in one step.
If successful, returns the number of queried domains. The possible error codes are : (errDatabaseNotReady, errIndexOutOfBounds, errInternalException, errOutOfMemory).

---

long **BulkSetDomains** ([in] SAFEARRAY(long) **DomainIds,** [in] SAFEARRAY(long)* **LineIdCounts,** [in] SAFEARRAY(long)* **BulkLineIds,** [in] SAFEARRAY(RSurfaceAttr)* **SurfaceAttrs**)

DomainIds *list of domain indexes. Each index must satisfy : 1 ≤ DomainId ≤ AxisVMDomains.Count*
LineIdCounts *list of contour line counts for each domain.*
BulkLineIds *appended list of domain contour lines. LineIdCounts controls its segmentation. If LineIdCounts 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)*
SurfaceAttrs *domain properties of each domain*

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 are : (errDatabaseNotReady, errIndexOutOfBounds, errInternalException, errOutOfMemory, deEmptyContour, deMoreThanOneContourFound, deThicknessMustBePositive, deMaterialIndex, deReferenceIndexOutOfBounds, 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**)

Warning! *This function has become obsolete, was superseded by GenerateMeshOnSelectedDomains_V171*

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 **GenerateMeshOnSelectedDomains_V171** ([i/o] RDomainMeshParameters_V171 **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*

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

---

long **GetCompositeRibParameters** ([in] long **Index,** [out] RDomainCompositeRib **CompositeRibParams**)

Index *domain index*

CompositeRibParams *parameters of the composite rib domain*

If successful, returns the domain index, otherwise returns an error code (see EGeneralError or EDomainsError).

---

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 stifness matrix, more in manual: domain with custom stiffness matrix*

B *coupling stifness matrix, more in manual: domain with custom stiffness matrix*

D *plate flexural stifnness matrix, more in manual: domain with custom stiffness matrix*

S *adjusted shear stifness matrix, more in manual: domain with custom stiffness matrix*

Get the domain's custom stiffness matrixes. If successful, returns the domain index, otherwise returns an error code (see EGeneralError or EDomainsError).

---

long **GetExcentricity** ([in] long **Index,** [i/o] RDomainExcentricity **DomainExcentricity**)

Index *domain index*

DomainExcentricity *parameters for defining eccentricity*

If successful, returns the domain index, otherwise returns an error code (see EGeneralError or EDomainsError).

---

long **GetHollowCoreParameters** ([in] long **Index,** [i/o] RDomainHollowCore **HollowCoreParams**)

Index *domain index*

HollowCoreParams *hollow core parameters*

If successful, returns the domain index, otherwise returns an error code (see EGeneralError or EDomainsError).

---

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*

If successful, returns the domain index, otherwise returns an error code (see EGeneralError or EDomainsError).

---

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

ItemIds *list of selected domains*

If successful, returns the number of selected elements ,otherwise returns an error code (see EGeneralErrors or EDomainsError).

---

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

DomainIds *array of selected wall domain indexes.*

If successful, returns the number of selected wall domains, otherwise returns an error code (errDatabaseNotReady).

---

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

DomainIds *array of selected slab domain indexes.*

If successful, returns the number of selected slab domains, otherwise returns an error code (errDatabaseNotReady).

---

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

DomainIds *array of selected other domain indexes.*

If successful, returns the number of selected other domains, otherwise returns an error code (errDatabaseNotReady).

---

long **GetTrapezoidalParameters** ([in] long **Index,** [out] RDomainTrapezoidal **TrapezoidalParams**)

Index *domain index*

TrapezoidalParams *trapezoidal parameters of the domain*

If successful, returns the domain index, otherwise returns an error code (see EGeneralError or EDomainsError).

---

long **GetVariableThickness** ([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 **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*

If successful, returns the domain index, otherwise returns an error code (see EGeneralError or EDomainsError).

---

long **GetXLAMParameters_V161** ([in] long **Index,** [out] RXLAMParams **XLAMParams**)

Index *domain index*

XLAMParams *XLAM parameters of the domain*

If successful, returns the domain index, otherwise returns an error code (see EGeneralError or EDomainsError).

---

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_"*

Example: NewBase=100 and FormatStr= 'new_' then names are: 'new100', 'new101',…etc.
If successful, returns number of selected domains. Otherwise returns an error code (errDatabaseNotReady).

---

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

Select *selection state*

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

NOTE: Call Refresh function afterwards if not called between functions BeginUpdate and EndUpdate

---

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

Select *selection state*

If Select is True, selects all slab domains.
If Select is False, deselects all slab domains.
If successful, returns the number of selected slab domains, 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 other type of domain.
If Select is False, deselects all other type of domain.
If successful, returns the number of selected other type of domain, otherwise returns an error code (errDatabaseNotReady).

NOTE: Call Refresh function afterwards if not called between functions BeginUpdate and EndUpdate

---

long **SelectAllWallsAtStorey** ([in] ELongBoolean **Select,** [in] long **StoreyId**)

Select *selection state*
StoreyId *storey index*

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).

NOTE: Call Refresh function afterwards if not called between functions BeginUpdate and EndUpdate

---

long **SelectAllSlabsAtStorey** ([in] ELongBoolean **Select,** [in] long **StoreyId**)

Select *selection state*
StoreyId *storery index*

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).

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*

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).

NOTE: Call Refresh function afterwards if not called between functions BeginUpdate and EndUpdate

---

long **SetCompositeRibParameters** ([in] long **Index,** [i/o] RDomainCompositeRib **CompositeRibParams**)

Index *domain index*
CompositeRibParams *composite rib parameters of the domain*

If successful, returns the domain index, otherwise returns an error code (see EGeneralError or EDomainsError).

---

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 stifness matrix, more in manual: domain with custom stiffness matrix*
B *coupling stifness matrix, more in manual: domain with custom stiffness matrix*
D *plate flexural stifnness matrix, more in manual: domain with custom stiffness matrix*
S *adjusted shear stifness matrix, more in manual: domain with custom stiffness matrix*

Set the domain's custom stiffness matrixes. If successful, returns the domain index, otherwise returns an error code (see EGeneralError or EDomainsError).

---

long **SetExcentricity** ([in] long **Index,** [i/o] RDomainExcentricity **DomainExcentricity**)

Index *domain index*

DomainExcentricity *parameters for defining eccentricity*

Returns the domain index, otherwise returns an error code (see EGeneralError or EDomainsError).

---

long **SetHollowCoreParameters** ([in] long **Index,** [i/o] RDomainHollowCore **HollowCoreParams**)

Index *domain index*

HollowCoreParams *hollow core parameters*

If successful, returns the domain index, otherwise returns an error code (see EGeneralError or EDomainsError).

---

long **SetRibbedDomainActual** ([in] long **Index,** [in] ElongBoolean **Actual**)

Index *domain index. Must be already a ribbed domain.*

Actual *lbTrue – will use actual ribs*

lbFalse – will use virtual ribs

The domain must be already a ribbed domain. If it isn't a ribbed one, the error deNotARibbedDomain is returned. If successful, returns the domain index, otherwise returns an error code (see EGeneralError or EDomainsError).

---

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. If you don't want ribs in this direction, set h, w, d to 0.00*

y *parameters for defining ribs in y direction. If you don't want ribs in this direction, set h, w, d to 0.00*

If successful, returns the domain index, otherwise returns an error code (see EGeneralError or EDomainsError).

---

long **SetTrapezoidalParameters** ([in] long **Index,** [i/o] RDomainTrapezoidal **TrapezoidalParams**)

Index *domain index*

TrapezoidalParams *trapezoidal parameters of the domain*

If successful, returns the domain index, otherwise returns an error code (see EGeneralError or EDomainsError).

---

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).

---

long **SetXLAMParameters_V161** ([in] long **Index,** [i/o] RXLAMParams **XLAMParams**)

Index *domain index*

XLAMParams *XLAM parameters of the domain. Warning : the thickness of the domain will be overriden by the thickness of the selected XLAM type*

If successful, returns the domain index, otherwise returns an error code (see EGeneralError or EDomainsError).

Properties

AxisVMAttachments*	**Attachments** *Get the attachments interface*
AxisVMAttributes*	**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	**IsCompositeRib** [long **Index**] *If lbTrue then it is a composite rib domain (read only)* **Index** *index of the domain*
ElongBoolean	**IsExcentic** [long **Index**] *If lbTrue then has excentric parameters (read only)* **Index** *index of the domain*
ElongBoolean	**IsHollowCore** [long **Index**] *If lbTrue then it is a hollow core domain (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	**IsTrapezoidal** [long **Index**] *If lbTrue then it is a trapezoidal domain (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* *NOTE: Call Refresh function afterwards if not called between functions BeginUpdate and EndUpdate*
long	**SelCount** *Get number of selected domains in the model*
double	**StiffnessReduction[**long **Index]** • ***Warning!*** *This function was extended by StiffnessReduction_V153. Get or set stiffness reduction for the cross-section area (Ac). Default value is 1. Used only for atLinearStatic and atLinearVibration analysis types. (only valid for Eurocode, SIA, NTC standards)*
double	**StiffnessReduction_V153[**long **Index,** ESurfaceStiffnessReduction **Component]** *Get or set stiffness reduction for the given component. Default value is 1. Used only for atLinearStatic and atLinearVibration analysis types. (only valid for SIA, NTC and Eurocode 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 (available only in EC_NO)*
ecClassXA2 = 18,	*Environment class XA2 (available only in EC_NO)*
ecClassXA3 = 19,	*Environment class XA3 (available only in EC_NO)*
ecClassXA4 = 20,	*Environment class XA4 (available only in EC_NO)*
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)*

}

Enviromental 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

**In case of Norwegian EC S1 and S2 structural classes are available only. S1 refers to 50 years design life time, S2 refers to 100 years design life time*

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**
= {
sltd_OneWayX = 0 *One way spanning slab in x direction*
sltd_OneWayY = 1 } *One way spanning slab in y direction*
Direction of one-way spanning slab

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 }
For RC design in accordance with NEN design code.

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**	*x, y, z coordinates of a a 3D point or components of a 3D vector [m]*

)

RRebarPos = (

double	**TopX**	*Distance of top reinforcement centre line to top edge on x direction*
double	**BottomX**	*Distance of bottom reinforcement centre line to bottom edge on x direction*
double	**TopY**	*Distance of top reinforcement centre line to top edge on y direction*
double	**BottomY**	*Distance of bottom reinforcement centre line to bottom edge on y direction*

)
Rebar position

RReinforcementParameters = (

long	**ConcreteId**	*Material index of the concrete*
long	**RebarSteelGradeId**	*Material index of the rebar steel*
Double	**Thickness**	*Thickness of the domain (surface element) used for RC design [m] *EC,ITA,SIA: in case of trapezoidal sheet domains, thickness is considered only if TrapSheetOnlyFormWork = lbTrue*

)

RReinforcementParameters_DIN = (

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 ApplyMinimumCover = lbFalse)*
Double	**cb**	*cover of the main reinforcement at bottom (only if ApplyMinimumCover = lbFalse)*
Double	**AggregateSize**	*maximum size of the concrete aggregate*
ELongBoolean	**ApplyMinimumCover**	*Calculate concrete covers to all reinforcements based on enviroment and structural class*
EEnvironmentClass	**EnvClass_T**	*exposure class for top surface*
EEnvironmentClass	**ExpClass_B**	*exposure class for bottom surface*

Double	**ks**	*load factor for seismic forces (default is 1 =>no change)*
Double	**ShearReinforcementAngle**	*angle of shear reinforcement for VRdmax calculation*
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*
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 ApplyMinimumCover = lbFalse)*
Double	**cb**	*cover of the main reinforcement at bottom (only if ApplyMinimumCover = lbFalse)*
Double	**AggregateSize**	*maximum size of the concrete aggregate*
ELongBoolean	**ApplyMinimumCover**	*Calculate concrete covers to all reinforcements based on enviroment 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 lbFalse: level arm is calculated from the internal forces; lbTrue: 0.9*d level arm is used*
ELongBoolean	**SeelHoferMartiEquation**	*use the design formula of H.Seelhofer and P.Marti*
ELongBoolean	**TrapSheetOnlyFormWork**	*trapezoidal sheet is used as formwork only*
ELongBoolean	**TrapSheetOneLayerReinf**	*there is only one layer reinforcement in the upper section of trapezoidal sheet slab*
ELongBoolean	**TrapSheetConsidered**	*trapezoidal sheet is considered in the calculation of required reinforcement*

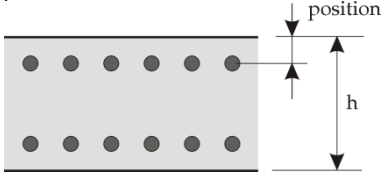
)

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

RReinforcementParameters_MSZ = (

RRebarPos	**RebarPos**	*Rebar position, centre line of rebar to the edge*
Double	**UnfavorableEccentricity_Pos**	*N > 0*
Double	**UnfavorableEccentricity_Neg**	*N < 0*
ELongBoolean	**AutoPsi**	*See MSZ national design code*

)



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 ApplyMinimumCover = lbFalse)*
Double	**cb**	*cover of the main reinforcement at bottom (only if ApplyMinimumCover = lbFalse)*
Double	**AggregateSize**	*maximum size of the concrete aggregate*
ELongBoolean	**ApplyMinimumCover**	*Calculate concrete covers to all reinforcements based on enviroment 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 ApplyMinimumCover = lbFalse)*
Double	**cb**	*cover of the main reinforcement at bottom (only if ApplyMinimumCover = lbFalse)*
Double	**AggregateSize**	*maximum size of the concrete aggregate*
ELongBoolean	**ApplyMinimumCover**	*Calculate concrete covers to all reinforcements based on enviroment 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 lbFalse: level arm is calculated from the internal forces; lbTrue: 0.9*d level arm is used*
ELongBoolean	**SeelHoferMartiEquation**	*use the design formula of H.Seelhofer and P.Marti*

ELongBoolean	**TrapSheetOnlyFormWork**	*trapezoidal sheet is used as formwork only*
ELongBoolean	**TrapSheetOneLayerReinf**	*there is only one layer reinforcement in the upper section of trapezoidal sheet slab*
ELongBoolean	**TrapSheetConsidered**	*trapezoidal sheet is considered in the calculation of required reinforcement*

)

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 ApplyMinimumCover = lbFalse)*
Double	**cb**	*cover of the main reinforcement at bottom (only if ApplyMinimumCover = lbFalse)*
Double	**AggregateSize**	*maximum size of the concrete aggregate*
ELongBoolean	**ApplyMinimumCover**	*Calculate concrete covers to all reinforcements based on enviroment 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	**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≤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 lbFalse: level arm is calculated from the internal forces; lbTrue: 0.9*d level arm is used*
ELongBoolean	**SeelHoferMartiEquation**	*use the design formula of H.Seelhofer and P.Marti*
ELongBoolean	**TrapSheetOnlyFormWork**	*trapezoidal sheet is used as formwork only*
ELongBoolean	**TrapSheetOneLayerReinf**	*there is only one layer reinforcement in the upper section of trapezoidal sheet slab*
ELongBoolean	**TrapSheetConsidered**	*trapezoidal sheet is considered in the calculation of required reinforcement*

)

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*

)

Functions

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

LineIds *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, errIndexOutOfBounds, deEmptyHole [LineIds 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, errIndexOutOfBounds).

---

long **DeleteElasticFoundation**

If delete successful, returns domainID, otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBounds).

---

long **DeleteReinforcementParameters**

If successful, returns index of the domain, otherwise returns an error code(errDatabaseNotReady, errIndexOutOfBounds)

---

long **DeleteMesh**

Deletes the mesh of domain. If successful, returns index of the domain, otherwise returns an error code(errDatabaseNotReady, errIndexOutOfBounds)

---

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, errIndexOutOfBounds).

---

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, errIndexOutOfBounds)

---

long **GenerateMesh** ([i/o] RDomainMeshParameters **MeshParameters,**
[out] SAFEARRAY(long)* **ErrorCodes,** [out] SAFEARRAY(long)* **ErrorPoints,**
[out] SAFEARRAY(long)* **ErrorLines**)

Warning! *This function has become obsolete, was superseded by GenerateMesh_V171*

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 **GenerateMesh_V171** ([i/o] RDomainMeshParameters_V171 **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**)

A *membrane stifness matrix, more in manual: domain with custom stiffness matrix*
B *coupling stifness matrix, more in manual: domain with custom stiffness matrix*
D *plate flexural stifnness matrix, more in manual: domain with custom stiffness matrix*
S *adjusted shear stifness matrix, more in manual: domain with custom stiffness matrix*

Get the domain's custom stiffness matrixes. If successful, returns the domain index.
Possible error codes are errDatabaseNotReady.

---

long **GetMeshParameters** ([i/o] RDomainMeshParameters **Value**)

Warning! *This function has become obsolete, was superseded by GetMeshParameters_V171*

Get mesh parameters of the domain. If successful, returns the domain index.
Possible error codes are errIndexOutOfBounds, errDatabaseNotReady.

---

long **GetMeshParameters_V171** ([i/o] RDomainMeshParameters_V171 **Value**)

Get mesh parameters of the domain. MeshOnlyUnmeshed, GenDomainIntersection, KeepGuideLinesAfterFailure are filled with constant lbFalse, as they are only generation time parameters. If successful, returns the domain index.
Possible error codes are errIndexOutOfBounds, errDatabaseNotReady.

---

long **GetMeshSurfacesCoordinates** ([out] SAFEARRAY(RSurfaceCoordinates)* **SurfacesCoordinates**)

SurfacesCoordinates *Surface coordinates of all surfaces*

If successful, returns number of generated surface elements in domain, otherwise returns an error code(errDatabaseNotReady, errIndexOutOfBounds)

---

long **GetNormalVector** ([i/o] RPoint3D **Value**)

Get the normal vector of the domain. If successful, returns the domain index.
Possible error codes are errIndexOutOfBounds, errDatabaseNotReady.

---

long **GetReinforcementParameters** ([i/o] RReinforcementParameters* **ReinforcementParameters**, [out] SAFEARRAY(byte) **DesignCodeParameters**)

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).*

If successful, returns index of the domain, otherwise returns an error code(errDatabaseNotReady, errIndexOutOfBounds)
Record type depends on current **NationalDesignCode**
RReinforcementParameters_DIN:
ndcGerman_DIN1045_1

RReinforcementParameters_EC:

- *ndcEuroCode, ndcEuroCode_GER , ndcEuroCode_Austrian, ndcEuroCode_UK, ndcEuroCode_NL , ndcEuroCode_FIN , ndcEuroCode_HU, ndcEuroCode_RO, ndcEuroCode_CZ , ndcEuroCode_B , ndcEuroCode_PL , ndcEuroCode_DK, ndcEuroCode_S*

RReinforcementParameters_ITA

- *ndcItalian*

RReinforcementParameters_MSZ

- *ndcHungarian_MSZ*

RReinforcementParameters_NEN

- *ndcDutch_NEN*

RReinforcementParameters_SIA
- *ndcSwiss_SIA26x*

RReinforcementParameters_STAS
- *ndcRomanian_STAS*

---

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) **LineIds,** [i/o] RSurfaceAttr **SurfaceAttr**)

LineIds *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 [LineIds array is empty], deMoreThanOneContourFound [multiple contours]).

---

long **Modify_vb** (Visual Basic compatible function of **Modify**)

---

long **ModifyHole** ([in] long **HoleIndex**, [in] SAFEARRAY(long) **LineIds**)

HoleIndex *Index of the hole*
LineIds *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) **LineIds**)

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 stifness matrix, more in manual: domain with custom stiffness matrix*
B *coupling stifness matrix, more in manual: domain with custom stiffness matrix*
D *plate flexural stifnness matrix, more in manual: domain with custom stiffness matrix*
S *adjusted shear stifness 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**)

Warning! *This function has become obsolete, was superseded by SetMeshParameters_V171*

Set mesh parameters of the domain. If successful, returns the domain index.
Possible error codes are errIndexOutOfBounds, errDatabaseNotReady.

---

long **SetMeshParameters_V171** ([i/o] RDomainMeshParameters_V171 **Value**)

Set mesh parameters of the domain. MeshOnlyUnmeshed, GenDomainIntersection, KeepGuideLinesAfterFailure are ignored, as they are only generation time parameters. 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:
ndcGerman_DIN1045_1

RReinforcementParameters_EC:
- *ndcEuroCode, ndcEuroCode_GER , ndcEuroCode_Austrian, ndcEuroCode_UK, ndcEuroCode_NL , ndcEuroCode_FIN , ndcEuroCode_HU, ndcEuroCode_RO, ndcEuroCode_CZ , ndcEuroCode_B , ndcEuroCode_PL , ndcEuroCode_DK, ndcEuroCode_S*

RReinforcementParameters_ITA
- *ndcItalian*

RReinforcementParameters_MSZ
- *ndcHungarian_MSZ*

RReinforcementParameters_NEN
- *ndcDutch_NEN*

RReinforcementParameters_SIA
- *ndcSwiss_SIA26x*

RReinforcementParameters_STAS
ndcRomanian_STAS

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*
ELongBoolean	**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**
ELongBoolean	**MeshExists** *True if the domain is meshed (Read only property*)
SAFEARRAY(long)*	**MeshSurfaceIds** *Get array of mesh surface element indexes according to IAxisVMSurfaces*
BSTR	**Name** *Get name of the domain (read only property)*
ELongBoolean	**IsWall** *True if domain is a wall (read only property)*
ELongBoolean	**IsSlab** *True if domain is a slab (read only property)*
ELongBoolean	**IsOtherType** *True if domain is an other type (read only property)*
long	**OuterDomainId** *Get index of the outer domain, if successful I. Otherwise returns an error code(errDatabaseNotReady, errIndexOutOfBounds)*
ELongBoolean	**ReinforcementParametersExists** *True if finds reinforcement parameters. (read only property)*
long	**SeismicStoreyId** *Get seismic storey index of the domain, if successful I. Otherwise returns an error code(errDatabaseNotReady, errIndexOutOfBounds)*
double	**StiffnessReduction** • ***Warning!*** *This function was extended by StiffnessReduction_V153.Get or set stiffness reduction for the cross-section area (Ac). Default value is 1. Used only for atLinearStatic and atLinearVibration analysis types. (only valid for Eurocode, SIA, NTS standards)*
double	**StiffnessReduction_V153** **[**ESurfaceStiffnessReduction **Component]** *Get or set stiffness reduction for the given component. Default value is 1. Used only for atLinearStatic and atLinearVibration analysis types. (only valid for Eurocode, SIA, NTS standards)*
long	**StoreyId** *Get storey index of the domain, if successful I. Otherwise returns an error code(errDatabaseNotReady, errIndexOutOfBounds)*
ELongBoolean	**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 < DomainId ≤ 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, errIndexOutOfBounds).

---

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

DomainId	*index of the domain support* *(0 < DomainId ≤ IAxisVMDomainSupports.Count)*
StiffnessesXYZ	*stiffnesses of the domain support*
ShearStiffness	*shear stiffness of the domain support (noted as R_G on the window for supports) [kN/m]*
NonlinearityXYZ	*nonlinear behaviour of the domain support*
ResistancesXYZ	*resistance of the domain support*

Adds a Winkler-Pasternak domain support. If successful, returns number of domain supports, otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBounds).

---

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

Index	*index of the domain support 1 ≤ Index ≤ Count*

If successful returns domain support index, otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBounds)

---

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 ≤ Index ≤ 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 ≤ Index ≤ 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 ≤ Index ≤ 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)**
NonLinearityXYZ *nonlinearity of the domain support*
If successful, returns the index of the domain support otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBounds).

long **GetPasternakStiffness** ([i/o] double* **Value)**
Value *shear stiffness of the domain support (noted as R_G on the window for supports) [kN/m]*
If successful, returns the index of the domain support otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBounds).

long **GetStiffnessesXYZ** ([i/o] RStiffnessesXYZ* **StiffnessesXYZ)**
StiffnessesXYZ *stiffnesses of the elastic foundation*
If successful, returns the index of the domain support otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBounds).

long **GetResistancesXYZ** ([i/o] RResistancesXYZ* **ResistancesXYZ)**
ResistancesXYZ *resistance of the domain support*
If successful, returns the index of the domain support otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBounds).

long **SetPasternakStiffness** ([in] double **Value)**
Value *shear stiffness of the domain support (noted as R_G on the window for supports) [kN/m]*
If successful, returns the index of the domain support otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBounds).

long **SetStiffnessesXYZ** ([i/o] RStiffnessesXYZ***StiffnessesXYZ)**
StiffnessesXYZ *stiffnesses of the domain support*
If successful, returns the index of the domain support otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBounds).

long **SetNonLinearityXYZ** ([i/o] RNonLinearityXYZ* **NonLinearityXYZ)**
NonLinearityXYZ *nonlinearity of the domain support*
If successful, returns the index of the domain support otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBounds).

long **SetResistancesXYZ** ([i/o] RResistancesXYZ* **ResistancesXYZ)**
ResistancesXYZ *resistance of the domain support*
If successful, returns the index of the domain support otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBounds).

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 ≤ Index ≤ 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 ≤ Index ≤ 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 ≤ Index ≤ Count*

WindowIndex *index of the displayed window, 0 ≤ Index ≤ 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	*point of function out of bounds*
fueFailedToModifyFunction = -100002	*function modification failed*
fueFileExists = -100003	*error during saving function, file already exists*
fueFailedToAddFromFile = -100004	*error during loading function, file open failed*
fueNameAlreadyExists = -100005	*function with the same name already exists in the model*
fueInvalidFunction = -100006 }	*function points not valid, see AxisVM manual for more info*

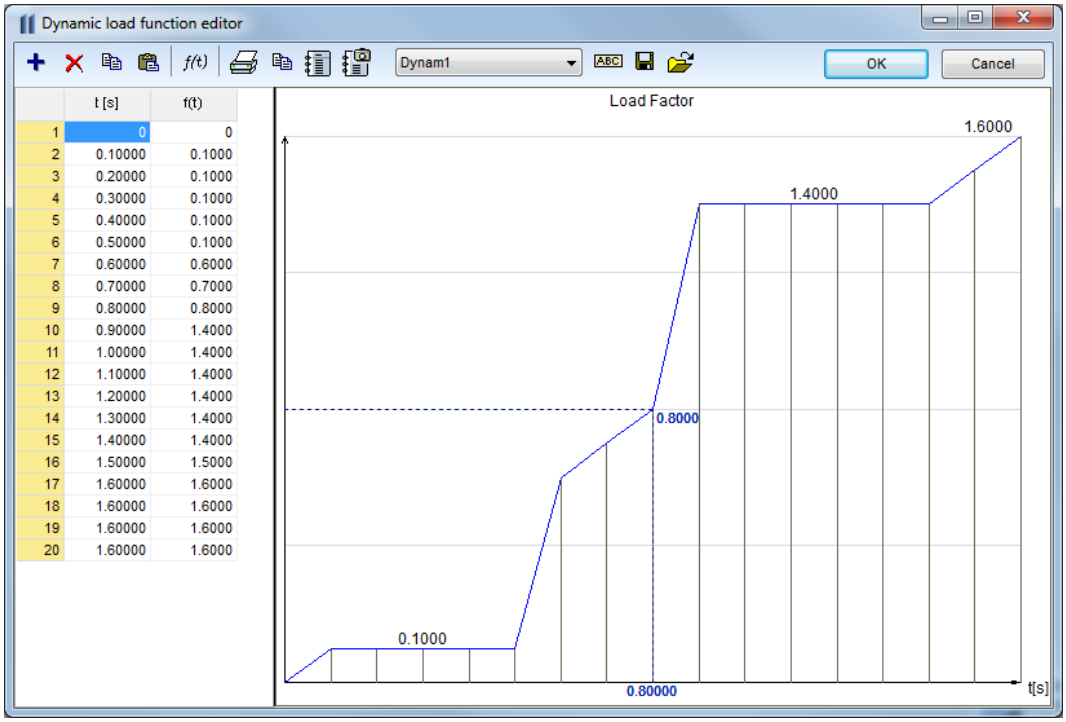
Functions

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:



	t [s]	f(t)
1	0	0
2	0.10000	0.1000
3	0.20000	0.1000
4	0.30000	0.1000
5	0.40000	0.1000
6	0.50000	0.1000
7	0.60000	0.6000
8	0.70000	0.7000
9	0.80000	0.8000
10	0.90000	1.4000
11	1.00000	1.4000
12	1.10000	1.4000
13	1.20000	1.4000
14	1.30000	1.4000
15	1.40000	1.4000
16	1.50000	1.5000
17	1.60000	1.6000
18	1.60000	1.6000
19	1.60000	1.6000
20	1.60000	1.6000

If successful, returns index of the new dynamic load 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 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*

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

---

long **Clear**

Deletes all dynamic load functions.

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

---

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

index *dynamic load function index*

If successful, returns number of dynamic load functions after delete, otherwise an error code (errDatabaseNotReady or errIndexOutOfBounds).

---

long **DeletePoint** ([in] long **index,** [in] long **PointIndex,**)

index *dynamic load function index*

PointIndex *index of a point in dynamic load function*

If successful, returns number of points after deleting a point from function, otherwise an error code (fuePointIndexOutOfBounds, errDatabaseNotReady or errIndexOutOfBounds).

---

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*

Deletes function point range starting with point index **StartPointIndex** (including) and ending with **EndPointIndex** (including).

If successful, returns number of points after deleting points from function, otherwise an error code (fuePointIndexOutOfBounds, errDatabaseNotReady or errIndexOutOfBounds).

---

long **GetPoints** ([in] long **index,** [out] SAFEARRAY(RPoint2d)* **FunctionPoints**)

index *dynamic load function index*

FunctionPoints *point array of the dynamic load 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 dynamic load function*

If successful, returns dynamic load function index, otherwise an error code (errDatabaseNotReady).

---

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.*

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 *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**] G*et number of points of the dynamic load function with index* **Index**.

IAxisVMEdgeConnections

Edge connections of the model.

Error codes

enum **EEdgeConnectionError:** = {

eceLineIndexOutOfBounds = -100001	*EEdgeConnectionRec.LineId in IAxisVMEdgeConnections.Add is invalid*
eceDomainIndexOutOfBounds = -100002	*EEdgeConnectionRec.DomainId in IAxisVMEdgeConnections.Add is invalid*
eceErrorAdding = -100003	*IAxisVMEdgeConnections.Add was not successful I*
eceSpringError = -100004}	*wrong spring for the task (for example a rotational one where a translational is expected)*

Records / structures

REdgeConnectionRec = (

Warning! This record has become obsolete, it was superseded by REdgeConnectionRec_V172

long	**LineID**	*Id of line (edge)*
long	**DomainID**	*Id of domainID*
RStiffnesses	**Stiffnesses**	*Stiffnesses*
RResistances	**Resistances**	*Resistances*

)

REdgeConnectionRec_V172 = (

long	**LineID**	*Id of line (edge)*
long	**DomainID**	*Id of domainID*
RSpringParamIndexes	**SpringParamIndexes**	*Indexes of the spring characteristics*

)

Functions

long **Add** ([i/o] REdgeConnectionRec* **EdgeConnectionRec**)

Warning! *This function has become obsolete, was superseded by Add_V172*

EdgeConnectionRec *Edge connection parameters*

If successful returns EdgeConnectionID, otherwise an error code (errDatabaseNotReady, eceLineIndexOutOfBounds, eceDomainIndexOutOfBounds, eceErrorAdding)

---

long **Add_V172** ([i/o] REdgeConnectionRec_V172* **EdgeConnectionRec**)

EdgeConnectionRec *Edge connection parameters*

If successful returns EdgeConnectionID, otherwise an error code (errDatabaseNotReady, eceLineIndexOutOfBounds, eceDomainIndexOutOfBounds, eceErrorAdding, eceSpringError)

---

long **Clear**

Returns number of deleted edge connections, otherwise an error code (errDatabaseNotReady).

---

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

Index *index of the edge connection, 1 ≤ Index ≤ Count*

If successful returns EdgeConnectionID, otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBounds)

---

long **DeleteSelected**

Returns number of deleted edge connections, otherwise an error code (errDatabaseNotReady).

---

long **GetRec** ([in] long **Index,** [i/o] REdgeConnectionRec* **EdgeConnectionRec**)

Warning! *This function has become obsolete, was superseded by GetRec_V172*

Index *Edge connection index, 1 ≤ Index ≤ Count*

EdgeConnectionRec *Edge connection parameters*

If successful returns EdgeConnectionID, otherwise an error code (errDatabaseNotReady, errIndexOutOfBounds)

---

long **GetRec_V172** ([in] long **Index,** [i/o] REdgeConnectionRec_V172* **EdgeConnectionRec**)

Index *Edge connection index, 1 ≤ Index ≤ Count*

EdgeConnectionRec *Edge connection parameters*

If successful returns EdgeConnectionID, otherwise an error code (errDatabaseNotReady, errIndexOutOfBounds)

---

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

ItemIds *Index list of selected edge connections*

Returns the number of selected edge connections

---

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

Select *selection state*

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

---

long **SetRec** ([in] long **Index,** [i/o] REdgeConnectionRec* **EdgeConnectionRec**)

Warning! *This function has become obsolete, was superseded by SetRec_V172*

Index *Edge connection index, 1 ≤ Index ≤ Count*
EdgeConnectionRec *Edge connection parameters*
If successful returns EdgeConnectionID, otherwise an error code (errDatabaseNotReady, errIndexOutOfBounds)

---

long **SetRec_V172** ([in] long **Index,** [i/o] REdgeConnectionRec_V172* **EdgeConnectionRec**)

Index *Edge connection index, 1 ≤ Index ≤ Count*
EdgeConnectionRec *Edge connection parameters*
If successful returns EdgeConnectionID, otherwise an error code (errDatabaseNotReady, errIndexOutOfBounds, eceSpringError)

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 ≤ Index ≤ 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*

IAxisVMEnvelopes

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	*LoadCaseId is invalid*
eeLoadCombinationIdOutOfBounds = -100002	*LoadCombinationId is invalid*
eeErrorAddingEnvelope = -100003	*Can’t add new envelope*
eeErrorModifyingEnvelope = -100004	*Can’t edit or modify the envelope*
eeNotUserDefinedEnvelope = -100005 }	*Not an user defined envelope*

Enumerated types

enum **EEnvelopeGroup** = {

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

Type of envelope group

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).

---

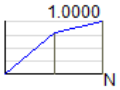
long **Update**

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	**UserEnevelopeCount** *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

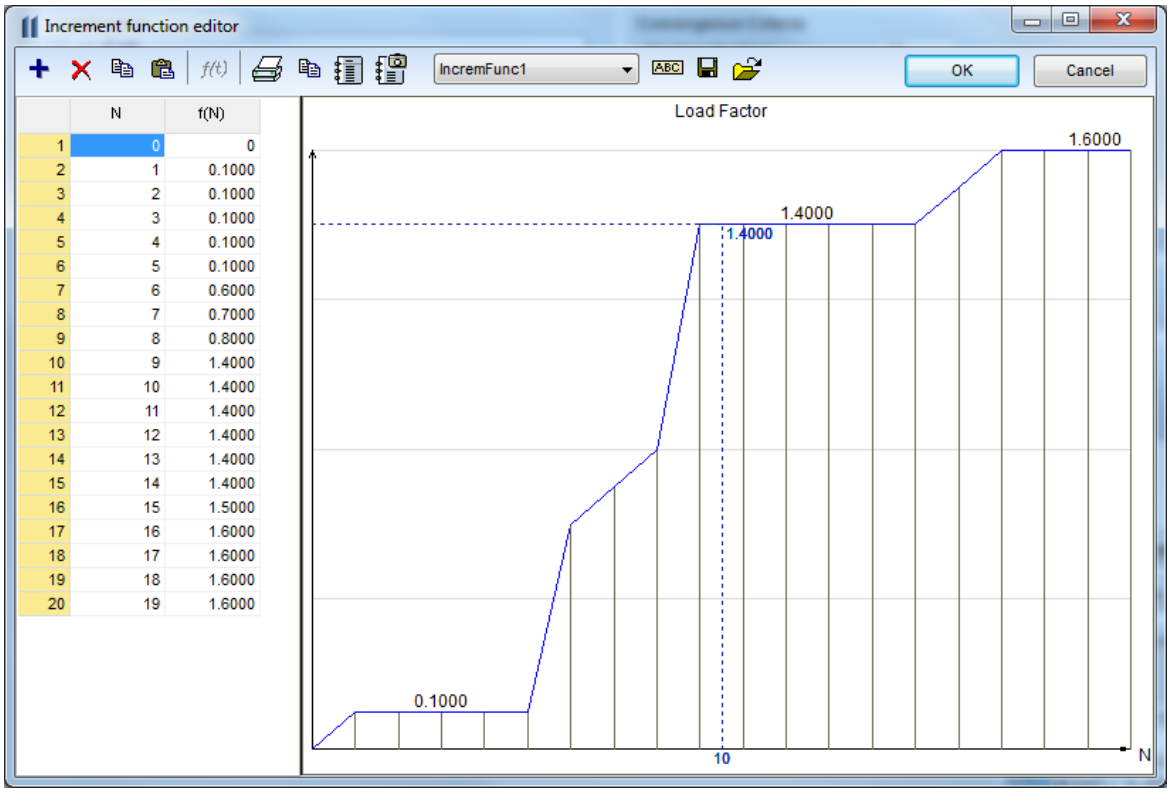
long **Add** ([in] BSTR **Name,** [in] SAFEARRAY(RPoint2D) **FunctionPoints**)

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** = {

ltAxisVM = 0x0,	*AxisVM layer*
ltDXF = 0x1 ,	*DXF layer, created with DXF import*
ltPDF= 0x2 }	*PDF layer, created with PDF import*

Layer types

enum **ELayerPenStyle** = {

lpsSolid = 0x0,	*solid line*
lpsDash = 0x1 ,	*solid line*
lpsDot = 0x2 ,	*doted line*
lpsDashDot= 0x3 ,	*dash dot line*
lpsDashDotDot= 0x4 }	*dash dot dot line*

Pen style

enum **ELayerShapeType** = {

lst3DLine = 0x0,	*3D line*
lst3DPolygon = 0x1 ,	*3D polygon*
lst3DPolygonFilled = 0x2 }	*closed 3D polygon*

Type Shape

enum **ETextHorizontalAlignment** = {

thaLeft = 0x0,	*left alignment*
thaCenter = 0x1 ,	*center alignment*
thaRight = 0x2 ,	*right alignment*
thaAligned = 0x3,	*custom alignment*
thaMiddle = 0x4 ,	*middle alignment*
thaFit = 0x5 }	*fitted alignment*

Horizontal text alignment

enum **ETextVerticalAlignment** = {

tvaBaseline = 0x0,	*aligned to baseline*
tvaBottom = 0x1 ,	*aligned to bottom*
tvaCenter = 0x2 ,	*aligned to center*
thaAligned = 0x3,	*aligned to baseline*
tvaTop = 0x4 }	*aligned to top*

Vertical text alignment

Error codes

enum **ELayerError** = {

laeInvalidName = –100001	*Emty string or name already exists*
laeInvalidPenWidth = –100002	*width should be more than 0 and less than or equal 2.11*
laeInvalidFontName = –100003	*Font not found*

}

Records / structures

RLayerShapeAttributes = (

long	**LayerIndex**	*index of the layer where shape is*
unsigned long	**Colour**	*see Colour. There are 3 special constants :* *-1 : ByLayer* *-2 : ByBlock (shouldn't be used)* *-3 : DefaultBlackWhite (i.e. on a light screen background it will choose black)*
ELayerPenStyle	**PenStyle**	*pen style*
double	**PenWidth**	*width of the pen*

)

RLayerTextParams = (

long	**LayerIndex**	*index of the layer*
unsigned long	**Colour**	*see Colour. There are 3 special constants :* *-1 : ByLayer* *-2 : ByBlock (shouldn't be used)* *-3 : DefaultBlackWhite (i.e. on a light screen background it will choose black)*
ELayerPenStyle	**PenStyle**	*pen style*
double	**PenWidth**	*width of the pen*
double	**FontSize**	*height of the text*
double	**Angle**	*in radians*
RPoint3D	**AlignmentPoint1**	*Alignment point 1*
RPoint3D	**AlignmentPoint2**	*Alignment point 2*
RPoint3D	**NormalVector**	*Normal vector*
ETextHorizontalAlignment	**HorizontalAlignment**	*Type of horizontal alignment*
ETextVerticalAlignment	**VerticalAlignment**	*Type of verical alignment*

)

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 *index of the text*

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 *index of the shape*

LayerShapeAttributes *attributes of the shape*

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 *index of the layer*

ShapeIDs *indexes of shapes on the layer*

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 *index of the shape*

Polygon *the polygon or line defining the shape*

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 *text index*

TextParams *parameters of the text*

Get paremeters 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 *index of the layer*

TextIDs *indexes of texts on the layer*

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 *index of the shape*

LayerShapeAttributes *attributes of the shape*

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 *text index*

TextParams *parameters of the text*

Set paremeters of the text. If successful, returns the text index otherwise an error code (see EGeneralError or ELayerError).

Properties

long **Count** *Get number of layers.*

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**.

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

IAxisVMLines

Lines of the model.

Note:

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

Enumerated types

enum **ELineGeomType** = {

lgtStraightLine = 0x0,	*straigth line*
lgtCircleArc = 0x1 }	*circle or arc*

Line geometry.

---

enum **ELineStiffnessReduction** = { *stiffness reduction factor components*

lsr_AX = 0,	*Ax of the cross-section area component*
lsr_AY = 1,	*Ay of the cross-section area component*
lsr_AZ = 2,	*Az of the cross-section area component*
lsr_IX = 3,	*Ix of the cross-section area component*
lsr_IY = 4,	*Iy of the cross-section area component*
lsr_IZ = 5 }	*Iz of the cross-section area component*

---

enum **EReleasePosType** = *{*

rptAuto = 0	*at the middle*
rptRatio = 1	*position is interpreted as a ratio [0..1]*
rptLength = 2 }	*position is interpreted as a distance [m]*

position type of the interface.

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)*

)

REccReleases = (

EReleaseType	**x**	*translational release along the x axis.Only rtRigid and rtHinged is allowed.*
EReleaseType	**y**	*translational release along the y axis.Only rtRigid and rtHinged is allowed.*
EReleaseType	**z**	*translational release along the z axis.Only rtRigid and rtHinged is allowed.*
EReleaseType	**xx**	*rotational release along the x axis.Only rtRigid and rtHinged is allowed.*
EReleaseType	**yy**	*rotational release along the y axis.Only rtRigid and rtHinged is allowed.*
EReleaseType	**zz**	*rotational release along the z axis.Only rtRigid and rtHinged is allowed.*
EReleasePosType	**PosType**	*position type of the release interface*
double	**Pos**	*position of the release interface, only for PosType rptRatio or rptLength. For ratio, it is a value between [0..1], for length, it is a value in [m])*

)

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 = (

RCircleArcGeomData	**CircleArc**	*arc geometry*
REllipseArcGeomData	**EllipseArc**	*ellipse arc geometry **(not used)***

)

RPoint3d = (

double	**x, y, z**	*x, y, z coordinates of a a 3D point or components of a 3D vector [m]*

)

RLineData = (

long	**NodeId1**	*start node id*
long	**NodeId2**	*end node id*
ELineGeomType	**GeomType**	*Type of line geometry (straight line, arc, …)*
RCircleArcGeomData	**CircleArc**	*arc geometry, if the GeomType is lgtCircleArc*
REllipseArcGeomData	**EllipseArc**	*ellipse arc geometry **(not used)***

)

RLineAttr = (

Warning! *This record was superseded by RLineAttr_V161*

ELineType	**LineType;**	*type of the line. Depending on it only the relevant parts of the record will be taken into account*
long	**MaterialIndex**	*material index*
long	**StartCrossSectionIndex**	*index of the starting cross section*
long	**EndCrossSectionIndex**	*index of the ending cross section*
EAutoExcentricityType	**AutoEccentricityType**	eccentricity type of ribs
RPoint3D	**StartEccentricity**	*starting eccentricity of ribs*
RPoint3D	**EndEccentricity**	*ending eccentricity of ribs*
ELineNonLinearity	**TrussType**	*nonlinear behaviour of the truss*
double	**Resistance**	*axial resistance (absolute value) only for TrussTypes of lnlTensionOnly and lnlCompressionOnly*
long	**ServiceClass**	*service class of timber (valid values are 1, 2, 3)*
double	**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)*
long	**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**	*gap type*
double	**ActiveStiffness**	*gap stiffness when active [kN/m]*
double	**InactiveStiffness**	*gap stiffness when inactive [kN/m]*
double	**InitialOpening**	*initial opening [m]*
double	**MinPenetration**	*minimum penetration [m]*
double	**MaxPenetration**	*maximum penetration [m]*
double	**AdjustmentRatio**	*adjustment ratio of the active stiffness*
ESpringDirection	**SpringDirection**	*coordinate system of spring stiffnesses*
RStiffnesses	**Stiffnesses**	*stiffnesses*

)

Not all fields are active at the same time. For a detailed walkthrough of the valid field combinations for RLineAttr, check out IAxisVMLines.BulkSetAttr.

RlineAttr_V161 = (

ELineType	**LineType;**	*type of the line. Depending on it only the relevant parts of the record will be taken into account*
long	**MaterialIndex**	*material index*
long	**StartCrossSectionIndex**	*index of the starting cross section*
long	**EndCrossSectionIndex**	*index of the ending cross section*
EMemberLocXOrientation	**LocalXOrientation**	*local x direction with respect to the node with the lower index*
RReleases_V161	**StartRelease**	*releases at the lower index node*
RReleases_V161	**EndRelease**	*releases at the higher index node*
long	**Reference**	*local z reference index, 0 for auto. Note : not all z references are valid for a given line.*
EAutoExcentricityType	**AutoEccentricityType**	*eccentricity type of ribs. Only used when StartEccentricityType and EndEccentricityType is leet_RibDomain*
RPoint3D	**StartEccentricity**	*starting custom eccentricity. Taken into account only for eccentricity types leet_RibDomain and leet_CustomOffset*
RPoint3D	**EndEccentricity**	*ending custom eccentricity. Taken into account only for eccentricity types leet_RibDomain and leet_CustomOffset*
ELEEccType	**StartEccentricityType**	*eccentricity type at the lower index node*
ELEEccType	**EndEccentricityType**	*eccentricity type at the higher index node*
ELEEccAlignementPoint	**StartAlignementPoint**	*alignement point on the cross section envelope box at the lower index node. Taken into account only for eccentricity types leet_AlignementPoint and leet_Group*
ELEEccAlignementPoint	**EndAlignementPoint**	*alignement point on the cross section envelope box at the higher index node. Taken into account only for eccentricity types leet_AlignementPoint and leet_Group*
long	**EccGroupIndex**	*eccentricity group index. Taken into account only for eccentricity type leet_Group*
long	**StartRefLine**	*reference line index at the lower index node. Taken into account only for eccentricity type leet_Ref*
long	**EndRefLine**	*reference line index at the higher index node. Taken into account only for eccentricity type leet_Ref*

ELEEccAlignementPoint	**RefStartAlignementPoint**	*alignement point on the cross section envelope box at the lower index node reference line. Taken into account only for eccentricity type leet_Ref*
ELEEccAlignementPoint	**RefEndAlignementPoint**	*alignement point on the cross section envelope box at the higher index node reference line. Taken into account only for eccentricity type leet_Ref*
REccReleases	**StartEccRelease**	*release data at the lower index node Taken into account only for eccentricity types leet_CustomOffset, leet_AlignementPoint, leet_Group, leet_Ref*
REccReleases	**EndEccRelease**	*release data at the higher index node Taken into account only for eccentricity types leet_CustomOffset, leet_AlignementPoint, leet_Group, leet_Ref*
ELineNonLinearity	**TrussType**	*nonlinear behaviour of the truss*
double	**Resistance**	*axial resistance (absolute value) only for TrussTypes of lnlTensionOnly and lnlCompressionOnly*
long	**ServiceClass**	*service class of timber (valid values are 1, 2, 3)*
double	**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)*
long	**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)*
ELongBoolean	**Beam7DOF**	*Taken into account only for LineType = ltBeam. Controls wether the beam is 7-DOF.*
long	**MaterialColor**	*color override for the material color. For no override use -1 (0xFFFFFFFF) or 0.*
long	**ContourColor**	*color override for the material color. For no override use -1 (0xFFFFFFFF).*
EGapType	**GapType**	*gap type*
double	**ActiveStiffness**	*gap stiffness when active [kN/m]*
double	**InactiveStiffness**	*gap stiffness when inactive [kN/m]*
double	**InitialOpening**	*initial opening [m]*
double	**MinPenetration**	*minimum penetration [m]*
double	**MaxPenetration**	*maximum penetration [m]*
double	**AdjustmentRatio**	*adjustment ratio of the active stiffness*
ESpringDirection	**SpringType**	*type of the spring*
RSpringCharacteristics	**SpringCharacteristics**	*stiffness indexes*
long	**SeismicIsolatorIndex**	*seismic isolator index in AxisVMSpringParams.*
double	**IsolatorD2**	*maximum design displacement at ULS [m] (only for seismic isolator)*

)

Not all fields are active at the same time. For a detailed walkthrough of the valid field combinations for RLineAttr_V161, check out IAxisVMMembers.BulkSetMembers_V161.

Functions

long **Add** ([in] long **StartNode,** [in] long **EndNode,** [in] ELineGeomType **GeomType,** [i/o] RLineGeomData **GeomData,**)

StartNode	*index of the startpoint (0 < StartNode ≤ AxisVMNodes.Count)*
EndNode	*index of the endpoint (0 < EndNode ≤ AxisVMNodes.Count)*
GeomType	*line geometry*
GeomData	*geometry data*

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

---

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

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

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

---

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

StartNode	*startpont coordinates*
EndNode	*endpont coordinates*

StartDOF *nodal degrees of freedom for the start node*
EndDOF *nodal degrees of freedom for the end node*
GeomType *line geometry*
GeomData *geometry data*

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

---

long **BulkAdd** ([in] SAFEARRAY (RLineData) **LineData,** [out] SAFEARRAY (long) *** LineIds**)

LineData *list of the line data records indexes of lines .The LineData NodeId1 and NodeId2 must satisfy: (1 ≤ NodeId ≤ AxisVMNodes.Count)*
LineIds *indexes of the created lines .Each index will satisfy: (1 ≤ Index ≤ AxisVMLines.Count)*

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).

---

long **BulkGetLineData** ([in] SAFEARRAY(long) **LineIds,** [out] SAFEARRAY (RLineData) *** LineData**)

LineIds *indexes of lines .Each index must satisfy: (1 ≤ Index ≤ AxisVMLines.Count)*
LineData *list of the line data records*

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).

---

long **BulkGetAttr** ([in] SAFEARRAY(long) **LineIds,** [out] SAFEARRAY (RLineAttr) *** LineAttr**)

Warning! *This function has become obsolete, was superseded by BulkGetAttr_V161*

LineIds *indexes of lines .Each index must satisfy: (1 ≤ Index ≤ AxisVMLines.Count)*
LineAttr *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 ltEdge, ltLLLink, …). In those cases however none of the other fields will contain relevant data, and calling BulkSetAttr with such records will result in an error.*

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).

---

long **BulkGetAttr_V161** ([in] SAFEARRAY(long) **LineIds,** [out] SAFEARRAY (RLineAttr_V161) *** LineAttr**)

LineIds *indexes of lines .Each index must satisfy: (1 ≤ Index ≤ AxisVMLines.Count)*
LineAttr *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_V161, check out IAxisVMMembers BulkSetMembers_V161. In addition to the LineTypes handled in IAxisVMLines.BulkSetAttr, the rest of the possible LineTypes are returned in BulkGetLineAttr (like ltEdge, ltLLLink, …). In those cases however none of the other fields will contain relevant data, and calling BulkSetAttr with such records will result in an error.*

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 BulkSetMembers_V161 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).

---

long **BulkGetMemberIds** ([in] SAFEARRAY (long) **LineIds,** [out] SAFEARRAY (long) *** MemberIds**)

LineIds *list of the line indexes to query. Each index must satisfy: (1 ≤ Index ≤ AxisVMLines.Count)*
MemberIds *list of the member indexes. Each index will satisfy: (0 ≤ Index ≤ AxisVMMembers.Count). 0 means the line is not part of a member.*

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).

---

long **BulkSetAttr** ([in] SAFEARRAY(long) **LineIds,** [in] SAFEARRAY (RLineAttr) **LineAttr**)

Warning! *This function has become obsolete, was superseded by BulkSetAttr_V161*

LineIds *indexes of lines .Each index must satisfy: (1 ≤ Index ≤ AxisVMLines.Count)*

LineAttr *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 allways has to be specified.Only the following subset of LineType is valid : ltSimpleLine, ltTruss, ltBeam, ltRib, ltSpring, ltGap.Trying to set for example a LineType of ltLLLink will result in an error (however querying an already existing line with a LineType of ltLLLink through BulkGetLineAttr is valid).*
LineType= ltSimpleLine : no other field is required
LineType=ltTruss : MaterialIndex, StartCrossSectionIndex=EndCrossSectionIndex (they must be equal), TrussType, Resistance. If MaterialIndex refers to timber, then also ServiceClass and kdef.
LineType=ltBeam : MaterialIndex, StartCrossSectionIndex, EndCrossSectionIndex. If MaterialIndex refers to timber, then also ServiceClass and kdef.
LineType=ltRib : 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.
LineType= ltSpring : SpringDirection, Stiffnesses.
LineType= ltGap : GapType, ActiveStiffness, InactiveStiffness, InitialOpening, MinPenetration, MaxPenetration, AdjustmentRatio.

If LineAttr MaterialIndex is active, it must satisfy: (1 ≤ MaterialIndex ≤ AxisVMMaterials.Count)
If LineAttr StartCrossSectionIndex is active, it must satisfy: (1 ≤ StartCrossSectionIndex ≤ AxisVMCrossSections.Count)
If LineAttr EndCrossSectionIndex is active, it must satisfy: (1 ≤ EndCrossSectionIndex ≤ AxisVMCrossSections.Count)
If LineAttr Domain1 is active, it must satisfy: (0 ≤ Domain1 ≤ AxisVMDomains.Count)
If LineAttr Domain2 is active, it must satisfy: (0 ≤ Domain2 ≤ AxisVMDomains.Count)
If LineAttr ServiceClass is active, it must be 1, 2 or 3

Sets the atributes 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= ltSimpleLine, 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, lneMaterialIndexOutOfBounds, lneCrossSectionIndexOutOfBounds, lneStartEndCrossSectionTypeIncompatible, lneIllegalServiceClassValue, lneDomainIndexOutOfBounds).

---

long **BulkSetAttr_V161** ([in] SAFEARRAY(long) **LineIds,** [in] SAFEARRAY (RLineAttr_V161) **LineAttr**)

LineIds *indexes of lines .Each index must satisfy: (1 ≤ Index ≤ AxisVMLines.Count)*

LineAttr *list of the line attribute records. Not all fields of the RLineAttr are active at the same time. For a detailed walkthrough of the valid field combinations for RLineAttr_V161, check out IAxisVMMembers BulkSetMembers_V161.*

Sets the atributes 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= ltSimpleLine, then set their attributes with this function.

Warning! *The fields related to eccentricities cannot be set reliably with this function. If eccentricities are to be used, always manage them through IAxisVMMembers BulkSetMembers_V161.*

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, lneMaterialIndexOutOfBounds, lneCrossSectionIndexOutOfBounds, lneStartEndCrossSectionTypeIncompatible, lneIllegalServiceClassValue, lneDomainIndexOutOfBounds, lneInvalidRefZ, lneReleaseFunctionIndexError, lneReleaseInvalidType, lneReleaseInvalidMaterial, lneReleaseInvalidComponent).

---

long **BulkSetLineData** ([in] SAFEARRAY(long) **LineIds,** [in] SAFEARRAY (RLineData) **LineData**)

LineIds *indexes of lines .Each index must satisfy: (1 ≤ Index ≤ AxisVMLines.Count)*
LineData *list of the line data records. The LineData NodeId1 and NodeId2 must satisfy: (1 ≤ NodeId ≤ AxisVMNodes.Count)*

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).

---

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

Index *line index (0 < Index ≤ 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 line index, otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBounds).

---

long **Clear**

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

---

long **CrossLines** ([in] SAFEARRAY(long) **LineIds**)

LineIds *indexes of lines to intersect*

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.

Please note: *Nodes, Lines etc. will be re-indexed. Use UID property if you want refer to them after this function.*

Possible error codes are errIndexOutOfBounds, errDatabaseNotReady,lneEmptyLineList.

---

long **CrossLines_vb** (Visual Basic compatible function of **CrossLines**)

---

long **DefineSelectedLinesAsRigidBody**

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

---

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

Index *index of the line to delete*

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

---

long **DeleteNameOfAllTrusses**

Deletes previously added default name of all trusses. If successful, returns number of lines. Otherwise returns an error code (errDatabaseNotReady).

---

long **DeleteSelected**

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

---

long **GetFiniteElementCount** ([in] ELineType **LineType**)

LineType *Type of member*

Get number of FE line elements (beams, trusses or ribs) if successful, otherwise returns an error code (errDatabaseNotReady).

---

long **GetContinuousLineIDs** ([i/o] SAFEARRAY(long)* **LineIDs,** [in] double **MaxAngleDifferenceX,** [in] double **MaxAngleDifferenceZ,** [out] 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*

Get indices of continuous line indices. If successful It returns the number of continuous lines, otherwise it returns an error code (errDatabaseNotReady, lneEmptyLineList, lneLinesNotContinuous).

---

long **GetMidpoint** ([in] long **Index,** [i/o] RPoint3d **Value**)

Index	*line index* *(0 < Line ≤ Count)*
Value	*midpoint coordinates*

Retrieves the midpoint coordinates of a line.
If successful, returns Index, otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBounds).

---

long **GetSelectedBeamIds** ([out] SAFEARRAY (long) * **LineIds**)

LineIds	*Index list of selected horizontal line elements*

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).

---

long **GetSelectedColumnIds** ([out] SAFEARRAY (long) * **LineIds**)

LineIds	*Index list of selected vertical line elements*

Get indexes of selected vertical line elements, If successful, returns the number of selected vertical line elements, otherwise returns an error code (errDatabaseNotReady).

---

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

ItemIds	*Index list of selected lines*

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

---

long **GetSelectedOtherIds** ([out] SAFEARRAY (long) * **LineIds**)

LineIds	*list of selected neither vertical or horizontal lines*

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).

---

long **GetSurfaces** ([in] long **LineId,** [out] SAFEARRAY (long) * **Surfaces**)

LineId	*line index. It must satisfy: (1 ≤ LineId ≤ AxisVMLines.Count)*
Surfaces	*list of surfaces having LineID as an edge. Each surface index will satisfy: (1 ≤ Index ≤ AxisVMSurfaces.Count)*

Retrieves the surfaces having LineID as an edge. If successful, returns the number of attached surfaces. The possible error codes are (errDatabaseNotReady, errIndexOutOfBounds, errInternalException, errOutOfMemory).

---

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

StartNode	*start node index of the line* *(0 < StartNode ≤ AxisVMNodes.Count)*
EndNode	*end node index of the line* *(0 < EndNode ≤ AxisVMNodes.Count)*

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).

---

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

StartNode	*start node index of the line (0 < StartNode ≤ AxisVMNodes.Count)*
EndNode	*end node index of the line (0 < EndNode ≤ 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 <= FiniteElementNumber <= GetFiniteElementCount. More info here.*

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

---

long **PointOnLine** ([in] SAFEARRAY(long) **LineIds,** [i/o] RPoint3d **Coord,** [in] double **Eps**)

LineIds	*indexes of lines .Each index must satisfy: (1 ≤ Index ≤ AxisVMLines.Count)*
Coord	*the coordinate to be tested*
Eps	*tolerance of testing. A good value is IAxisVMSettings.EditingTolerance*

The return value is the LineId on which the Coord is. If there are multiple lines on which the Coord is, the first such one is returned from the LineIds list. If no lines contain the Coord within the given Eps, the result is 0. In case of an error, a negative error code is returned (errDatabaseNotReady, errIndexOutOfBounds, errInternalException, errOutOfMemory).

---

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 horizontall or verical lines.
If Select is False, deselects all neither horizontall or verical lines.
If successful, returns the number of selected neither horizontall or verical 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*

If Select is True, selects all horizontal lines at storey.
If Select is False, deselects all horizontal lines at storey.
If successful, returns the number of selected horizontal 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 **SelectAllOthersAtStorey** ([in] ELongBoolean **Select,** [in] long **StoreyId**)

Select *selection state*

StoreyId *storey index*

If Select is True, selects all neither horizontall or verical lines at storey.
If Select is False, deselects all neither horizontall or verical lines at storey.
If successful, returns the number of selected neither horizontall or verical 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

---

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

Properties

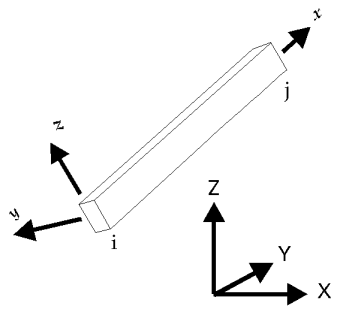
AxisVMAttachments*	**Attachments** *Get the attachments interface*
AxisVMAttributes*	**Attributes** *Get the attributes interface*
long	**Count** *number of lines in the model*
AxisVMLine*	**Item** [long **Index**] *line interface by index*
long	**IndexOfUID** [long **UID**] *Get index of the line*
	UID *unique index of the line*
long	**LineByMidpoint** [long **Index**] *Get the index of the line by the given midpoint index.* *(Index > AxisVMLines.Count)*
ELongBoolean	**LocalX_is_ij** [long **Index**] *Is lbTrue if local x axis of the line is in i-j direction* *(Index > AxisVMLines.Count)*
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*
ELongBoolean	**Selected** [long **Index**] • *Get or set the selection status of a line* *NOTE: Call Refresh function afterwards if not called between functions BeginUpdate and EndUpdate*
long	**SelCount** *number of selected lines in the model*
double	**StiffnessReduction**[long **Index,** ELineStiffnessReduction **Component**] • *Get or set stiffness reduction for the given component. Default value is 1. Used only for atLinearStatic and atLinearVibration analysis types. (only valid for Eurocode, SIA, NTS standards)*
double	**StiffnessReduction_A**[long **Index**] • ***Warning!*** *This function was extended by StiffnessReduction. 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 Eurocode, SIA, NTS standards)*
double	**StiffnessReduction_I**[long **Index**] • ***Warning!*** *This function was extended by StiffnessReduction. 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 Eurocode, SIA, NTS 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 = 0xFFFFFFFF }	*Colour depends on assigned material colour*

enum **EGapType** = {

agtActiveInTension = 0x0,	*gap active in tension only*
agtActiveInCompression = 0x1 }	*gap active in compression only*

Gap element type.

enum **ELineNonlinearity** = {

lnlTensionAndCompression = 0x0,	*linear behaviour*
lnlTensionOnly = 0x1,	*tension only*
lnlCompressionOnly = 0x2 }	*compression only*

Types of nonlinear behaviour.

enum **ELineType** = {

ltTruss = 0x0,	*truss*
ltBeam = 0x1,	*beam*
ltRib = 0x2,	*rib*
ltSpring = 0x3,	*spring*
ltGap = 0x4,	*gep element*
ltEdge = 0x5,	*surface edge*
ltHole = 0x6,	*hole edge*
ltSimpleLine = 0x7,	*line without element properites*
ltNNLink = 0x8,	*node-to-node link element*
ltLLLink = 0x9 }	*line-to-line link element*

Structural type of the line.

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

enum **EMemberLocXOrientation** = {

mlxo_ij = 1,	*the local x goes from the lower node index towards the higher node index*
mlxo_ji = 2 }	*the local x goes from the higher node index towards the lower node index*

local x orientation with respect to the node with the lower index.

enum **EReleaseType** = {

rtRigid = 0x0,	*rigid*
rtHinged = 0x1,	*hinged*
rtSemiRigid = 0x2,	*semi-rigid*
rtPlastic = 0x3,	*plastic*
rtPushover = 0x4 }	*for push over*

Type of end release.

enum **ESpringDirection** = {

sdGlobal = 0x0,	*global*
sdGeometry = 0x1,	*by geometry*
sdPointReference = 0x2,	*by reference point*
sdVectorReference = 0x3,	*by reference vector*
sdElementRelative = 0x4,	*by element*
sdNodeRelative = 0x5 }	*by node*

Coordinate system for spring stiffness components.

enum **EAutoExcentricityType** = *{*

aetCustom = 0x0,	*Custom eccentricity*
aetTop = 0x1,	*Bottom of the rib aligned with top of the domain*
aetMid = 0x2,	*Centre of the rib aligned with centre of the domain*
aetBottom = 0x3 }	*Top of the rib aligned with bottom of the domain*

Type of auto eccentricity for ribs

enum **ELEEccType** = *{*

leet_None = 0,	*no eccentricity*
leet_CustomOffset = 1,	*eccentricity values taken from the StartEccentricity/EndEccentricity fields*
leet_AlignementPoint = 2,	*eccentricity at an extremity of the cross-section envelope box*
leet_Group = 3,	*eccentricity aligned for all elements in the group. The alignement is specified in the StartAlignementPoint field.*
leet_Ref = 4,	*eccentricity relative to another line element.*
leet_RibDomain = 5 }	*rib aligned with a domain. See RibAutoEccentricityType.*

Types of line element eccentricities

enum **ELEEccAlignementPoint** = *{*

leeap_TopLeft = 1,	*alignement point is the top left point of the cross-section envelope box*
leeap_TopCenter = 2,	*alignement point is the top center point of the cross-section envelope box*
leeap_TopRight = 3,	*alignement point is the top right point of the cross-section envelope box*
leeap_CenterLeft = 4,	*alignement point is the center left point of the cross-section envelope box*
leeap_CenterCenter = 5,	*alignement point is the center of gravity of the cross-section*
leeap_CenterRight = 6,	*alignement point is the center right point of the cross-section envelope box*
leeap_BottomLeft = 7,	*alignement point is the bottom left point of the cross-section envelope box*
leeap_BottomCenter = 8,	*alignement point is the bottom center point of the cross-section envelope box*
leeap_BottomRight = 9 }	*alignement point is the bottom right point of the cross-section envelope box.*

alignement points for line element cross-sections

Error codes

enum **ELineError** = {

lneNodeIndexOutOfBounds = –100001	*node index is out of bounds*
lneReferenceIndexOutOfBounds = –100002	*reference index is out of bounds*
lneReadOnlyPropertyForThisLineType = –100003	*the property is read-only for this line type*
lnePropertyNotValidForThisLineType = –100004	*the property is not compatible with this line type*
lneMaterialIndexOutOfBounds = –100005	*material index is out of bounds*
lneCrossSectionIndexOutOfBounds = –100006	*cross-section index is out of bounds*
lneNoLinesAreSelected = –100007	*no lines are selected*
lneLineHasNoMidPoint = –100008	*line has no midpoint (i.e. it is not a rib or edge)*
lneEmptyLineList = –100009	*resulting line list is empty*
lneSectionIndexOutOfBounds = –100010	*invalid section index*
lneNotBeam = –100011	*the line is not a beam*
lneNotGap = –100012	*the line is not a gap element*
lneNotRib = –100013	*the line is not a rib element*
lneNotSpring = –100014	*the line is not a spring element*
lneNotTruss = –100015	*the line is not a truss element*
lneNodeNotOnLine = –100016	*node is not on the line*
lneErrorSplittingLine = –100017	*splitting lines was not successful l*
lneNMustBeGreaterThan1 = –100018	*spliting by value n was less than 1*
lneIllegalServiceClassValue = –100019	*service class value of the timber is incorrect*
lneDomainIndexOutOfBounds = –100020	*provided DomainID was ioncorrect using functions: IAxisVMLine.DefineAsRibWithAutoExcentricity and IAxisVMLine.DefineAsTimberRibWithAutoExcentricity*
lneStoreyIdOutOfBounds = –100021	*storey index invalid*
lneInvalidLineType = –100022	*invalid line type*
lneReinforcementParametersNotExsist = –100023	*reinforcement parameters not exsist*
lneInvalidColumnRebarsId = –100024	*invalid column rebar index*
lneInvalidConcreteMaterialId = –100025	*invalid concrete material index*
lneInvalidRebarSteelGradeId = –100026	*invalid rebar steel grade index*
lneInvalidRelease = –100027	*other release error*
lneInvalidLineTypeOrFEnumber = –100028	*invalid line type*
lneInvalidFunctionIDofRelease= –100029	*invalid function index related to the DOF*
lneReleaseInitAndLimitMustBe0= –100030	*Initial rotational stiffness and moment resistance of the release must be 0*
lneFunctionIdMustBe0= –100031	*FunctionId of the release must be 0*
lneLinesNotContinuous= –100032	*Lines in the array do not compose a continuous line*
lneStartEndCrossSectionTypeIncompatible = -100033	*can not generate various custom cross-sections*
lneInvalidRCCheckingParameters = -100034	*at least one of the RC checking parameters is invalid*
lneRCShrinkageEpsMustBePositive = -100035	*shrinkage strain must be positive*
lneStirrupParametersAreInvalid = -100036	*stirrup parameters are invalid*
lneShearCrackAngleIsInvalid = -100037	*defined shear crack angle is too low/high*
lneInvalidSteelMaterialId = -100038	*invalid steel grade index*
lneInvalidStiffnessReduction = -100039	*stiffness reduction should be > 0.00 and <= 1.00*
lneStiffnessReductionNotAllowed = -100040	*design code doesn't allow for stiffness reduction*
lneInvalidStiffnessReductionMat = -100041	*line's material doesn't allow for stiffness reduction*
lneReleaseFunctionIndexError = -100042	*invalid function index for a release*
lneReleaseInvalidType = -100043	*invalid type for a release*
lneInvalidRefZ = -100044	*invalid reference*
lneReleaseInvalidMaterial = -100045	*invalid material for a release (for now plastic hinges requires steel elements)*
lneReleaseInvalidComponent = -100046, **lneSpringTypeIncompatible** = -100047	*release component cannot be of the specified type i spring type isn't in accordance with its role (for example a displacement type spring for a rotational component)*
lneSpringIndexOutOfBounds = -100048}	*spring index is out of bounds of the valid spring indexes (1.. AxisVMSpringParams.Count)*

Records / structures

RLineGeomData = (

RCircleArcGeomData	**CircleArc**	*arc geometry*
REllipseArcGeomData	**EllipseArc**	*ellipse arc geometry* ***(not used)***

)

RPoint3d = (

double	**x, y, z**	*x, y, z coordinates of a a 3D point or components of a 3D vector [m]*

)

RRelease = (

Warning! This record has become obsolete, it was superseded by RRelease_V161

EReleaseType	**Release**	*release type (rigid / hinged / semi-rigid / plastic)*
double	**Init**	*(if Release = rtSemiRigid) initial rotational stiffness of the connection [kNm/rad]*
double	**Limit**	*(if Release = rtSemiRigid) moment resistance of the connection [kNm]*
long	**FunctionId**	*index of the used (pushover hinge) function, see here*

)

RReleases = (
Warning! This record has become obsolete, it was superseded by RReleases_V161

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

)

RRelease_V161 = (

EReleaseType	**Release**	*release type*
long	**FunctionId**	*index of the function. So far only two release types have function indexes : rtSemiRigid and rtPushover. For semi-rigid hinges it is an index in IAxisVMSpringParams. For pushover hinges it is an index in IAxisVMPushoverHingeFunctions.*

)

RReleases_V161 = (

RRelease_V161	**x**	*release in x direction. Valid types : rtRigid, rtHinged, rtSemiRigid. For steel elements rtPlastic is valid too.*
RRelease_V161	**y**	*release in y direction. Valid types : rtRigid, rtHinged, rtSemiRigid.*
RRelease_V161	**z**	*release in z direction. Valid types : rtRigid, rtHinged, rtSemiRigid.*
RRelease_V161	**xx**	*release around the x axis. Valid types : rtRigid, rtHinged, rtSemiRigid.*
RRelease_V161	**yy**	*release around the y axis. Valid types : rtRigid, rtHinged, rtSemiRigid, rtPushover. For steel elements rtPlastic is valid too.*
RRelease_V161	**zz**	*release around the z axis. Valid types : rtRigid, rtHinged, rtSemiRigid, rtPushover. For steel elements rtPlastic is valid too.*
RRelease_V161	**ew**	*warping release, valid only for 7-DOF elements. For 7-DOF elements the valid types are: rtRigid, rtHinged, rtSemiRigid*

)

RSpringCharacteristics = (

long	**x, y, z**	*stiffness index in AxisVMSpringParams for x, y, z translation. 0 is a valid index, it means no stiffness (denoted as -- in the user interface when defining a spring)*
long	**xx, yy, zz**	*stiffness index in AxisVMSpringParams for xx, yy, zz rotation. 0 is a valid index, it means no stiffness (denoted as -- in the user interface when defining a spring)*

)

RStiffnesses = (

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

)

Functions

long **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, if you want to set the eccentricity use IAxisVMMembers.BulkSetMembers_V161 instead)***
EndEccentricity	*eccentricity at the end node* ***(not used, if you want to set the eccentricity use IAxisVMMembers.BulkSetMembers_V161 instead)***

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 (IneCrossSectionIndexOutOfBounds, IneMaterialIndexOutOfBounds, 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*

Defines a line as a gap element.
If successful, returns the line index according to IAxisVMLines , 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**)

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, if you want to set the eccentricity use IAxisVMMembers.BulkSetMembers_V161 instead)***
EndEccentricity	*eccentricity at the end node* ***(not used, if you want to set the eccentricity use IAxisVMMembers.BulkSetMembers_V161 instead)***

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
If successful, returns the line index according to IAxisVMLines , otherwise returns an error code (IneCrossSectionIndexOutOfBounds, IneMaterialIndexOutOfBounds, errDatabaseNotReady, errIndexOutOfBounds).

---

long **DefineAsSpring** ([in] ESpringDirection **SpringDirection,** [i/o] RStiffnesses **Stiffnesses**)

SpringDirection	*coordinate system of spring stiffnesses*
Stiffnesses	*stiffnesses*

Defines a line as a spring element.
If successful, returns the line index according to IAxisVMLines , otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBounds).

---

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

MaterialIndex	*index of the material* *(0 < MaterialIndex ≤ AxisVMMaterials.Count)*
CrossSectionIndex	*index of the cross-section* *(0 < CrossSectionIndex ≤ AxisVMCrossSections.Count)*
TrussType	*nonlinear behaviour*
Resistance	*Axial resistance (absolute value) only for lnlTensionOnly and lnlCompressionOnly types of truss*

Defines a line as a truss element. 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 (IneCrossSectionIndexOutOfBounds, IneMaterialIndexOutOfBounds, errDatabaseNotReady, errIndexOutOfBounds).

---

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 lnlTensionOnly and lnlCompressionOnly 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, lneMaterialIndexOutOfBounds, lneCrossSectionIndexOutOfBounds, lneIllegalServiceClassValue).

---

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, lneMaterialIndexOutOfBounds, lneCrossSectionIndexOutOfBounds, lneIllegalServiceClassValue).

---

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, lneMaterialIndexOutOfBounds, lneCrossSectionIndexOutOfBounds, lneIllegalServiceClassValue).

---

long **DefineAsRibWithAutoExcentricity** ([in] long **MaterialIndex,** [in] long **StartCrossSectionIndex,**
[in] long **EndCrossSectionIndex,** [in] EAutoExcentricityType **AutoExcentricityType,**

[in] double **kx,** [in] long **Domain1,** [in] long **Domain2**)

MaterialIndex	*Material Index*
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)*
AutoExcentricityType	*Type of the automatic eccentricity*
kx	*Friction resistance between rib and surface*
Domain1	*Domain index (filled if connected to domain)*
Domain2	*Domain index (filled if rib is between two domains)*

Defines a line as a rib element, where eccentricity is calculated using domian thicknesses (Domain1,Domain2). 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, lneMaterialIndexOutOfBounds, lneCrossSectionIndexOutOfBounds, lneIllegalServiceClassValue, lneDomainIndexOutOfBounds).

---

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	*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)*
AutoExcentricityType	*Type of the automatic eccentricity*
kx	*Friction resistance between rib and surface*
Domain1	*Domain index (filled if connected to domain)*
Domain2	*Domain index (filled if rib is between two domains)*

Defines a line as a timber rib element, where eccentricity is calculated using domian thicknesses (Domain1,Domain2). 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, lneMaterialIndexOutOfBounds, lneCrossSectionIndexOutOfBounds, lneIllegalServiceClassValue, lneDomainIndexOutOfBounds).

---

long **DeleteColumnReinforcementParameters**

If successful, returns line index, otherwise returns an error code (errDatabaseNotReady).

---

long **DeleteLineElement**

Delete all assigned properties (cross-sections, materials, etc) from the line. If successful, returns line index, otherwise returns an error code (errDatabaseNotReady).

---

long **GetBeamData** ([out] long **MaterialIndex,** [out] long **StartCrossSectionIndex,**
[out] 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, if you want to get the eccentricity use IAxisVMMembers.BulkGetMembers_V161 instead)***

EndEccentricity *eccentricity at the end node* ***(not used, if you want to get the eccentricity use IAxisVMMembers.BulkGetMembers_V161 instead)***

Get properties of a beam. If successful, returns the line index according to IAxisVMLines, otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBounds, lneNotBeam).

---

long **GetColumnReinforcementParameters** ([i/o] RColumnReinforcementParameters **ColumnReinforcementParameters**)

ColumnReinforcementParameters *column reinforcement parameters*

If successful, returns the line index, otherwise returns an error code (errDatabaseNotReady, lneReinforcementParametersNotExsist, lneInvalidLineType).

---

long **GetEndReleases** ([i/o] RReleases **Value**)

Get the end releases (only if LineType = ltBeam or ltRib).
If successful, returns the line index according to IAxisVMLines, otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBounds).

---

long **GetGapData** ([out] EGapType **GapType,** [out] double **ActiveStiffness,** [out] double **InactiveStiffness,** [out] double **InitialOpening,** [out] double **MinPenetration,** [out] double **MaxPenetration,** [out] double **AdjustmentRatio**)

GapType *gap type*

ActiveStiffness *stiffness of the elemect when it is active*

InactiveStiffness *stiffness of the elemect when it is inactive*

InitialOpening *if GapType = agtActiveInTension: initial penetration*
if GapType = agtActiveInCompression: initial opening

MinPenetration *if auto active stiffness adjustment is on stiffness is reduced if penetration is under this value*

MaxPenetration *if auto active stiffness adjustment is on stiffness is multiplied if penetration is above this value*

AdjustmentRatio *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.

Get properties of a gap element. If successful, returns the line index according to IAxisVMLines, otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBounds, lneNotGap).

---

long **GetGeomData** ([i/o] RLineGeomData **Value**)

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, lnePropertyNotValidForThisLineType).

long **GetMidpoint** ([i/o] RPoint3d **Value**)

Retrieves the midpoint coordinates.
If successful, returns the line index according to IAxisVMLines, otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBounds).

---

long **GetRibData** ([out] long **MaterialIndex,** [out] long **StartCrossSectionIndex,** [out] 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 beginning* ***(not used, if you want to get the eccentricity use IAxisVMMembers.BulkGetMembers_V161 instead)***

EndEccentricity *eccentricity at the end* ***(not used, if you want to get the eccentricity use IAxisVMMembers.BulkGetMembers_V161 instead)***

Get properties of a rib. If successful, returns the line index according to IAxisVMLines, otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBounds, lneNotRib).

---

long **GetRibDataWithAutoExcentricity** ([out] long **MaterialIndex,**
[out] long **StartCrossSectionIndex,** [out] long **EndCrossSectionIndex,**
[out] EAutoExcentricityType **AutoExcentricityType,** [out] double **kx,** [out] long **Domain1,**
[out] long **Domain2**)

MaterialIndex *Material Index*
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)*
AutoExcentricityType *Type of the automatic eccentricity*
kx *Friction resistance between rib and surface*
Domain1 *Domain index (filled if connected to domain)*
Domain2 *Domain index (filled if rib is between two domains)*

If successful, returns line index, otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBounds,lneNotRib).

---

long **GetSpringData** ([out] ESpringDirection **SpringDirection,** [out] RStiffnesses **Stiffnesses**)

SpringDirection *spring direction*
Stiffnesses *spring stiffnesses*

Get properties of a spring. If successful, returns the line index according to IAxisVMLines, otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBounds, lneNotSpring).

---

long **GetStartReleases** ([i/o] RReleases **Value**)

Get the start releases (only if LineType = ltBeam or ltRib).
If successful, returns the line index according to IAxisVMLines, otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBounds).

---

long **GetTrussData** ([out] long **MaterialIndex,** [out] long **CrossSectionIndex,**
[out] ELineNonlinearity **TrussType,** [out] double **Resistance**)

MaterialIndex *index of the material*
(0 < MaterialIndex ≤ AxisVMMaterials.Count)
CrossSectionIndex *index of the cross-section*
(0 < CrossSectionIndex ≤ AxisVMCrossSections.Count)
TrussType *nonlinear behaviour of the truss*
Resistance *Axial resistance (absolute value) only for lnlTensionOnly and lnlCompressionOnly types of truss*

Get properties of a truss. If successful, returns the line index according to IAxisVMLines, otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBounds, lneNotTruss).

---

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 lnlTensionOnly and lnlCompressionOnly types of truss*

If successful, returns line index, otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBounds, lneNotTruss).

---

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)*

If successful, returns line index, otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBounds, lneNotBeam).

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)*

If successful, returns line index, otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBounds, lneNotRib).

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	*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)*
AutoExcentricityType	*Type of the automatic eccentricity*
kx	*Friction resistance between rib and surface*
Domain1	*Domain index (filled if connected to domain)*
Domain2	*Domain index (filled if rib is between two domains)*

If successful, returns line index, otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBounds,lneNotRib).

long **SetColumnReinforcementParameters** ([i/o] RColumnReinforcementParameters **ColumnReinforcementParameters**)

ColumnReinforcementParameters	*column reinforcement parameters*

If successful, returns the line index, otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBounds, lneInvalidConcreteMaterialId, lneInvalidRebarSteelGradeId, lneInvalidConcreteMaterialId, lneInvalidColumnRebarsId, lneInvalidLineType).

long **SetEndReleases** ([i/o] RReleases **Value**)

Set the end releases (only if LineType = ltBeam or ltRib).
If successful, returns the line index according to IAxisVMLines, otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBounds).

long **SetGeomData** ([i/o] RLineGeomData **Value**)

Set 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, lnePropertyNotValidForThisLineType).

long **SetGeomType** ([in] ELineGeomType **GeomType,** [i/o] RLineGeomData **Value**)

Set the geometry type and the data for the arc (only if GeomType = lgtCircleArc).
If successful, returns the line index according to IAxisVMLines, otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBounds, lnePropertyNotValidForThisLineType).

long **SetStartReleases** ([i/o] RReleases **Value**)

Set the start releases (only if LineType = ltBeam or ltRib).
If successful, returns the line index according to IAxisVMLines, otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBounds).

long **SplitByNode** ([in] long **NodeId**, [out] long **LineId1**, [out] long **LineId2**)

NodeId *Index of node which will dictate the location of split*
LineId1 *Line index of first part after split*
LineId2 *Line index of second part after split*

If successful, returns 1, otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBounds, lneNodeIndexOutOfBounds, lneNodeNotOnLine, lneErrorSplittingLine). After a successful SplitByNode the IAxisVMLine interface will become invalid (the original line gets deleted), it shouldn't be used for further interaction.

long **SplitByN** ([in] long **N**)

N Number of parts after line split

If successful, returns 1, otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBounds, lneNMustBeGreaterThan1.). 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 *leReferenceIndexOutOfBounds*.

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 horizontall or verical (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 = ltTruss)*

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**[ELineStiffnessReduction **Component**] •

Get or set stiffness reduction for the given component. Default value is 1. Used only for atLinearStatic and atLinearVibration analysis types. (only valid for Eurocode, SIA, NTS standards)

double **StiffnessReduction_A**

Warning! *This function was extended by StiffnessReduction. 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 Eurocode, SIA, NTS standards)*

double **StiffnessReduction_I**

Warning! *This function was extended by StiffnessReduction. 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 Eurocode, SIA, NTS standards)*

long **StoreyId** *Get storey index of the line*

double **TrussResistance** • *Get or set (only if LineType = ltTruss) resistance of the truss [kN] (absolute value) only for lnlTensionOnly and lnlCompressionOnly 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

RNonlinearity = (

ELineNonlinearity	**x, y, z,**	*nonlinear behaviour in x, y, z direction*
	xx, yy, zz	*and around the x, y, z axis*

)

RNonlinearityXYZ = (

ELineNonlinearity	**x, y, z**	*nonlinear behaviour in x, y, z direction*

)

RResistances = (

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

)

RResistancesXYZ = (

double	**x, y, z**	*resistances in x, y, z direction [kN/m]*

)

RStiffnesses = (

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

)

RStiffnessesXYZ = (

double	**x, y, z**	*stiffnesses in x, y, z direction [kN/m/m]*

)

NOTE: Direction of nonlinearity, stiffness and resistance depends on line support type ELineSupportType

RBulkLineSupport = (

ELineSupportType	**SupportType**	*type of the support*
long	**LineId**	*identifier of the line (1 ≤ LineId ≤ AxisVMLines.Count)*
RStiffnesses	**Stiffnesses**	*stiffnesses components. For support types lstRibElasticFoundation, lstBeamElasticFoundation, only the first 3 components are taken into account : x, y, z*
RNonLinearity	**NonLinearity**	*nonlinearity components. For support types lstRibElasticFoundation, lstBeamElasticFoundation, only the first 3 components are taken into account : x, y, z*
RResistances	**Resistances**	*resistance components. For support types lstRibElasticFoundation, lstBeamElasticFoundation, only the first 3 components are taken into account : x, y, z*
long	**SurfaceId1**	*surface index 1. Only for support types lstEdgeGlobal, lstEdgeRelative, lstEdgeReference. If it is specified, DomainId1 must be 0.(0 ≤ SurfaceId1 ≤ AxisVMSurfaces.Count)*
long	**SurfaceId2**	*surface index 2. Only for support types lstEdgeGlobal, lstEdgeRelative, lstEdgeReference. Must be zero if only one surface is connecting to the support. .If it is specified, DomainId1, DomainId2 must be 0. (0 ≤ SurfaceId2 ≤ AxisVMSurfaces.Count)*
long	**DomainId1**	*domain index 1. Only for support types lstEdgeGlobal, lstEdgeRelative, lstEdgeReference. If it is specified, SurfaceId1, SurfaceId2 must be 0. (0 ≤ DomainId1 ≤ AxisVMDomains.Count)*
long	**DomainId2**	*domain index 2. Only for support types lstEdgeGlobal, lstEdgeRelative, lstEdgeReference. Must be zero if only one domain is connecting to the support.(0 ≤ DomainId2 ≤ AxisVMDomains.Count)*
long	**ReferenceId**	*reference index. For support type lstEdgeReference it must be greater than zero, for the other support types it must be zero. (0 ≤ ReferenceId ≤ AxisVMReferences.Count)*

)

RBulkWSLineSupport = (

ELineSupportType	**SupportType**	*type of the support*
long	**LineId**	*identifier of the line (1 ≤ LineId ≤ AxisVMLines.Count)*
RStiffnesses	**Stiffnesses**	*stiffnesses components. For support types lstRibElasticFoundation, lstBeamElasticFoundation, only the first 3 components are taken into account : x, y, z*
RNonLinearity	**NonLinearity**	*nonlinearity components. For support types lstRibElasticFoundation, lstBeamElasticFoundation, only the first 3 components are taken into account : x, y, z*
RResistances	**Resistances**	*resistance components. For support types lstRibElasticFoundation, lstBeamElasticFoundation, only the first 3 components are taken into account : x, y, z*
double	**ShearStiffness**	*shear stiffness of the support (noted as RG on the window for supports) [kN]. If it is 0, a traditional Winkler support will be created*

long	**SurfaceId1**	*surface index 1. Only for support types lstEdgeGlobal, lstEdgeRelative, lstEdgeReference. If it is specified, DomainId1 must be 0.(0 ≤ SurfaceId1 ≤ AxisVMSurfaces.Count)*
long	**SurfaceId2**	*surface index 2. Only for support types lstEdgeGlobal, lstEdgeRelative, lstEdgeReference. Must be zero if only one surface is connecting to the support. .If it is specified, DomainId1, DomainId2 must be 0. (0 ≤ SurfaceId2 ≤ AxisVMSurfaces.Count)*
long	**DomainId1**	*domain index 1. Only for support types lstEdgeGlobal, lstEdgeRelative, lstEdgeReference. If it is specified, SurfaceId1, SurfaceId2 must be 0. (0 ≤ DomainId1 ≤ AxisVMDomains.Count)*
long	**DomainId2**	*domain index 2. Only for support types lstEdgeGlobal, lstEdgeRelative, lstEdgeReference. Must be zero if only one domain is connecting to the support.(0 ≤ DomainId2 ≤ AxisVMDomains.Count)*
long	**ReferenceId**	*reference index. For support type lstEdgeReference it must be greater than zero, for the other support types it must be zero. (0 ≤ ReferenceId ≤ AxisVMReferences.Count)*

)

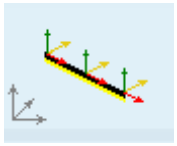
Error codes

enum **ELineSupportsError** = {

IseSectionIdOutOfBounds = –100001	*index of the section is out of range*
IsePadFootingNotDefined = –100002	*line support has no pad footing defined*
IseMaterialIndexOutOfBounds = –100003}	*material index is out of bounds*

Functions

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

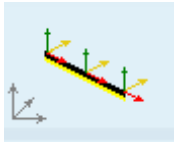


BeamId	*index of the beam* *(0 < BeamId ≤ AxisVMLines.Count)*
StiffnessesXYZ	*stiffnesses of the elastic foundation in local direction*
NonlinearityXYZ	*nonlinear behaviour of the elastic foundation in local direction*
ResistancesXYZ	*resistances for stiffness components in local direction*

*Adds an elastic foundation to a beam in beam's coordination system. If successful, returns the support index, otherwise returns an error code (leInvalidLineType, if the line is not a beam or errDatabaseNotReady, errIndexOutOfBounds). The length of the beam should be smaller than 0.5 * min((4*Ex*Iz/stiffnes.y)^0.25, (4*Ex*Iy/stiffnes.z)^0.25)*

---

long **AddBeamPasternakFoundation** ([in] long **BeamId,**
[i/o] RStiffnessesXYZ **StiffnessesXYZ,** [in] double **ShearStiffness,** [i/o]
RNonLinearityXYZ **NonlinearityXYZ,** [i/o] RResistancesXYZ **ResistancesXYZ**)

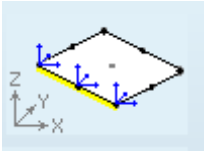


BeamId	*index of the beam* *(0 < BeamId ≤ AxisVMLines.Count)*
StiffnessesXYZ	*stiffnesses of the elastic foundation in local direction*
ShearStiffness	*shear stiffness of the support (noted as R_G on the window for supports) [kN]*
NonlinearityXYZ	*nonlinear behaviour of the elastic foundation in local direction*
ResistancesXYZ	*resistances for stiffness components in local direction*

*Adds a Winkler-Pasternak foundation to a beam in beam's coordination system. If successful, returns the support index, otherwise returns an error code (leInvalidLineType, if the line is not a beam or errDatabaseNotReady, errIndexOutOfBounds). The length of the beam should be smaller than 0.5 * min((4*Ex*Iz/stiffnes.y)^0.25, (4*Ex*Iy/stiffnes.z)^0.25)*

---

long **AddEdgeGlobal** ([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**)



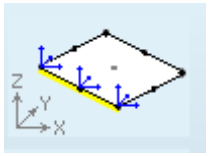
Stiffnesses	*stiffness components in the global system*
Nonlinearity	*nonlinear behaviour of the support*
Resistances	*resistances for stiffness components*

EdgeId *index of the edge*
(0 < EdgeId ≤ AxisVMLines.Count)
SurfaceId1 *first connecting surface*
(0 < SurfaceId1 ≤ AxisVMSurfaces.Count)
SurfaceId2 *second connecting surface*
(0 < SurfaceId2 ≤ AxisVMSurfaces.Count)
DomainId1 *first connecting domain*
(0 < DomainId1 ≤ AxisVMDomains.Count)
DomainId2 *second connecting domain*
(0 < DomainId2 ≤ AxisVMDomains.Count)

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, errIndexOutOfBounds).

---

long **AddEdgeGlobalPasternak** ([i/o] RStiffnesses **Stiffnesses,** [in] double **ShearStiffness,** [i/o] RNonLinearity **Nonlinearity,** [i/o] RResistances **Resistances,** [in] long **EdgeId,** [in] long **SurfaceId1,**
[in] long **SurfaceId2,** [in] long **DomainId1,** [in] long **DomainId2**)

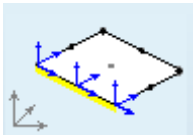


Stiffnesses *stiffness components in the global system*
ShearStiffness *shear stiffness of the support (noted as R_G on the window for supports) [kN]*
Nonlinearity *nonlinear behaviour of the support*
Resistances *resistances for stiffness components*
EdgeId *index of the edge*
(0 < EdgeId ≤ AxisVMLines.Count)
SurfaceId1 *first connecting surface*
(0 < SurfaceId1 ≤ AxisVMSurfaces.Count)
SurfaceId2 *second connecting surface*
(0 < SurfaceId2 ≤ AxisVMSurfaces.Count)
DomainId1 *first connecting domain*
(0 < DomainId1 ≤ AxisVMDomains.Count)
DomainId2 *second connecting domain*
(0 < DomainId2 ≤ AxisVMDomains.Count)

Adds a Winkler-Pasternak 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, errIndexOutOfBounds).

---

long **AddEdgeRelative** ([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**)



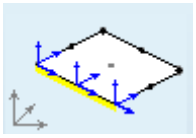
Stiffnesses *stiffness components in the local system of the edge*
Nonlinearity *nonlinear behaviour of the support*
Resistances *resistances for stiffness components*
EdgeId *index of the edge*
(0 < EdgeId ≤ AxisVMLines.Count)
SurfaceId1 *first connecting surface*
(0 < SurfaceId1 ≤ AxisVMSurfaces.Count)
SurfaceId2 *second connecting surface*
(0 < SurfaceId2 ≤ AxisVMSurfaces.Count)
DomainId1 *first connecting domain*
(0 < DomainId1 ≤ AxisVMDomains.Count)
DomainId2 *second connecting domain*
(0 < DomainId2 ≤ AxisVMDomains.Count)

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, errIndexOutOfBounds).

---

long **AddEdgeRelativePasternak** ([i/o] RStiffnesses **Stiffnesses,** [in] double **ShearStiffness,** [i/o] RNonLinearity **Nonlinearity,** [i/o] RResistances **Resistances,** [in] long **EdgeId,** [in] long **SurfaceId1,** [in] long **SurfaceId2,** [in] long **DomainId1,** [in] long **DomainId2**)



Stiffnesses	*stiffness components in the local system of the edge*
ShearStiffness	*shear stiffness of the support (noted as R_G on the window for supports) [kN]*
Nonlinearity	*nonlinear behaviour of the support*
Resistances	*resistances for stiffness components*
EdgeId	*index of the edge (0 < EdgeId ≤ AxisVMLines.Count)*
SurfaceId1	*first connecting surface (0 < SurfaceId1 ≤ AxisVMSurfaces.Count)*
SurfaceId2	*second connecting surface (0 < SurfaceId2 ≤ AxisVMSurfaces.Count)*
DomainId1	*first connecting domain (0 < DomainId1 ≤ AxisVMDomains.Count)*
DomainId2	*second connecting domain (0 < DomainId2 ≤ AxisVMDomains.Count)*

Adds a Winkler-Pasternak 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, errIndexOutOfBounds).

---

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

RibId	*index of the rib (0 < RibId ≤ AxisVMLines.Count)*
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 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, errIndexOutOfBounds). 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)*

---

long **AddRibPasternakFoundation** ([in] long **RibId,** [i/o] RStiffnessesXYZ **StiffnessesXYZ,** [in] double **ShearStiffness,** [i/o] RNonLinearityXYZ **NonlinearityXYZ,** [i/o] RResistancesXYZ **ResistancesXYZ**)

RibId	*index of the rib (0 < RibId ≤ AxisVMLines.Count)*
StiffnessesXYZ	*stiffnesses of the elastic foundation in local direction*
ShearStiffness	*shear stiffness of the support (noted as R_G on the window for supports) [kN]*
NonlinearityXYZ	*nonlinear behaviour of the elastic foundation in local direction*
ResistancesXYZ	*resistances for stiffness components in local direction*

*Adds a Winkler-Pasternak 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, errIndexOutOfBounds). 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)*

---

long **BulkAdd** ([in] SAFEARRAY(RBulkLineSupport) **LineSupports,**[out] SAFEARRAY(long)* **Indexes**)

LineSupports *list of support records to be created*
Indexes *list of the created support indexes. If a particular support couldn't be created, its corresponding index will be 0*

Creates multiple supports in one step. It is faster than caling the respective singular functions successively. Returns the number created member supports, which can be zero if none were created.

It can trigger the following events in case of an error (some of them may be triggered on a by record case) :

IAxisVMLineSupportsEvents.Error, with the errorcodes (errDatabaseNotReady)

IAxisVMLinesEvents.Error, with the errorcodes (errIndexOutOfBounds, leInvalidLineType)

---

long **BulkAddPasternak** ([in] SAFEARRAY(RBulkWSLineSupport) **LineSupports,**[out] SAFEARRAY(long)* **Indexes**)

LineSupports *list of support records to be created*
Indexes *list of the created support indexes. If a particular support couldn't be created, its corresponding index will be 0*

Creates multiple supports in one step. If the value of the field ShearStiffness is set to 0 in a record, a traditional Winkler support is created for that. Thus, BulkAddPasternak can be used to create mixed Winkler, Winkler-Pasternak supports. It is faster than caling the respective singular functions successively. Returns the number created member supports, which can be zero if none were created.

It can trigger the following events in case of an error (some of them may be triggered on a by record case) :

IAxisVMLineSupportsEvents.Error, with the errorcodes (errDatabaseNotReady)

IAxisVMLinesEvents.Error, with the errorcodes (errIndexOutOfBounds, leInvalidLineType)

---

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

Index *support index*

Deletes the line support indexed by **Index**. *If successful, returns the line support index otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBounds).*

---

long **DeleteSelected**

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

---

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

ItemIds *list of selected line supports*

If successful, returns the number of selected elements otherwise returns an error code

---

long **GetNodeIds** ([in] long **Index,** [out] long **StartNodeID,** [out] long **EndNodeID**)

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*

ELongBoolean **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*

ELongBoolean **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** = {

lstEdgeGlobal = 0x0,	*global edge support*
lstEdgeRelative = 0x1,	*edge relative support*
lstRibElasticFoundation = 0x2,	*elastic foundation of a rib*
lstBeamElasticFoundation = 0x3,	*elastic foundation of a beam*
lstEdgeReference = 0x4 }	*edge support, loc. z = reference*

Line support type

NOTE: Direction of nonlinearity, stiffness and resistance depends on line support type ELineSupportType

Records / structures

RWallStiffnessParams = (

RElementStiffnessParams	**CalcParams**	*common calculation parameters*
double	**WallThickness**	*thickness of the supporting element (wall)*

)

RLineSupportStiffParams = (

RWallStiffnessParams	**Top**	*calculation parameters of the supporting element (wall) above support*
RWallStiffnessParams	**Bottom**	*calculation parameters of the supporting element (wall) below support*

)

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 parametres 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 SupportType = lstEdgeGlobal or lstEdgeRelative). 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 SupportType = lstBeamElasticFoundation or lstRibElasticFoundation). If successful, returns the index of the line support, otherwise returns an error code(errIndexOutOfBounds, errDatabaseNotReady).

---

long **GetPasternakStiffness** ([i/o] double* **Value)**

Value *shear stiffness of the line support (noted as R_G on the window for supports) [kN]*

If successful, returns the index of the domain support otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBounds).

long **GetResistance** ([i/o] RResistances **Value**)

Get the resistance components of the line support (only if SupportType = lstEdgeGlobal or lstEdgeRelative). If successful, returns the index of the line support, otherwise returns an error code (errIndexOutOfBounds, errDatabaseNotReady).

long **GetResistanceXYZ** ([i/o] RResistancesXYZ **Value**)

Get the resistance components of the line support (only if SupportType = lstBeamElasticFoundation or lstRibElasticFoundation). If successful, returns the index of the line support, otherwise returns an error code (errIndexOutOfBounds, errDatabaseNotReady).

long **GetStiffnesses** ([i/o] RStiffnesses **Value**)

Get the stiffness components of the line support (only if SupportType = lstEdgeGlobal or lstEdgeRelative). If successful, returns the index of the line support, otherwise returns an error code (errIndexOutOfBounds, errDatabaseNotReady).

long **GetStiffnessesXYZ** ([i/o] RStiffnessesXYZ **Value**)

Get the stiffness components of the line support (only if SupportType = lstBeamElasticFoundation or lstRibElasticFoundation). If successful, returns the index of the line support, otherwise returns an error code (errIndexOutOfBounds, errDatabaseNotReady).

long **GetStiffnessCalcParams** ([i/o] RLineSupportStiffParams **Value**)

Get calculation parameters for calculating the stiffness of the line support. If unsuccessful, returns an error code(errIndexOutOfBounds, errDatabaseNotReady).

long **SetNonLinearityXYZ** ([i/o] RNonlinearityXYZ **Value**)

Set the nonlinear behaviour of the elastic foundation (only if SupportType = lstBeamElasticFoundation or lstRibElasticFoundation). If successful, returns the index of the line support, otherwise returns an error code(errIndexOutOfBounds, errDatabaseNotReady).

long **SetPasternakStiffness** ([i/o] double* **Value)**

Value *shear stiffness of the line support (noted as R_G on the window for supports) [kN]*

If successful, returns the index of the domain support otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBounds).

long **SetResistance** ([i/o] RResistances **Value**)

Set the resistance components of the line support (only if SupportType = lstEdgeGlobal or lstEdgeRelative). If successful, returns the index of the line support, otherwise returns an error code (errIndexOutOfBounds, errDatabaseNotReady).

long **SetResistanceXYZ** ([i/o] RResistancesXYZ **Value**)

Set the resistance components of the line support (only if SupportType = lstBeamElasticFoundation or lstRibElasticFoundation). If successful, returns the index of the line support, otherwise returns an error code (errIndexOutOfBounds, errDatabaseNotReady).

long **SetStiffnesses** ([i/o] RStiffnesses **Value**)

Set the stiffness components of the line support (only if SupportType = lstEdgeGlobal or lstEdgeRelative). If successful, returns the index of the line support, otherwise returns an error code (errIndexOutOfBounds, errDatabaseNotReady).

long **SetStiffnessesXYZ** ([i/o] RStiffnessesXYZ **Value**)

Set the stiffness components of the line support (only if SupportType = lstBeamElasticFoundation or lstRibElasticFoundation). If successful, returns the index of the line support, otherwise returns an error code (errIndexOutOfBounds, errDatabaseNotReady).

long **SetStiffnessCalcParams** ([i/o] RLineSupportStiffParams **Value**)

Set calculation parameters for calculating the stiffness of the line support. If unsuccessful, returns an error code(errIndexOutOfBounds, errDatabaseNotReady, lseMaterialIndexOutOfBounds).

Properties

If the support is an *lstRibElasticFoundation* or an *lstBeamElasticFoundation,* reading a property valid only for *lstEdgeGlobal* or *lstEdgeRelative* supports returns 0 and an attempt to write creates an error event.

If the support is an *lstEdgeGlobal* or an *lstEdgeRelative,* reading a property valid only for *lstRibElasticFoundation* or *lstBeamElasticFoundation* supports returns 0 and an attempt to write creates an error event.

long **DomainId1** *(SupportType = lstEdgeGlobal or lstEdgeRelative) first domain connecting to the edge*

long **DomainId2** *(SupportType = lstEdgeGlobal or lstEdgeRelative) second domain connecting to the edge*

long **EdgeId** *(SupportType = lstEdgeGlobal or lstEdgeRelative) edge index according to IAxisVMLines*

EPadFootingType **FootingType** *Type of pad footing if it has been defined in AxisVM*

ELongBoolean **HasFooting** *lbTrue 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 < SectionId ≤ SectionCount[AnalysisType])*

ELineSupportType **SupportType** *line support type*

long **SurfaceId1** *(SupportType = lstEdgeGlobal or lstEdgeRelative) first surface connecting to the edge*

long **SurfaceId2** *(SupportType = lstEdgeGlobal or lstEdgeRelative) 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

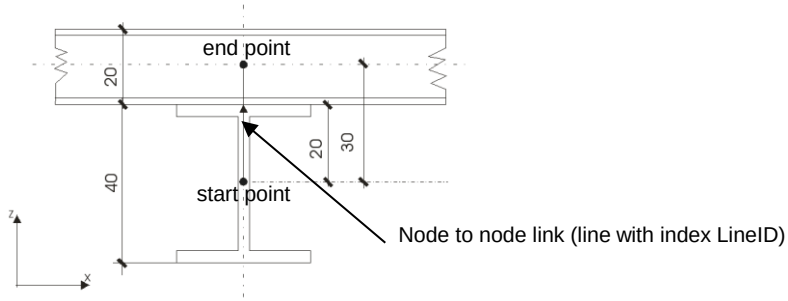
enum **ELinkElementError:** = {

leeLineIndexOutOfBounds = –100001	*line index invalid*
leeInvalidSystemType = –100002	*Invalid SystemGLR during seting 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 = -100007	*master/slave line and and master start/end link not connected*
leeNotConnectingMasterLineAndMasterEndLink = -100008	*master/slave line and and master start/end link not connected*
leeNotConnectingSlaveLineAndMasterStartLink = -100009	*master/slave line and and master start/end link not connected*
leeNotConnectingSlaveLineAndMasterEndLink = -100010}	*master/slave line and and master start/end link not 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

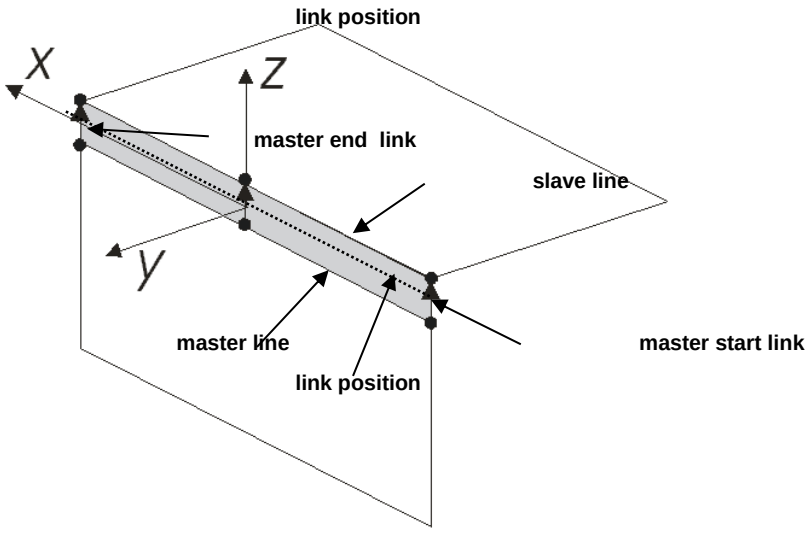
RNNLinkElementRec = (

long	**LineID**	*Index of the line where NN link will be created*
ESystem	**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)*
EBeamRibDistributionType	**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)*
RStiffnesses	**Stiffnesses**	*stiffness of the NN link*
RResistances	**Resistances**	*resistance of the NN link*
RNonLinearity	**NonLinearity**	*nonlinearity of the NN link*

)

Line to Line link

Example: A wall to slab connection. The master line is 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 inter-face 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

RLLLinkElementRec = (

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*
EBeamRibDistributionType	**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	**Stiffnesses**	*stiffness of the LL link*
RResistances	**Resistances**	*resistance of the LL link*
RNonLinearity	**NonLinearity**	*nonlinearity of the LL link*

)

RLinkElementRec = (

RNNLinkElementRec	**NNLinkElementRec;**
RLLLinkElementRec	**LLLinkElementRec;**

)

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] RLLLinkElementRec * **LLLinkElementRec**)

LLLinkElementRec *LL link element parameters*

Add line to line link. If successful returns LinkID, otherwise an error code (errDatabaseNotReady, leeLineIndexOutOfBounds, leeErrorAddingLL, leeNotConnectingMasterLineAndMasterStartLink, leeNotConnectingMasterLineAndMasterEndLink, leeNotConnectingSlaveLineAndMasterStartLink, leeNotConnectingSlaveLineAndMasterStartLink)

---

long **Clear**

Returns number of link elements before delete, otherwise an error code (errDatabaseNotReady).

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

Index *index of the link element, 1 ≤ Index ≤ Count*

If successful returns index, otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBounds)

long **DeleteSelected**

Returns number of deleted link elements, otherwise an error code (errDatabaseNotReady).

long **GetRec** ([in] long **Index,** [i/o] RLinkElementRec* **LinkElementRec**)

Index *link element index, 1 ≤ Index ≤ Count*

LinkElementRec *link element record*

If successful returns LinkID, otherwise an error code (errDatabaseNotReady, leeLineIndexOutOfBounds, leeInvalidLinkElementType)

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

ItemIds *Index list of selected link elements*

Returns the number of selected link elements

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

Select *selection state*

If Select is True, selects all link elements.
If Select is False, deselects all link elements.
If successful, returns the number of selected link elements, otherwise returns an error code

(errDatabaseNotReady)

NOTE: Call Refresh function afterwards if not called between functions BeginUpdate and EndUpdate

long **SetRec** ([in] long **Index,** [i/o] RLinkElementRec* **LinkElementRec**)

Index *link element index, 1 ≤ Index ≤ Count*

LinkElementRec *link element record*

If successful returns LinkID, otherwise an error code (errDatabaseNotReady, leeLineIndexOutOfBounds, leeInvalidLinkElementType, leeReferenceIndexOutOfBounds, leeInvalidLinkElementType, leeNotConnectingMasterLineAndMasterStartLink, leeNotConnectingMasterLineAndMasterEndLink, leeNotConnectingSlaveLineAndMasterStartLink, leeNotConnectingSlaveLineAndMasterStartLink)

Properties

long **Count**

Get number of link elements

ELinkElementType **LinkElementType** [long **Index**]

Index *index of the link element, 1 ≤ Index ≤ Count*

Get element type, otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBounds, leeInvalidLinkElementType)

ELongBoolean **Selected** **[**long **Index]**

Index *index of the link element, 1 ≤ Index ≤ 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 s*et 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,	*Lowest node of model*
bhtCustom = 0x1 }	*Use BaseHeight set by user*

How to determinate base height

enum **ELoadCaseType** = {

lctStandard = 0x0,	*standard loads*
lctInfluenceLine = 0x1,	*influence line loads*
lctSeismic = 0x2,	*seismic loads, output only*
lctVibration = 0x3,	*vibration loads*
lctPreStress = 0x4,	*prestress loads*
lctMoving = 0x5,	*moving loads*
lctDynamic = 0x6,	*dynamic loads*
lctPushOver = 0x7,	*push over loads, output only*
lctImperfection = 0x8 ,	*imperfection loads, output only*
lctSnow = 0x9 ,	*snow loads, output only*
lctSnowExcept = 0xA ,	*exceptional snow loads, output only*
lctWind = 0xB,	*wind loads, output only*
lctManualSeismic = 0xC ,	*Manual seismic loads*
lctManualPreStress = 0xD	*Manual prestress loads*
lctFire = 0xE ,	*fire load case/fire effects*
lctLocalImperfection = 0xF ,	*local imperfection load case*
lctCFD = 0x10	*computational fluid dynamics load case (practically wind load) (not yet implemented)*
lctSelfWeight = 0x11 }	*automatic self weight loadcase*

Load case type.

enum **ELoadDurationClass** = {

ldcOther = 0x0,	*Other*
ldcPermanent = 0x1,	*Permanent*
ldcLong = 0x2,	*Long*
ldcMedium = 0x3,	*Medium*
ldcShort = 0x4,	*Short*
ldcInstant = 0x5 }	*Instant*

Load duration class

enum **EModalCombType** = {

mctAuto = 0x0,	*automatic*
mctSRSS = 0x1,	*Square Root of Sum of Squares*
mctCQC = 0x2 }	*Complete Quadratic Combination*

Combination type for modal responses in one direction

enum **ESeismicCombType** = {

sctQuadratic = 0x0,	*quadratic mean*
sctMax = 0x1,	*combination with 30%*
sctAuto = 0x2 }	*automatic*

Combination type for spatial components.

enum **EVibrationType** = {

vtFirstOrder = 0x0,	*first order vibration*
vtSecondOrder = 0x1 }	*second order vibration*

Vibration result type.

enum **ESwayDirection** = {

sdPlusX = 0x0,	*Sway in positive x direction*
sdMinX = 0x1,	*Sway in negative x direction*
sdPlusY = 0x2,	*Sway in positive y direction*
sdMinY = 0x3,	*Sway in negative y direction*
sdCustom = 0x4,	*Sway in custom direction*

Sway direction

enum **ETerrainCategory** = {

tc0 = 0x0,	*Terrain category 0*
tcI = 0x1,	*Terrain category I*
tcII = 0x2,	*Terrain category II*
tcIII = 0x3,	*Terrain category III*
tcIV = 0x4	*Terrain category IV*

}

Terrain category

enum **ERoofType** = {

rtUndefined = 0x0,	*Undefined roof*
rtFlat = 0x1,	*Flat roof*
rtMonopitch = 0x2,	*Monopitch roof*
rtDuopitch = 0x3,	*Duopitch roof*
rtHip = 0x4,	*Hip roof*
rtBarrel = 0x5	*Barrel roof*

}

Roof type

enum **EFlatRoofEdgeType** = {

retNone = 0x0,	*Nothing on the edge*
retSharpEaves = 0x1,	*Edge with sharp eaves*
retWithParapetWall = 0x2,	*Edge with parapet wall*
retRoundEaves = 0x3,	*Edge with round eaves*
retMansardEaves = 0x4,	*Edge with mansard eaves*

}

Edge type of flat roof

enum **ELargeRoofModeSIA** = {

lrms_Value = 0x0,	*user specified value*
lrms_LoadPanel = 0x1,	*automatically determined based on the load panel*

}

Edge type of flat roof

enum **ESeismicLimitState** = *{*

selsOther = 0,	*other*
selsOperational = 1,	*operational limit state*
selsDamage = 2,	*damage limit state*
selsLifeSafety = 3,	*life-safety limit state*
selsCollapse = 4,	*collapse limit state*

}

seismic limit state for NationalDesignCode=ndcItalian

Error codes

enum **ELoadCasesError** = {

lcaePropertyNotValidForThisType = –100001	*property is not compatible with the load case type*
lcaeNameExists = –100002	*load case with the same name already exists*
lcaeErrorCreatingStandardSeismicCases = -100003	*Error creating standard seismic cases*
lcaeGroupIdOutOfBounds = -100004	*loadcase group index out of bounds*
lcaeErrorCreatingPushOverCases = -100005	*Error creating standard pushover cases*
lcaeInvalidAnalysisTypeDirX = -100006	*Invalis analysis type for loadcase in x direction*
lcaeLoadCaseIndexOutOfBoundsDirX = -100007	*index out of bounds for loadcase in x direction*
lcaeVibrationModeIndexOfOutBoundsDirX = -100008	*Vibration mode index out of bounds for loadcase in x direction*
lcaeInvalidAnalysisTypeDirY = -100009	*Invalis analysis type for loadcase in y direction*
lcaeLoadCaseIndexOutOfBoundsDirY = -100010	*index out of bounds for loadcase in y direction*
lcaeVibrationModeIndexOfOutBoundsDirY = -100011	*Vibration mode index out of bounds for loadcase in y direction*
lcaeErrorCreatingPreStressCases = -100012	*Prestress case can not be created*
lcaeNoPushOverCases = -100013	*Push over cases don't exist or push over not supported by national design code*
lcaeInvalidLoadCaseType = -100014	*Load case type is not valid, use dedicated Create.. functions*
lcaeNoModeShapesForLoadCaseInDirectionX = -100015	*Calculate mode shapes for the selected loadcase or loadcombination in x direction*
lcaeNoModeShapesForLoadCaseInDirectionY = -100016	*Calculate mode shapes for the selected loadcase or loadcombination in y direction*
lcaeNoModeShapesInDirectionX = -100017	*Calculate mode shapes for the selected loadcase or loadcombination in x direction*
lcaeNoModeShapesInDirectionY = -100018	*Calculate mode shapes for the selected loadcase or loadcombination in y direction*
lcaeInvalidLoadGroupType = -100019	*Load case type is not valid, use dedicated functions: e.g.: Create…*
lcaeSWGmoduleNotAvailable = -100020	*Extension module SWG is not available*
lcaeNoSnowLoadCases = -100021	*Create snow load cases before calling function returning this error*
lcaeNoWindLoadCases = -100022	*Create wind load cases before calling function returning this error*
lcaeInvalidName = -100023	*Name of the load case is invalid*
lcaeNoSeismicParams = -100024	*Seismic parameters are not available*
lcaeSE1moduleNotAvailable = -100025	*Extension module SE1 is not available*
lcaeErrorSetingSeismicParams = -100026	*Seismic parameters are invalid*
lcaeLoadCaseLoadCombinationNotFound = -100027	*Load case or load combination index is invalid*
lcaeSeismicInvalidGroupID = -100028	*No seismic group with the specified ID*

}

Records / structures

RImperfectionParams = (

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

RPushOverDirectionParams = (

ELongBoolean	**Uniform**	*If true, then uniform load distribution*
ELongBoolean	**Modal**	*If true, then modal load distribution*
EAnalysisType	**VibrationAnalysisType**	*Analysis type, only* ***atLinearVibration*** *or* ***atNonLinearVibration*** *are valid*
long	**VibrationLoadCase**	*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*
long	**VibrationMode**	*Vibration mode; 0=< and <=ModeShapes, 0 if* AutoDominantMode=lbTrue
ELongBoolean	**AutoDominantMode**	*If true, then dominant vibration mode is selected and* **VibrationMode** *value is ignored*
long	**AccidentalEcc**)	*Accidental eccentricity*

RPushOverParams = (

RPushOverDirectionParams	**X**	*Pushover parameters in x direction*
RPushOverDirectionParams	**Y**	*Pushover parameters in y direction*
	)	

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) ν*
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 it 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) ν*
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*

)

RSnowLoadParams = (

Warning! *This record was superseded by RSnowLoadParams_V171*

double	**a**	*altitude above sea level [m]*
double	**C_e**	*exposure coefficient [ ]*
double	**C_t**	*thermal coefficient [ ]*
double	**C_esl**	*exceptional snow load coefficient [ ]*
double	**s_k**	*characteristic value of snow load on the ground [kN / m^2]*
double	**s_Ad**	*design value of exceptional snow load on the ground (only for design codes where there is exceptional snow load) [kN / m^2]*
double	**I_w**	*importance factor, only for EC_RO [ ]*
long	**Zone**	*snow zone, only for a select few design codes. The interpretation is dependent on the design code*

)

RSnowLoadParams_V171 = (

double	**a**	*altitude above sea level [m]*
double	**C_e**	*exposure coefficient [ ]*
double	**C_t**	*thermal coefficient [ ]*
double	**C_esl**	*exceptional snow load coefficient [ ]*
double	**s_k**	*characteristic value of snow load on the ground [kN / m^2]*
double	**s_Ad**	*design value of exceptional snow load on the ground (only for design codes where there is exceptional snow load) [kN / m^2]*
double	**I_w**	*importance factor, only for EC_RO [ ]*
double	**P_n**	*annual probability of exceedence [ ]*
double	**V**	*coefficient of variation [ ]*
double	**mu1_0**	*base value of the load shape coefficient [ ]*
long	**Zone**	*snow zone, only for a select few design codes. The interpretation is dependent on the design code*
BTSR	**CountyName_NO**	*county name, only for EC_NO*
BTSR	**MunicipalityName_NO**	*municipality name, only for EC_NO*
double	**sk0**	*only for EC_NO, see NS-EN 1991-1-3:2003+A1:2015+NA:2018, NA.4.1 Karakteristiske verdier [kN / m^2]*
double	**Hg**	*only for EC_NO, see NS-EN 1991-1-3:2003+A1:2015+NA:2018, NA.4.1 Karakteristiske verdier [m]*
double	**Deltask**	*only for EC_NO, see NS-EN 1991-1-3:2003+A1:2015+NA:2018, NA.4.1 Karakteristiske verdier [kN / m^2]*
double	**sk_maks**	*only for EC_NO, see NS-EN 1991-1-3:2003+A1:2015+NA:2018, NA.4.1 Karakteristiske verdier [kN / m^2]*
ELongBoolean	**LargeRoofSIA**	*only for SIA, large, low pitch roofs*
ELargeRoofModeSIA	**LargeRoofModeSIA**	*only for SIA, only for LargeRoofSIA = lbTrue, size source*
double	**MaxSizeSIA**	*only for SIA, only for LargeRoofSIA = lbTrue and LargeRoofModeSIA = lrms_Value, the user specified maximum size [m]*

)

RWindLoadParameters = (

Warning! This record has become obsolete, IAxisVMWindLoad is used instead

double	**A**	*Altitude above sea level*
double	**v_b0**	*Basic wind velocity*
double	**c_season**	*Season factor*
double	**c_o**	*Ortography 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**	*directional factors in –y direction*
double	**c_dir_ym**	*directional factors in +Y direction*
ERoofType	**RoofType**	*Roof type*
EFlatRoofEdgeType	**FlatRoofEdgeType**	*Used only if RoofType = rtFlat*
double	**FlatRoofEdgeParam**	*parapet heigh(h_p), radius (r) or pitch(alfa) depends on FlatRoofEdgeType*
ELongBoolean	**TorsionalEffect**	*If lbTrue then additional load cases will be created for torsional winds*
double	**u_xp**	*μ factor related to the area of openings in +X direction*
double	**u_xm**	*μ factor related to the area of openings in -X direction*
double	**u_yp**	*μ factor related to the area of openings in +Y direction*
double	**u_ym**	*μ factor related to the area of openings in -Y direction*
double	**Iw**	*Importance factor (ndcEuroCode_RO only)*
long	**Zone**	*Number of supported zones depened on national design code where first zone name is 1, second 2, etc..* *Used only in listed design codes:* ***ndcEuroCode_PL:** Strefa 1, Strefa 2, Strefa 3,* ***ndcEuroCode_GER:** Zone 1, Zone 2, Zone 3, Zone 4* ***ndcItalian:** Zona 1, Zona 2, Zona 3, Zona 4, Zona 5, Zona 6, Zona 7, Zona 8, Zona 9* ***ndcEuroCode_RO:** Zona 1, Zona 2, Zona 3, Zona 4, Zona 5* ***ndcEuroCode_CZ:** Zone 1, Zone 2, Zone 3, Zone 4, Zone 5* ***ndcEuroCode_NL**: Zone I, Zone II, Zone III* ***ndcEuroCode_B:** Zone 1, Zone 2, Zone 3, Zone 4*

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 goup 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**)

Name *This string is used for generating the pushover load case names*

PushOverParams *Pushover parameters*

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).

Important note!

To recalculate call the IAxisVMLoads.CreateStandardPushOverLoads function after loading a spectrum.

---

long **CreatePreStressCases** ([in] BSTR **Name,** [in] long **LoadGroup**)

Name *This string is used for generating the prestress load case names*

LoadGroup *index of the load group*

Creates prestress load cases. If successful, returns total number of load cases, otherwise returns an error code (errDatabaseNotReady, lcaeErrorCreatingPreStressCases, lcaeGroupIdOutOfBounds).

---

long **CreateSnowCases** ([in] BSTR **Name,** [i/o] RSnowLoadParams **SnowLoadParams**)

Warning! *This function has become obsolete, was superseded by CreateSnowCases_V171*

Name *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.*

SnowLoadParams *Snow load parameters*

Creates snow load cases and snow load goup 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).

Important note!

To calculate the snow loads also call the IAxisVMLoads.CreateSnowLoadOnLoadPanels.

---

long **CreateSnowCases_V171** ([in] BSTR **Name,** [i/o] RSnowLoadParams_V171 **SnowLoadParams**)

Name *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.*

SnowLoadParams *Snow load parameters*

Creates snow load cases and snow load goup 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).

Important note!

To calculate the snow loads also call the IAxisVMLoads.CreateSnowLoadOnLoadPanels.

---

long **CreateStandardSeismicCases** ([in] BSTR **lcname**)

lcname *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+*

Creates standard seismic load cases and seismic load goup 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).

Important note!

To calculate the seismic loads set the SeismicParameters with SetSeismicParams_V153 function, and optionaly 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.

---

long **CreateWindCases** ([in] BSTR **Name,** [i/o] RWindLoadParams **WindLoadParams**)

Warning! *This function has become obsolete, see IAxisVMWindLoad for handling wind loads*

Name *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.*

WindLoadParams *Wind load parameters*

Creates wind load cases and wind load goup 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).

Important note!

To calculate the snow loads also call the IAxisVMLoads.CreateWindLoadOnLoadPanels function.

---

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

Warning! *AxisVM assumes some rules are followed when deleting load cases. Use Delete_V153 if you want a safer way to delete load cases.*

Index *index of the load case to delete (0 < Index ≤ Count)*

Deletes a load case by index.
If successful, returns Index, otherwise returns an error code (errIndexOutOfBounds, errDatabaseNotReady).

---

long **Delete_V153** ([in] long **Index,** [in] ELongBoolean **GroupedDelete,** [in] ELongBoolean **DeleteEmptiedGroup**)

Index *index of the load case to delete (0 < Index ≤ Count)*

GroupedDelete *if the load case is part of an internally grouped entity (i.e. automatically more than 1 load case may be created, like for seismic loads, pushover loads, moving loads, tensioning, wind loads, snow loads), deleting a load case will delete the internally grouped load cases too, if set to True*

DeleteEmptiedGroup *if after a load case deletion the affected Load group becomes empty, it can be deleted too. If set to True, this emptied load group deletion happens automatically. Warning : when deleting a snow or wind load case, the load group will always be deleted, regardless of the value of DeleteEmptiedGroup*

Deletes a load case by index. If GroupedDelete is true, all the internally grouped load cases will be deleted if one of them is deleted. This internal grouping is a different concept from the Load groups. For example a moving load group may contain 3 different internal moving load case groups. Deleting a load form one of them even with GroupedDelete set to True, will remove only those internally grouped to the deleted load case. After this deletion, the moving load group will still contain the load cases for the 2 other internally grouped moving loads.
If successful, returns Index, otherwise returns an error code (errIndexOutOfBounds, errDatabaseNotReady).

---

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

Index *index of the load case to delete (0 < Index ≤ Count)*

Deletes all loads from the load case by index.
If successful, returns Index, otherwise returns an error code (errIndexOutOfBounds, errDatabaseNotReady).

---

long **DeleteSnowLoadCases** ()

Deletes all snow load cases and the snow load group. A call to CreateSnowCases_V171 can be used to recreate them later, according to new parameters. If only the load cases have to be recreated, CreateSnowCases_V171 already handles the deletion of the previous load cases, no need for this call. If successful, returns 1, otherwise returns an error code (errDatabaseNotReady).

---

long **GetImperfectionParams** ([in] long **Index,** [i/o] RImperfectionParams **ImperfectionParams**)

Index *index of the load case*

ImperfectionParams *Imperfection parameters*

Creates imperfection load case. Returns index if successful l or error code (errIndexOutOfBounds, errDatabaseNotReady, lcaeInvalidLoadCaseType).

---

long **GetPushOverParams** ([i/o] RPushOverParams **PushOverParams**)

PushOverParams *Pushover parameters*

Get pushover parameters. Returns 1 if successful l or error code (errDatabaseNotReady, lcaeNoPushOverCases, lcaeNoModeShapesInDirectionX, lcaeNoModeShapesInDirectionY).

---

long **GetSeismicParams** ([i/o] RSeismicParams **SeismicParams**)

Warning! *This function has become obsolete, was superseded by GetSeismicParams_V153*

SeismicParams *Seismic parameters according to the code*

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, lcaeNoSeismicLoadCases, lcaeSE1moduleNotAvailable).

---

long **GetSeismicParams_V153** ([in] long **GroupID,** [i/o] RSeismicParams_V153 **SeismicParams**)

GroupID *Seismic group ID*

SeismicParams *Seismic parameters according to the code*

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, lcaeNoSeismicLoadCases, lcaeSE1moduleNotAvailable, lcaeSeismicInvalidGroupID).

---

long **GetSnowLoadParams** ([i/o] RSnowLoadParams **SnowLoadParams**)

Warning! *This function has become obsolete, was superseded by GetSnowLoadParams_V171*

SnowLoadParams *Snow load parameters*

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, lcaeNoSnowLoadCases or lcaeSWGmoduleNotAvailable)

---

long **GetSnowLoadParams_V171** ([i/o] RSnowLoadParams_V171 **SnowLoadParams**)

SnowLoadParams *Snow load parameters*

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, lcaeNoSnowLoadCases or lcaeSWGmoduleNotAvailable)

---

long **GetWindLoadParams** ([i/o] RWindLoadParams **WindLoadParams**)

Warning! *This function has become obsolete, see IAxisVMWindLoad for handling wind loads*

WindLoadParams *Wind load parameters*

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, lcaeNoWindLoadCases or lcaeSWGmoduleNotAvailable)

---

long **SeismicGroupIDs** ([out] SAFEARRAY(long)* **GroupIds**)

GroupIds *list of all available seismic groups. Null if there is none.*

Queries the available seismic group indexes. GroupIds will be null if there is none. Return value is a positive number for successl or error code (errDatabaseNotReady).

---

long **SeismicSpectrumH** ([in] long **GroupID,** [out] IAxisVMSpectrum **Spectrum**)

GroupID *group ID of a seismic group*

Spectrum *the returned horizontal spectrum*

Gets the horizontal spectrum for the given GroupID. If successful it returns a positive number, otherwise it returns error (errDatabaseNotReady, lcaeNoSeismicLoadCases, lcaeSE1moduleNotAvailable, lcaeSeismicInvalidGroupID).

---

long **SeismicSpectrumV** ([in] long **GroupID,** [out] IAxisVMSpectrum **Spectrum**)

GroupID *group ID of a seismic group*

Spectrum *the returned vertical spectrum*

Gets the vertical spectrum for the given GroupID. You can check its Disabled property to see wether it is active or not. If successful it returns a positive number, otherwise it returns error (errDatabaseNotReady, lcaeNoSeismicLoadCases, lcaeSE1moduleNotAvailable, lcaeSeismicInvalidGroupID).

---

long **SetImperfectionParams** ([in] long **Index,** [i/o] RImperfectionParams **ImperfectionParams**)

Index *index of the load case*

ImperfectionParams *Imperfection parameters*

Creates imperfection load case. Returns index if successful l or error code (errIndexOutOfBounds, errDatabaseNotReady, lcaeInvalidLoadCaseType).

---

long **SetSeismicParams** ([i/o] RSeismicParams **SeismicParams**)

Warning! *This function has become obsolete, was superseded by SetSeismicParams_V153*

SeismicParams *Seismic parameters according to the code*

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, lcaeNoSeismicLoadCases, lcaeSE1moduleNotAvailable, lcaeErrorSetingSeismicParams, lcaeLoadCaseLoadCombinationNotFound).

---

long **SetSeismicParams_V153** ([i/o] RSeismicParams_V153 **SeismicParams**)

SeismicParams *Seismic parameters according to the code. GroupID field must be set to the intended seismic group ID.*

Set the seismic parameters according to the design code. If successful it returns the index of the first seismic load case from the seismic goup, otherwise it returns error (errIndexOutOfBounds, lcaeNoSeismicLoadCases, lcaeSE1moduleNotAvailable, lcaeErrorSetingSeismicParams, lcaeLoadCaseLoadCombinationNotFound, lcaeSeismicInvalidGroupID).

---

long **SetSnowLoadParams** ([i/o] RSnowLoadParams **SnowLoadParams**)

Warning! *This function has become obsolete, was superseded by SetSnowLoadParams_V171*

SnowLoadParams *Snow load parameters*

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, lcaeNoSnowLoadCases or lcaeSWGmoduleNotAvailable)

---

long **SetSnowLoadParams_V171** ([i/o] RSnowLoadParams_V171 **SnowLoadParams**)

SnowLoadParams *Snow load parameters*

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, lcaeNoSnowLoadCases or lcaeSWGmoduleNotAvailable)

---

long **SetWindLoadParams** ([i/o] RWindLoadParams**WindLoadParams**)

Warning! *This function has become obsolete, see IAxisVMWindLoad for handling wind loads*

WindLoadParams *Wind load parameters*

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, lcaeNoWindLoadCases or lcaeSWGmoduleNotAvailable)

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 *index of load case*

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 *index of load case*

ELoadDurationClass **LoadDurationClass** [long **Index**] • *Get or set load duration class. Negative number is an error code (errIndexOutOfBounds, errDatabaseNotReady).*

Index *index of load case*

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.

Index *index of load case*

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*

Index *index of load case*

long **UID** [long **Index**] *Get unique index of the load case which mains the same while exists in the model*

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*
ctULSAllSE1 = 20 ,	*ULSAll, with the seismic combination from seismic group 1 (the lowest groupID)*
ctULSAllSE2 = 21 ,	*ULSAll, with the seismic combination from seismic group 2*
ctULSAllSE3 = 22 ,	*ULSAll, with the seismic combination from seismic group 3*
ctULSAllSE4 = 23 ,	*ULSAll, with the seismic combination from seismic group 4*
ctULSAllSE5 = 24 ,	*ULSAll, with the seismic combination from seismic group 5*
ctULSAllSE6 = 25 ,	*ULSAll, with the seismic combination from seismic group 6*
ctULSAllSE7 = 26 ,	*ULSAll, with the seismic combination from seismic group 7*
ctULSAllSE8 = 27 ,	*ULSAll, with the seismic combination from seismic group 8 (the highest groupID)*
ctSemiAutoSLS1 = 28 ,	*when SLS combinations are taken into account, the default value for them is overridden by the first SLS type (for Eurocode it is the characteristic SLS). Only valid when semi-auto critical type is selected. It is a virtual combination type, actual load combinations mustn't have his type.*

ctSemiAutoSLS2 = 29 , *when SLS combinations are taken into account, the default value for them is overridden by the second SLS type (for Eurocode it is the frequent SLS). Only valid when semi-auto critical type is selected. It is a virtual combination type, actual load combinations mustn't have his type.*

ctSemiAutoSLS3 = 30 , *when SLS combinations are taken into account, the default value for them is overridden by the third SLS type (for Eurocode it is the quasi-permanent SLS). Only valid when semi-auto critical type is selected. It is a virtual combination type, actual load combinations mustn't have his type.*

ctAuto = 31, *automatic combination type based on the result component (like My for a beam), or the internal solutions used in more complex design calculations (like beam design)*

}

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,	*Accidental ULS combination (See 6.4.3.3 EN 1990)*
ctb_ULSExceptional = 0x00000020,	*same as* **ctb_ULS3**
ctb_ULSALL = 0x00000040,	*see* **ctULSALL**
ctb_ULSab = 0x00000080,	*see* **ctULSab**
ctb_USLa = 0x00000100,	*not used*
ctb_USLb = 0x00000200,	*not used*
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*
ctb_ULSAllSE1 = 0x00080000,	*see ctULSAllSE1*
ctb_ULSAllSE2 = 0x00100000,	*see ctULSAllSE2*
ctb_ULSAllSE3 = 0x00200000,	*see ctULSAllSE3*
ctb_ULSAllSE4 = 0x00400000,	*see ctULSAllSE4*
ctb_ULSAllSE5 = 0x00800000,	*see ctULSAllSE5*
ctb_ULSAllSE6 = 0x01000000,	*see ctULSAllSE6*
ctb_ULSAllSE7 = 0x02000000,	*see ctULSAllSE7*
ctb_ULSAllSE8 = 0x04000000	*see ctULSAllSE8*

}

Load combination types used for generating different types of combinations

Error codes

enum **ELoadCombinationsError** = {

lcoeDifferentFactorsAndIDsCount = –100001	*the number of load cases and factors are different*
lcoeNameExists = –100002	*existing load combination name*
lcoeAutoGenerationFailed = -100003	*Auto generation failed*
lcoeAutoGenerationFailedNoCriticalGroup = - 100004	*Load groups are not defined*
lcoeInvalidCombinationTypesValue = -100005 }	*sum of combination types is zero or invalid*

Records / structures

RLoadCombinationGenParameters = (

ELongBoolean	**ConsiderImperfections;**	*consider imperfection load cases*
ELongBoolean	**OverwriteGeneratedCombos**	*if true,delete previously generated load combinations*
ELongBoolean	**OverWriteDuplComboSameType**	*used only if* **OverwriteGeneratedCombos** *is True. Delete previously generated load combinations with same* ECombinationType

)

Functions

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

Name	*name of the load combination*
CombinationType	*combination type*
Factors	*combination factors*
LoadCaseIds	*combination load case indexes according to IAxisVMLoadCases*

Adds a new load combination to the model.
If successful, returns the index of the new combination, otherwise returns an error code (lcoeDifferentFactorsAndIDsCount, lcoeNameExists, errDatabaseNotReady).

---

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

---

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

Index *index of the load combination to delete (0 < Index ≤ Count)*

Deletes a load combination.
If successful, returns Index, otherwise returns an error code (errIndexOutOfBounds, errDatabaseNotReady).

---

long **GenerateAutoCombinations** ([in] long **CombinationTypes,** [in] ELongBoolean **ShowForms,**
[i/o] RLoadCombinationGenParameters **LoadCombinationGenParameters**)

CombinationTypes *sum of ECombinationTypeBits values, except ctb_Other (e.g. ctb_SLS1+ctb_ULS1=0x09 therefore characteristic SLS and fundamental ULS combination will be generated)*
ShowForms *Show window (with error messages) while generating combinations*
LoadCombinationGenParameters *Paremeters for generating load combinations*

Generates specified load combinations.
If successful, returns number of generated combinations, otherwise returns an error code (lcoeAutoGenerationFailed, lcoeAutoGenerationFailedNoCriticalGroup, lcoeInvalidCombinationTypesValue, errDatabaseNotReady).

---

long **GetCombination** ([in] long **Index,** [out] SAFEARRAY(double)* **Factors,**
[out] SAFEARRAY(long)* **LoadCaseIds**)

Index *index of the load combinazion (0 < Index ≤ Count)*
Factors *load combination factors*
LoadCaseIds *combination load case indexes according to IAxisVMLoadCases*

Retrieves a load combination.
If successful, returns Index, otherwise returns an error code (errIndexOutOfBounds, errDatabaseNotReady).

---

long **GetValidCombinationTypes** ([out] SAFEARRAY(ECombinationType)* **CombinationTypes**)

CombinationTypes *array of valid combination types*

Retrieves critical load combinations which can be used for reading results. If successful, returns 1, otherwise returns an error code (errDatabaseNotReady).

---

long **SetCombination** ([in] long **Index,** [in] SAFEARRAY(double)* **Factors,**
[in] SAFEARRAY(long)* **LoadCaseIds**)

Index *index of the load combinazion (0 < Index ≤ Count)*
Factors *load combination factors*
LoadCaseIds *combination load case indexes according to IAxisVMLoadCases*

Modifies an existing load combination.
If successful, returns Index, otherwise returns an error code (leDifferentFactorsandIDsCount, errIndexOutOfBounds, errDatabaseNotReady).

---

long **SetCombination_vb** (Visual Basic compatible function of **SetCombination**)

Properties

ECombinationType **CombinationType** [long **Index**] *Get type of a combination*

long **Count** *Get number of load combinations in the model*
Negative number is an error code (errDatabaseNotReady).

BSTR **Comment** [long **Index**] • *Get or set comments for combination.*

Index *index of load combination*

long **IndexOfUID** [long **UID**] *Get index of the load combination*

UID *unique index of the load combination*

BSTR **Name** [long **Index**] • *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.

long **UID** [long **Index**] *Get unique index of the load combination which remains the same while exists in the model*

Index *index of load combination*

IAxisVMLoadGroups

Load groups in the model.

Enumerated types

enum **EGroupCombinationType** = {

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

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

enum **ELoadGroupType** = {

lgtPermanent = 0x0,	*permanent load group*
lgtIncidental = 0x1,	*incidental load group*
lgtExceptional = 0x2,	*exceptional load group*
lgtSeismic = 0x3,	*seismic load group, output only*
lgtPrestress = 0x4,	*prestress load group*
lgtMoving = 0x5,	*moving load group*
lgtImperfection = 0x6,	*imperfection load group, output only*
lgtSnow = 0x7 ,	*snow load group, output only*
lgtSnowExcept = 0x8 ,	*exceptional snow load group, output only*
lgtWind = 0x9 ,	*wind load group, output only*
lgtManualSeismic = 0xA ,	*manual seismic load group*
lgtManualPreStress = 0xB,	*manual prestress load group*
lgtFire = 0xC,	*fire load group*
lgtCFD = 0xD }	*CFD load group*

Load group types.

Error codes

enum **ELoadGroupsError** = {

lgePropertyNotValidForThisType = –100001,	*property is not compatible with the load group type*
lgeNameExists = –100002 ,	*group with the same name already exists*
lgeInvalidType = –100003 ,	*use dedicated Create.. functions in IAxisVMLoadCases interface*

}

Functions

long **Add** ([in] BSTR **Name,** [in] ELoadGroupType **LoadGroupType,**
[in] ELongBoolean **SimultExc,** [in] EGroupCombinationType **CombinationType**)

Name	*name of the load group*
LoadGroupType	*load group type*
SimultExc	*(if LoadGroupType = lgtIncidental) set if load cases of the group can be simultaneous with load cases of exceptional groups*
CombinationType	*Behaviour of load cases of groups when determining the critical combination.*

Adds a new load group to the model.
If successful, returns the index of the new group, otherwise returns an error code (errDatabaseNotReady, lgeNameExists, lgeInvalidType).

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

Index *index of the load group to delete (0 < Index ≤ Count)*

Deletes a load group.
If successful, returns Index, otherwise returns 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. Incompatibilty creates an error event with the error code *lgePropertyNotValidForThisType*. Properties with *(EC)* are valid in *EC, SIA, DIN, STAS, I* 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	**GammaInf** • *(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	**GammaI** • *Get or set seismic safety factor (γ_I)*
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 I*
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** = {

lopeInvalidContourParams = –100001,	*Size of the array does not match total number of edges*
lopeLineIndexlListEmpty = –100002 ,	*Array of line indexes is empty*
lopeLineIndexOutOfBounds = –100003 ,	*Line index is out of bounds*
lopeInvalidContour = –100004 ,	*Contour is invalid, e.g. not closed, has only 2 points, etc.*
lopeSameLoadPanelExists = –100005 ,	*same load panel already exists*
lopeMemberIndexlListEmpty = –100006 ,	*Array of member indexes is empty*
lopeMemberIndexOutOfBounds = –100007 ,	*Member index is out of bounds*
lopeInvalidContourType = –100008,	*Type of the contour is invalid*
lopeNodeIndexlListEmpty = –100009 ,	*Array of node indexes is empty*
lopeDomainIndexlListEmpty = –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 errIndexOutOfBounds).

---

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, lopeLineIndexlListEmpty or lopeInvalidContour).

---

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, lopeMemberIndexlListEmpty or lopeInvalidContour).

---

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 lopeInvalidContour).

---

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 errIndexOutOfBounds).

---

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 = 0x0,	*User defined (generic) type of the load panel contour, not asociated with lines, members, domains.*
lpctAssociated = 0x1 }	*The load panel contour is asociated with lines, members, domains.*

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*
lpetOverhangingSnow = 0x3,	*for snow load panels, roof edge with overhanging snow*
lpetWallWithReturnCorner = 0x4 }	*for wind load panels, in case of free standing walls edge with return corner*

Type of load panel edge

Records / structures

RLoadPanelEdgeParams = (

ELoadPanelEdgeType	**LoadPanelEdgeType**	*Type of load panel edge*
Double	**h**	*only for lpetParapet, lpetWall, lpetWallWithReturnCorner : height parameter [m]*
Double	**Alpha**	*Angle of the roof above the abutting wall (only for lpetWall) [°]*
Double	**b_1**	*Width of the taller construction (only for lpetWall) [m]*

)

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 **GetContourLineIds** ([out] SAFEARRAY(long)*** LineIds)**

LineIds 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)*** LineIds)**

LineIds Index array (long) with line indexes where load from panel is applied

Returns LineIds array length, otherwise returns an error code (errDatabaseNotReady).

long **GetNodes** ([out] SAFEARRAY(long)* **NodeIds)**

NodeIds Index array (long) with node indexes where load from panel is applied

Returns NodeIds array length, otherwise returns an error code (errDatabaseNotReady).

---

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 lopeInvalidContourParams)

---

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, errIndexOutOfBounds)

---

long **SetNodes** ([i/o] SAFEARRAY(long)* **NodeIds)**

NodeIds Index array (long) with node indexes where load from panel is applied

Returns NodeIds array length, otherwise returns an error code (errDatabaseNotReady or ELoadPanelsError).

---

long **SetLines** ([i/o] SAFEARRAY(long)* **LineIds)**

LineIds Index array (long) with line indexes where load from panel is applied

Returns LineIds 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

ELongBoolean **Auto** • *Get or set whetheher the load panel will apply loads automatically.*

lbTrue: load will be applied automatically to all nodes and lines within the contour

lbFalse: load will be applied only to set nodes and lines(See SetNodes and SetLines functions)

ELoadPanelContourType **ContourType** *Get type of the load panel*

long **EdgeCount** *Get number of edges of the load panel*

ELongBoolean **SelectedEdge** [long **Index**] • *Get or set the selection status of a load panel's edge*

Index *index of the load panel's edge*

NOTE: Call Refresh function afterwards if not called between functions BeginUpdate and EndUpdate

IAxisVMLoads

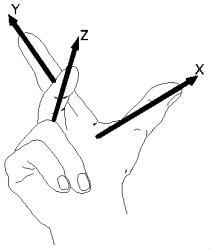
Loads of the model

NOTE:

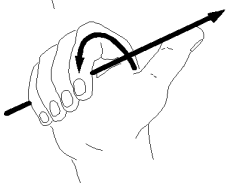
If load is applied to rib, then the point of application of defined loads are transfered, explained here.

Enumerated types

enum **EAxis** = {



Ax = 0x00,	*x direction*
Ay = 0x01,	*y direction*
Az = 0x02,	*z direction*



aXX = 0x03,	*about x axis*
aYY = 0x04,	*about y axis*
aZZ = 0x05 }	*about z axis*

Directions. For specific cases only a subset of these values might be valid, as specified in the description.

For IAxisVMCalculation only the displacement components are used. In case of displacement controlled nonlinear analysis it specifies the displacement component (X, Y, Z)

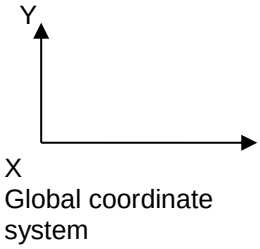
enum **EBeamRibDistributionType** = {

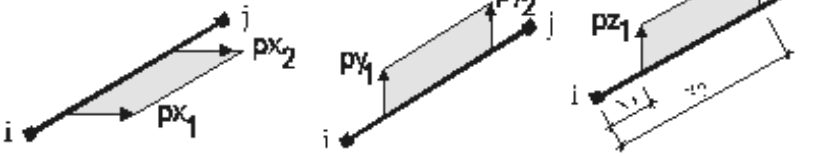
brdtLength = 0x00,	*by length*
brdtProjected = 0x01 }	*projected*

Beam/rib load distribution type.

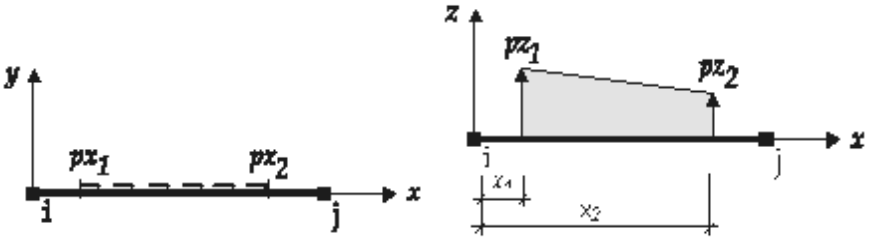
enum **EDistributionType** = {

dtGlobal = 0x00, *Global*





dtLocal = 0x01, *Local*



dtProjected = 0x02 *projected*

dtEdgeLocal = 0x03 } *dtEdgeLocal*

Mesh-independent surface load distribution type.

enum **ELoadDistributionType** = {

ldtConst = 0x00, *constant load intensity*
ldtLinear = 0x01 } *linear load intensity*

Mesh-independent surface load behaviour.

enum **ELoadType** = {

ltNodalForce = 0x00, *nodal force, see record RLoadNodalForce*
ltBeamConcentrated = 0x01, *concentrated load on beams, see record RLoadBeamConcentrated*
ltBeamDistributed = 0x02, *distributed load on beams, see record RLoadBeamDistributed*
ltBeamThermal = 0x03, *thermal load on beams, see record RLoadBeamThermal*
ltBeamStress = 0x04, *tension/compression load on beams, see record RLoadBeamStress*
ltBeamFault = 0x05, *„fault in length" load on beams, see record RLoadBeamFault*
ltBeamSelfWeight = 0x07, *self weight of beam*
ltTrussThermal = 0x08, *thermal load on trusses, see record RLoadTrussThermal*
ltTrussStress = 0x09, *tension/compression load on trusses, see record RLoadTrussStress*
ltTrussFault = 0x0A, *„fault in length" load on trusses, see record RLoadTrussFault*
ltTrussSelfWeight = 0x0B, *self weight of truss*
ltSurfaceSelfWeight = 0x0C, *self weight of surface element*
ltSurfaceDistributed = 0x0D, *distributed load on surface elements, see record RLoadSurfaceDistributed*
ltSurfaceEdge = 0x0E, *edge load on surface elements, see record RLoadSurfaceEdge*
ltSurfaceThermal = 0x0F, *thermal load on surface elements, see record RLoadSurfaceThermal*
ltSurfaceStress = 0x10, ***(not used)***
ltBeamInfluence = 0x11, *influence line load on beams, see record RLoadBeamInfluence*
ltDomainSelfWeight = 0x12, *self weight of domain*
ltDomainConstant = 0x13, *constant area load on domain, see record RLoadDomainConstant*
ltDomainEdge = 0x14, ***(not used)***
ltDomainThermal = 0x15, *thermal load on domains, see record RLoadDomainThermal*
ltDomainStress = 0x16, ***(not used)***
ltRibThermal = 0x17, *thermal load on ribs, see record RLoadRibThermal*
ltRibSelfWeight = 0x18, *self weight of rib*
ltRibConcentrated = 0x19, *concentrated load on ribs, see record RLoadRibConcentrated*
ltRibDistributed = 0x1A, *distributed load on ribs, see record RLoadRibDistributed*
ltSupportDisplacement = 0x1B, *support displacement, see record RLoadSupportDisplacement*
ltDomainConcentrated = 0x1C, *concentrated load on domains, see record RLoadDomainConcentrated*
ltSurfaceConcentrated = 0x1E, *concentrated load on surface elements, see record RLoadSurfaceConcentrated*
ltDomainPolyArea = 0x21, *distributed polygon area load on domains, see record RLoadDomainPolyArea*
ltDomainLinear = 0x22, *distributed area load on domains, see record RLoadDomainLinear*
ltDomainFluid = 0x24, *fluid load on domains, see record RLoadDomainFluid*
ltSurfaceFluid = 0x25, *fluid load on surface elements, see record RLoadSurfaceFluid*

ltLoadDomainPolyLine = 0x26,	*polygon load on domain, see record RLoadDomainPolyLine, to read all items use GetDomainPolyLineItems*
ltSurfaceToBeam = 0x27,	*surface load distributed over beams, see record RLoadSurfaceToBeam*
ltDomainPolyAssoc = 0x28,	*associative edge load on domains, see record RLoadDomainPolyAssoc, Access to associated lines through functions GetLines / SetLines*
ltSurfaceToBeamAssoc = 0x29,	*associative surface load distributed over beams, see record RLoadSurfaceToBeamAssoc*
ltDynamicNodalForce = 0x2A	*dynamic nodal force, see record RLoadDynamic*
ltDynamicNodalAcceleration = 0x2B,	*dynamic nodal acceleration, see record RLoadDynamic*
ltDynamicNodalSupportAcceleration = 0x2C,	*dynamic nodal support acceleration, see record RLoadDynamic*
ltBeamMemberConcentrated = 0x2D,	*concentrated force on beam structural member, see record RLoadBeamMemberConcentrated (obsolete)*
ltBeamMemberDistributed = 0x2E,	*distributed force on beam structural member, see record RLoadBeamMemberDistributed (obsolete)*
ltRibMemberConcentrated = 0x2F,	*concentrated force on rib structural member, see record RLoadRibMemberConcentrated (obsolete)*
ltRibMemberDistributed = 0x30,	*distributed force on rib structural member, see record RLoadRibMemberDistributed (obsolete)*
ltMemberConcentrated = 0x3A,	*concentrated force on beam structural member, see record RLoadMemberConcentrated*
ltMemberDistributed = 0x3B	*distributed force on beam structural member, see record RLoadMemberDistributed*
ltNone = 0xFFFFFFFF }	*none of the above*

Load types.

enum **ESurfaceDomainDistributionType** = {

sddtSurface = 0x00,	*surface*
sddtProjected = 0x01 }	*projected*

Surface/domain load distributon type.

enum **ESystem** = {

sysGlobal = 0x00,	*global system*
sysLocal = 0x01,	*local system*
sysReference = 0x02 }	*by reference*

Coordinate system of load components.

enum **EMemberLoadEcc** = {

mle_None = 0,	*no eccentricity*
mle_User = 1,	*user specified eccentricity*
mle_LeftTop = 2,	*load positioned on the left top of the cross-section*
mle_CenterTop = 3,	*load positioned on the center top of the cross-section*
mle_RightTop = 4,	*load positioned on the right top of the cross-section*
mle_LeftCenter = 5,	*load positioned on the left center of the cross-section*
mle_RightCenter = 6,	*load positioned on the right center of the cross-section*
mle_LeftBottom = 7,	*load positioned on the left bottom of the cross-section*
mle_CenterBottom = 8,	*load positioned on the center top of the cross-section*
mle_RightBottom = 9}	*load positioned on the right bottom of the cross-section*

Eccentricity type of member load

enum **ELoadDomainPolyLineItemType** = {

ldpitDomain = 0x00,	*on domain*
ldpitSurface = 0x01,	*on surface*
ldpitBlankLine = 0x02,	*on blank (virtual) line*
ldplitLoadPanel = 0x03 }	*on load panel*

Type of the loaded element

Error codes

enum **ELoadsError** = {

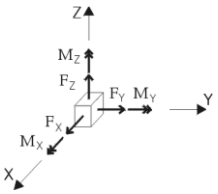
leInvalidLineType = –100001	*load type is not compatible with the line type*
leErrorAddingLoad = –100002	*error when adding load. It has a similar meaning to loeInvalidLoad, see the description there.*
leErrorSettingLoad = –100003	*error when setting load*
leInvalidLoadType = –100004	*invalid operation for this load type*
leNotValidLineTypeForThisLoad = –100005	*invalid line type when distributing surface load on lines*
leErrorSettingLines = –100006	*cannot set line list when distributing surface load on lines*
leErrorSettingPoly = –100007	*cannot set the load polygon*
leLoadCaseIndexOutOfBounds = –100008	*load record contains an invalid load case index*
leErrorSettingLoadCaseId = –100009	*load case index cannot be set*
leDomainIndexOutOfBounds = –100010	*load record contains an invalid domain index*
leMemberIndexOutOfBounds = –100011	*load record contains an invalid member index*
leThereAreNoSeismicStoreys= -100012	*CreateStandardSeismicLoads can return this error when the user requires torsion forces but the seismic storeys are not set*
loeNotDynamicLoadCase = -100013	*if the load case is not dynamic*
loeReferenceIndexOutOfBounds = -100014	*Reference index is out of boundaries*
loeErrorCreatingPushOverLoads = -100015	*Error while creating pushover loads*
loeInvalidLoadCaseType = -100016	*Load case type not valid*
loeInvalidRoofType = -100017	*Roof type not supported for this load*
loeLoadPanelIndexlListEmpty = -100018	*Array with load panel indexes is empty*
loeNoPushOverLoadCase = -100019	*Create pushover load cases before calling the function returning this error*
loeSE1moduleNotAvailable = -100020	*extension module SE1 is not available*
loeSE2moduleNotAvailable = -100021	*extension module SE2 is not available*
loeDYNmoduleNotAvailable = -100022	*extension module DYN is not available*
loeSWGmoduleNotAvailable = -100023	*extension module SWG is not available*
loeNoSnowLoadCase = -100024	*Create snow load cases before calling the function returning this error*
loeNoWindLoadCase = -100025	*Create wind load cases before calling the function returning this error*
loeNoSeismicLoadCase = -100026,	*Create seismic load cases before calling the function returning this error*
loePointIsOutOfLoadPanel = -100027	*the point is out of the load panel*
loeZeroLoadValueOnLoadPanel = -100028	*load with zero value has been defined*
loeDerivedSurfaceLoadsNotConverted = -100029	*derived surface loads have not been converted*
loeLoadComponentMustBeZero = -100030	*a load intensity supposed to be zero has a nonzero value (for example for nodal forces with a reference, only components Fx and Mx are valid. If Fz has a nonzero value, it is assumed to be a user error, and this error code is returned)*
loeInvalidLoad = -100031	*the load contains an invalid combination of fields (for example : all load intensities 0, distributed line loads with 0 length, thermal loads with Tref=T0, surface distributed loads with intensities in the plane of a plate, and so on). Generally speaking trying to create such a load manually in AxisVM through the user interface might provide some insight as to why is the load invalid. Some functions return leErrorAddingLoad instead, which has similar meaning.*
loeNoMassLoadOnNode = -100032	*the required action can be performed only on a node that has a mass load*
loeInvalidAxis = -100033	*invalid axis*
}	

Records / structures

Nodal loads

RLoadNodalForce = (

long	**LoadCaseId**	*load case index (0 < LoadCaseId ≤ AxisVMLoadCases.Count)*
long	**NodeId**	*node index (0 < Index ≤ 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	**ReferenceId**	*If set to 0, load components are in global directions.*
	)	*If 0 < ReferenceId ≤ AxisVMReferences.Count, the load direction is set by the reference. ReferenceId is the index of reference according to AxisVMReferences.*



Truss loads

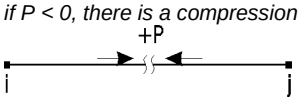
RLoadTrussFault = (

long	**LoadCaseId**	*load case index (0 < LoadCaseId ≤ AxisVMLoadCases.Count)*
long	**LineId**	*line index (0 < Index ≤ AxisVMLines.Count)*
double	**DL**	*fault in length*

)

RLoadTrussStress = (

long	**LoadCaseId**	*load case index (0 < LoadCaseId ≤ AxisVMLoadCases.Count)*
long	**LineId**	*line index (0 < Index ≤ 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



RLoadTrussThermal = (

long	**LoadCaseId**	*load case index (0 < LoadCaseId ≤ AxisVMLoadCases.Count)*
long	**LineId**	*line index (0 < Index ≤ AxisVMLines.Count)*
double	**Tref**	*reference temperature*
double	**T0**	*actual truss temperature*

)

Beam loads

RLoadBeamConcentrated = (

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

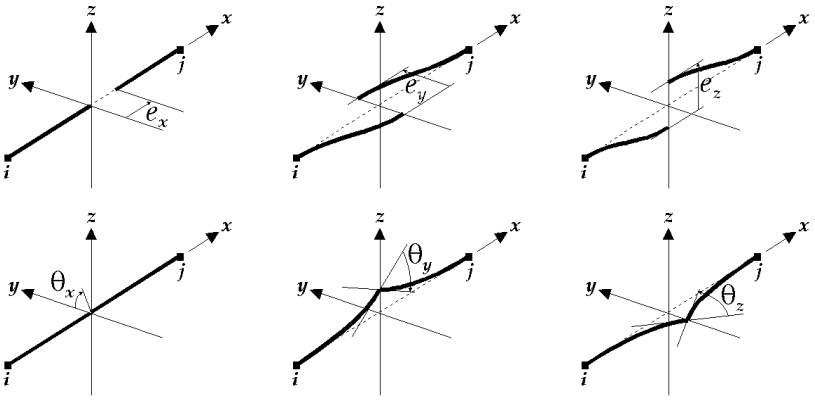
)

RLoadBeamFault = (

long	**LoadCaseId**	*load case index (0 < LoadCaseId ≤ AxisVMLoadCases.Count)*
long	**LineId**	*line index (0 < Index ≤ AxisVMLines.Count)*
double	**DL**	*fault in length*
	)	*See Fault in Length (Fabrication Error) in AxisVM manual*

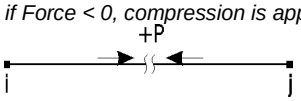
RLoadBeamInfluence = (

long	**LoadCaseId**	*load case index (0 < LoadCaseId ≤ AxisVMLoadCases.Count)*
long	**LineId**	*line index (0 < Index ≤ AxisVMLines.Count)*
double	**ex, ey, ez**	*relative displacements of the influence line load (–1, 0, 1)*
double	**Fx, Fy, Fz**	*relative rotations of the influence line load (–1, 0, 1)*
double	**Position**	*if Position ≥ 0, the beam influence [m],*
	)	*if Position < 0, the absolute value is beam influence /line length*



RLoadBeamStress = (

long	**LoadCaseId**	*load case index (0 < LoadCaseId ≤ AxisVMLoadCases.Count)*
long	**LineId**	*line index (0 < Index ≤ AxisVMLines.Count)*
double	**P**	*tension/compression force*
	)	*if Force > 0, tension is applied*
		if Force < 0, compression is applied



RLoadBeamThermal = (

long	**LoadCaseId**	*load case index (0 < LoadCaseId ≤ AxisVMLoadCases.Count)*
long	**LineId**	*line index (0 < Index ≤ AxisVMLines.Count)*
double	**Tref**	*reference temperature*
double	**Ttop**	*top cord temperature (in the Axis direction)*
double	**Tbot**	*bottom cord temperature (in the Axis direction)*
EAxis	**Axis**	*direction of temperature variation (in local y or z direction only)*
	)	

RLoadSurfaceToBeam = (

long	**LoadCaseId**	*load case index (0 < LoadCaseId ≤ AxisVMLoadCases.Count)*
EDistributionType	**DistributionType**	*type of the distributed load (global / local / projected)*
double	**Px, Py, Pz**	*surface load intensity [kN/m²]*
	)	

RLoadSurfaceToBeamAssoc = (

long	**LoadCaseId**	*load case index (0 < LoadCaseId ≤ AxisVMLoadCases.Count)*
EDistributionType	**DistributionType**	*a distributed load típusa (global / local / projected)*
double	**Px, Py, Pz**	*surface load intensity [kN/m²]*
	)	

Rib loads

RLoadRibConcentrated = (

long	**LoadCaseId**	*load case index (0 < LoadCaseId ≤ AxisVMLoadCases.Count)*
long	**LineId**	*line index (0 < Index ≤ 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 componenets*
		if SystemGLR = sysReference, only Fgx and Mgx are valid
	)	

RLoadRibDistributed = (

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

RLoadRibThermal = (

long	**LoadCaseId**	*load case index (0 < LoadCaseId ≤ AxisVMLoadCases.Count)*
long	**LineId**	*line index (0 < Index ≤ AxisVMLines.Count)*
double	**Tref**	*reference temperature*
double	**Ttop**	*top cord temperature (in the Axis direction)*
double	**Tbot**	*bottom cord temperature (in the Axis direction)*
EAxis	**Axis**	*direction of temperature variation (in local y or z direction only)*

)

Structural member loads

RLoadBeamMemberConcentrated = (

Warning! *This record was superseded by RLoadMemberConcentrated*

long	**LoadCaseId**	*load case index (0 < LoadCaseId ≤ AxisVMLoadCases.Count)*
long	**MemberId**	*member index*
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/member length*
ESystem	**SystemGLR**	*coordinate system of load componenets* *if SystemGLR = sysReference, only Fgx and Mgx are valid*

)

RLoadMemberConcentrated = (

long	**LoadCaseId**	*load case index (0 < LoadCaseId ≤ AxisVMLoadCases.Count)*
long	**MemberId**	*member index*
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/member length*
ESystem	**SystemGLR**	*coordinate system of load componenets* *if SystemGLR = sysReference, only Fgx and Mgx are valid*
EMemberLoadEcc	**EccentricityType**	*eccentricity type*
double	**UserEccY**	*eccentricity in cross-section y direction (only for EccentricityType mle_User) [m]*
double	**UserEccZ**	*eccentricity in cross-section z direction (only for EccentricityType mle_User) [m]*

)

double	**Position2**	*position of the 2nd point with the same sign convention*
EBeamRibDistributionTyoe	**DistributionType**	*distributed by length or projected*
ELongBoolean	**Trapezoid**	*trapezoid load*

)

RLoadRibMemberConcentrated = (

Warning! *This record was superseded by RLoadMemberConcentrated*

long	**LoadCaseId**	*load case index (0 < LoadCaseId ≤ AxisVMLoadCases.Count)*
long	**MemberId**	*member index*
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/member length*
ESystem	**SystemGLR**	*coordinate system of load componenets* *if SystemGLR = sysReference, only Fgx and Mgx are valid*

)

RLoadRibMemberDistributed = (

Warning! *This record was superseded by RLoadMemberDistributed*

long	**LoadCaseId**	*load case index (0 < LoadCaseId ≤ 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* *if Position ≥ 0, the load position [m],* *if Position < 0, the absolute value is load position/member length*
double	**Position2**	*position of the 2nd point with the same sign convention*
EBeamRibDistributionTyoe	**DistributionType**	*distributed by length or projected*
ELongBoolean	**Trapezoid**	*trapezoid load*

)

RLoadMemberDistributed = (

long	**LoadCaseId**	*load case index (0 < LoadCaseId ≤ 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* *if Position ≥ 0, the load position [m],* *if Position < 0, the absolute value is load position/member length*
double	**Position2**	*position of the 2nd point with the same sign convention*
EBeamRibDistributionTyoe	**DistributionType**	*distributed by length or projected*
ELongBoolean	**Trapezoid**	*trapezoid load*
EMemberLoadEcc	**EccentricityType**	*eccentricity type*
double	**UserEccY**	*eccentricity in cross-section y direction (only for EccentricityType mle_User) [m]*
double	**UserEccZ**	*eccentricity in cross-section z direction (only for EccentricityType mle_User) [m]*

)

Surface loads

RLoadSurfaceConcentrated = (

long	**LoadCaseId**	*load case index (0 < LoadCaseId ≤ AxisVMLoadCases.Count)*
long	**SurfaceId**	*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	**ReferenceId**	*If set to 0, load components are in global directions.* *If 0 < ReferenceId ≤ AxisVMReferences.Count, the load direction is set by the reference. ReferenceId is the index of reference according to AxisVMReferences.*
ESystem	**SystemGLR**	*coordinate system of load components (local or global only)*

)

RLoadSurfaceDistributed = (

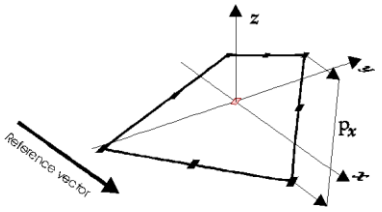
long	**LoadCaseId**	*load case index (0 < LoadCaseId ≤ AxisVMLoadCases.Count)*
long	**SurfaceId**	*surface index (0 < Index ≤ AxisVMSurfaces.Count)*
double	**qx, qy, qz**	*x, y, z force components [kN/m²]*
ESystem	**SystemGLR**	*coordinate system of load components (local or global only)*
ESurfaceDomainDistributionType	**DistributionType**	*type of the distributed load*

)

RLoadSurfaceEdge = (

long	**LoadCaseId**	*load case index (0 < LoadCaseId ≤ AxisVMLoadCases.Count)*
long	**SurfaceId**	*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, System3, System4**	*load coordinate systems on edges (local or global only)*
ELongBoolean	**EdgeLoaded1, EdgeLoaded2, EdgeLoaded3, EdgeLoaded4**	*True, if a load is applied on the edge connecting corner nodes* *(For quadrilateral elements: 1 → 2, 2 → 3, 3 → 4, 4 → 1)* *(For triangular elements: 1 → 2, 2 → 3, 3 → 1)* *For triangular elements the fourth component is ignored.* *edge load in local x direction*
ESurfaceDomainDistributionType	**DistributionType1, DistributionType2, DistributionType3, DistributionType4**	*type of the distributed load*

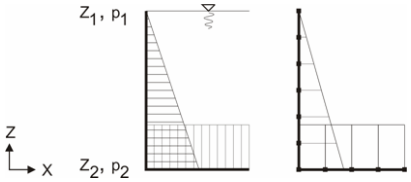
)



RLoadSurfaceFluid = (

long	**LoadCaseId**	*load case index (0 < LoadCaseId ≤ AxisVMLoadCases.Count)*
long	**SurfaceId**	*surface index (0 < Index ≤ AxisVMSurfaces.Count)*
EAxis	**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/m²]*

)



RLoadSurfaceThermal = (

long	**LoadCaseId**	*load case index (0 < LoadCaseId ≤ AxisVMLoadCases.Count)*
long	**SurfaceId**	*surface index (0 < Index ≤ AxisVMSurfaces.Count)*
double	**Tref**	*reference temperature*
double	**Ttop**	*top temperature according to the local z direction*
double	**Tbot**	*bottom temperature according to the local z direction*

)

RLoadSurfaceSelfWeight = (

long	**LoadCaseId**	*load case index (0 < LoadCaseId ≤ AxisVMLoadCases.Count)*
long	**SurfaceId**	*surface index (0 < Index ≤ AxisVMSurfaces.Count)*

)

Domain loads

RLoadDomainPolyArea = (

long	**LoadCaseId**	*load case index (0 < LoadCaseId ≤ AxisVMLoadCases.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*
double	**P1, P2, P3**	*load intensity in [kN/m²]. If LoadDistributionType is constant then. P1, P2, P3 is the load intensity in x,y,z directions, in accordance with DistributionType (i.e. local or global).* ***Warning!*** *AddDomainPolyArea handles it differently. If LoadDistributionType is linear, P1, P2, P3 will define the load intensity at the three load reference points.*
double	**x1, y1, z1**	*coordinates of the 1st load reference point (ignored for LoadDistributionType = ldtConst)*
double	**x2, y2, z2**	*coordinates of the 2nd load reference point (ignored for LoadDistributionType = ldtConst)*
double	**x3, y3, z3**	*coordinates of the 3rd load reference point (ignored for LoadDistributionType = ldtConst)*
ELongBoolean	**WindowLoad**	*window load (load falling on an opening is distributed along the edges)*

)

RLoadDomainLinear = (

long	**LoadCaseId**	*load case index (0 < LoadCaseId ≤ AxisVMLoadCases.Count)*
long	**DomainId**	*domain index (0 < Index ≤ AxisVMDomains.Count)*
EDistributionType	**DistributionType**	*type of the distributed load (global / local / projected)*
ELoadDistributionType	**LoadDistributionType**	*should always be ldtLinear. To define constant intensity loads use the RLoadDomainConstant record type instead, with its corresponding functions*
EAxis	**Component**	*load component*
double	**P1, P2, P3**	*for the only valid LoadDistributionType, ldtLinear, they are the load intensity at the load reference points [kN/m²]*
double	**x1, y1, z1**	*global coordinates of the 1st load reference point*

double	**x2, y2, z2**	*global coordinates of the 2nd load reference point*
double	**x3, y3, z3**	*global coordinates of the 3rd load reference point*
ELongBoolean	**WindowLoad**	*window load (load falling on an opening is distributed along the edges)*

)

RLoadDomainConcetrated = (

long	**LoadCaseId**	*load case index (0 < LoadCaseId ≤ AxisVMLoadCases.Count)*
long	**DomainId**	*domain index (0 < Index ≤ 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	**ReferenceId**	*If set to 0, load components are in global directions. If 0 < ReferenceId ≤ AxisVMReferences.Count, the load direction is set by the reference. ReferenceId is the index of reference according to AxisVMReferences.*
ESystem	**SystemGLR**	*load coordinate system*

)

RLoadDomainConstant = (

Warning! This record has become obsolete, it was superseded by RLoadDomainConstant_V154

long	**LoadCaseId**	*load case index (0 < LoadCaseId ≤ AxisVMLoadCases.Count)*
long	**DomainId**	*domain index (0 < Index ≤ AxisVMDomains.Count)*
double	**qx, qy, qz**	*area loads in x, y, z directions [kN/m²],*
ESurfaceDomainDistributionType	**DistributionType**	*type of distributed load (surface / projected)*
ESystem	**SystemGLR**	*load coordinate system*

)

RLoadDomainConstant_V154 = (

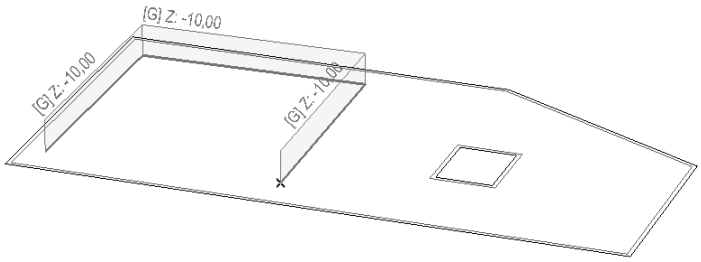
long	**LoadCaseId**	*load case index (0 < LoadCaseId ≤ AxisVMLoadCases.Count)*
long	**DomainId**	*domain index (0 < Index ≤ AxisVMDomains.Count)*
double	**qx, qy, qz**	*area loads in x, y, z directions [kN/m²],*
ESurfaceDomainDistributionType	**DistributionType**	*type of distributed load (surface / projected)*
ESystem	**SystemGLR**	*load coordinate system*
ELongBoolean	**WindowLoad**	*window load (load falling on an opening is distributed along the edges)*

)

RLoadDomainPolyLine = (

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*

)



RLoadDomainPolyAssoc = (

Warning! *This record was incomplete, it superseded by RloadDomainPolyAssoc_V161*

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*

)

RloadDomainPolyAssoc_V161 = (

long	**LoadCaseId**	*load case index (0 < LoadCaseId ≤ AxisVMLoadCases.Count)*
long	**MemberId**	*member index (0 < MemberId ≤ AxisVMMembers.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)*
Rpoint3D	**NormalVector**	*normal vector for the plane of the load (it will search for domains only in that plane)*

)

RCircleArcGeomData	**CircleArcGeomData**	*Circle's or arc's geometric data*

)

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*

)

RLoadDomainSelfWeight = (

long	**LoadCaseId**	*load case index (0 < LoadCaseId ≤ AxisVMLoadCases.Count)*
long	**DomainId**	*domain index (0 < Index ≤ AxisVMDomains.Count)*

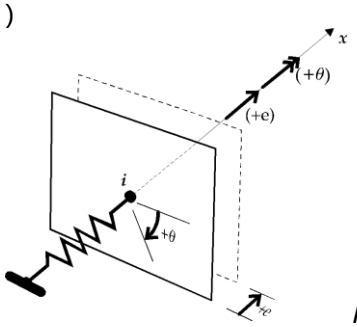
)

Support loads

RLoadSupportDisplacement = (

long	**LoadCaseId**	*load case index (0 < LoadCaseId ≤ AxisVMLoadCases.Count)*
long	**SupportId**	*nodal support index (0 < Index ≤ AxisVMNodalSupports.Count)*
double	**ex, ey, ez**	*support displacement in x, y, z directions*
double	**fx, fy, fz**	*support rotation about the x, y, z axis*

)



$$P_{support} = K_{support} \cdot e$$

RLoadDynamic = (

ELoadType	**DynamicLoadType**	*type of dynamic load, see here*
long	**LoadCaseId**	*load case index*
long	**NodeId**	*node index*
double	**Fx**	*nodal force in x direction*
double	**Fy**	*nodal force in y direction*
double	**Fz**	*nodal force in z direction*
double	**Mx**	*nodal moment about x direction*
double	**My**	*nodal moment about y direction*
double	**Mz**	*nodal moment about z direction*
long	**ReferenceId**	*index of the reference*
long	**FxFunctionId**	*IAxisVMDynamicLoadFunctions index for force in x direction*
long	**FyFunctionId**	*IAxisVMDynamicLoadFunctions index for force in y direction*
long	**FzFunctionId**	*IAxisVMDynamicLoadFunctions index for force in z direction*
long	**MxFunctionId**	*IAxisVMDynamicLoadFunctions index for moment about x direction*
long	**MyFunctionId**	*IAxisVMDynamicLoadFunctions index for moment about y direction*
long	**MzFunctionId**	*IAxisVMDynamicLoadFunctions index for moment about z direction*

)

Load panel loads

RLoadPanelPolyArea = (

long	**LoadCaseId**	*load case index (0 < LoadCaseId ≤ AxisVMLoadCases.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 in [kN/m²]. If LoadDistributionType is constant then. P1, P2, P3 is the load intensity in x,y,z directions, in accordance with DistributionType (i.e. local or global).* ***Warning!*** *AddLoadPanelPolyArea handles it differently.* *If LoadDistributionType is linear, P1, P2, P3 will define the load intensity at the three load reference points.*
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*
ELongBoolean	**WindowLoad**	*window load (load falling on an opening is distributed along the edges)*

)

RLoadPanelConcentrated = (

long	**LoadCaseId**	*load case index (0 < LoadCaseId ≤ AxisVMLoadCases.Count)*
long	**LoadPanelId**	*load panel index (0 < Index ≤ AxisVMLoadPanels.Count)*
double	**Fx, Fy, Fz**	*force in x, y, z directions*
double	**Mx, My, Mz**	*moment in x, y, z directions*
double	**x, y, z**	*x, y, z coordinates*
ESystem	**SystemGLR**	*load coordinate system*
long	**ReferenceId**	*If set to 0, load components are in global directions.* *If 0 < ReferenceId ≤ AxisVMReferences.Count, the load direction is set by the reference. ReferenceId is the index of reference according to AxisVMReferences.*

)

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 in [kN/m²]. If LoadDistributionType is constant then. P1, P2, P3 is the load intensity in x,y,z directions, in accordance with DistributionType (i.e. local or global).* ***Warning!*** *AddLoadPanelLinear handles it differently.* *If LoadDistributionType is linear, P1, P2, P3 will define the load intensity at the three load reference points.*
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*

)

RLoadPanelLinear_V162 = (

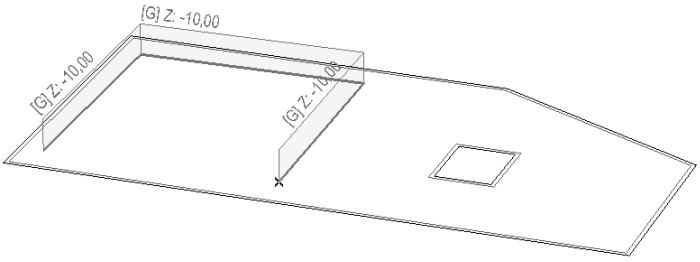
long	**LoadCaseId**	*load case index (0 < LoadCaseId ≤ AxisVMLoadCases.Count)*
long	**LoadPanelId**	*load panel index (0 < Index ≤ AxisVMLoadPanels.Count)*
EDistributionType	**DistributionType**	*type of the distributed load (global / local / projected)*
ELoadDistributionType	**LoadDistributionType**	*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 in [kN/m²]. If LoadDistributionType is constant then. P1, P2, P3 is the load intensity in x,y,z directions, in accordance with DistributionType (i.e. local or global). If LoadDistributionType is linear, P1, P2, P3 will define the load intensity at the three load reference points.*
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*
ELongBoolean	**WindowLoad**	*window load (load falling on an opening is distributed along the edges)*

)

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*

)



General line loads

RLoadLineSelfWeigth = (

long	**LoadCaseId**	*load case index (0 < LoadCaseId ≤ AxisVMLoadCases.Count)*
long	**LineId**	*line index (0 < LineId ≤ AxisVMLines.Count)*

)

RLoadLineDistributed = (

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

)

Nodal mass

RNodalMass = (

long	**Node**	*node index (0 < Index ≤ AxisVMNodes.Count)*
double	**mx**	*mass in x direction [kg]*
double	**my**	*mass in y direction [kg]*
double	**mz**	*mass in z direction [kg]*

)

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 **SafeArrayAccesssData** COM function to get the pointer to the array data.

```
HRESULT SafeArrayAccesssData(
        SAFEARRAY* psa                  psa is LoadData obtained from GetLoad
        void HUGEP** ppvData);          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 **SafeArrayUnAccesssData**.

```
HRESULT SafeArrayUnAccesssData(
        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.

```
RetValue = 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.

```
RetValue = 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                          vt = VT_UI1
        unsigned long cDims                 cDims = 1
        SAFEARRAYBOUND* rgsabound           rgsabound
                                                unsigned long cElements = SizeOf(Record)
                                                long lLBound = 1
        PVOID pvExtra );                    ppRecInfo obtained from the previous function
```

2. Call **SafeArrayAccesssData** COM function to get the pointer to the array data.

```
HRESULT SafeArrayAccesssData(
        SAFEARRAY* psa                      psa is the return value of SafeArrayCreateEx
        void HUGEP** ppvData);              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.

```
RetValue = 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:

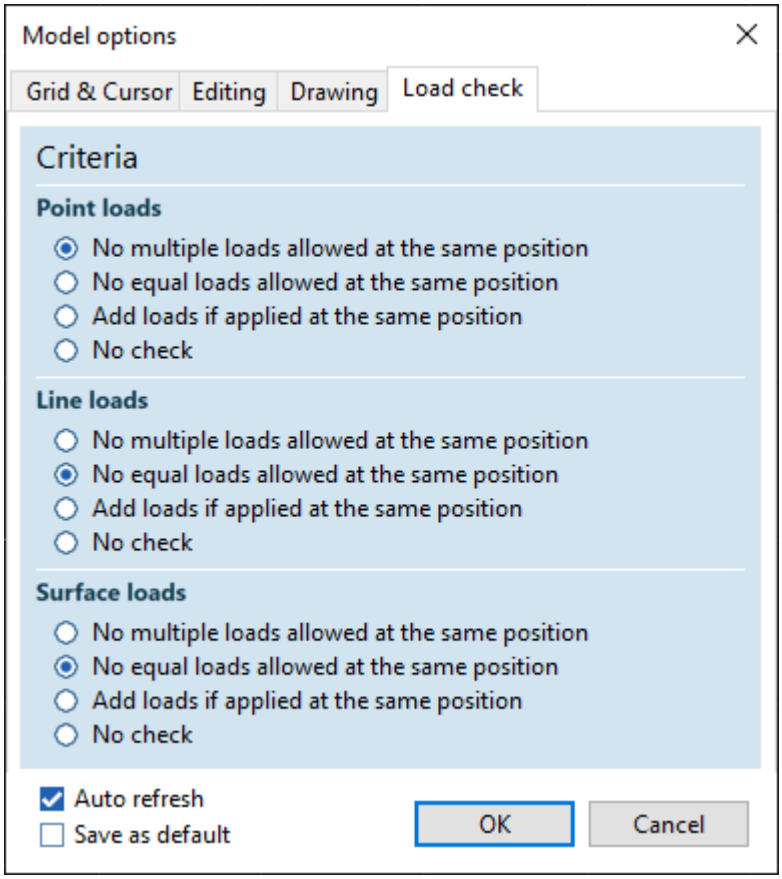
```
Public Declare Sub CopyMemory Lib "kernel32" Alias "RtlMoveMemory" _
                   (Destination As Any, Source As Any, ByVal Length As Long)
```

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

Depending on the settings in the window below,loads may be deleted or altered at some points (for example before analysis) :



For example, in the case of “No equal loads allowed at the same position”, 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 e.g. 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, loeInvalidLoad).

IMPORTANT NOTE: If the added load meets this, then it will be deleted before analysis!

long **AddNodalForces** ([in] SAFEARRAY(RLoadNodalForce)* **NodalForces** [out] SAFEARRAY(long)* **Indexes**)

NodalForces *list of nodal force records*

Indexes *list of the load indexes that were created. The first index is for the first element of NodalForces, and so on. If a particular record contained invalid data, that load won't be added to the database, and an index of 0 will be returned at the place corresponding to that record*

Creates several nodal loads in one call. It is faster than successive calls of AddNodalForce. Returns the count of created loads (which can be lower than the length of NodalForces). Can trigger the following errors through events (errIndexOutOfBounds, errDatabaseNotReady, loeLoadComponentMustBeZero, loeInvalidLoad).

IMPORTANT NOTE: If an 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 **LineId,** [in] long **LoadCaseId**)

LineId *line index (truss)*
(0 < Index ≤ AxisVMLines.Count)

LoadCaseId *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, loeInvalidLoad).

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, loeInvalidLoad).

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, loeInvalidLoad).

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, loeInvalidLoad)

IMPORTANT NOTE:

If the added load meets this, then it will be deleted before analysis!

---

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

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

LoadCaseId *load case index*
(0 < Index ≤ 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, loeInvalidLoad).

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)* **LineIds,**
[i/o] RLoadSurfaceToBeam **Data**)

ContourPoly *closed polygon of the surface load to distribute*

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

LineIds *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*

Data *load data*

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).

IMPORTANT NOTE:

If the added load meets this, then it will be deleted before analysis!

---

long **AddSurfaceToBeam_vb** (Visual Basic compatible function of **AddSurfaceToBeam**)

---

long **AddSurfaceToBeamAssoc** ([in] SAFEARRAY(long)* **ContourLineIds,** [in] ELongBoolean **AutoFindLines,** [in] SAFEARRAY(long)* **LineIds,** [i/o] RLoadSurfaceToBeamAssoc **Data**)

ContourLineIds *line indexes of contour lines of the closed polygon*

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

LineIds *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*

Data *load data*

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).

IMPORTANT NOTE:

If the added load meets this, then it will be deleted before analysis!

---

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 transfered, explained here.

long **AddRibConcentrated** ([i/o] RLoadRibConcentrated **Data**)

Data *load data*

Creates a concentrated load on a rib. If successful, returns load index, otherwise returns an error code (leInvalidLineType, errIndexOutOfBounds, errDatabaseNotReady, loeInvalidLoad).

IMPORTANT NOTE:

If the added load meets this, then it will be deleted before analysis!

---

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, loeInvalidLoad).

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 ≤ AxisVMLines.Count)

LoadCaseId *load case index*
(0 < Index ≤ 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, loeInvalidLoad).

IMPORTANT NOTE:

If the added load meets this, then it will be deleted before analysis!

General line load

long **AddLineDistributeds** ([in] SAFEARRAY(RLoadLineDistributed)* **LineDistributeds** [out] SAFEARRAY(long)* **Indexes**)

LineDistributeds *list of line distributed load records*

Indexes *list of the load indexes that were created. The first index is for the first element of LineDistributeds, and so on. If a particular record contained invalid data, that load won't be added to the database, and an index of 0 will be returned at the place corresponding to that record*

Creates several line loads in one call. It is faster than successive calls to singular creations. The lines should be only of the types that can have distributed load (i.e. beam or rib). Returns the count of created loads (which can be lower than the length of LineDistributeds). Can trigger the following errors through events (errIndexOutOfBounds, errDatabaseNotReady, leInvalidLineType, loeInvalidLoadCaseType, loeInvalidLoad).

long **AddLineSelfWeights** ([in] SAFEARRAY(RLoadLineSelfWeigth)* **LineSelfWeigths** [out] SAFEARRAY(long)* **Indexes**)

LineSelfWeigths *list of line self weight records*

Indexes *list of the load indexes that were created. The first index is for the first element of LineSelfWeigths, and so on. If a particular record contained invalid data, that load won't be added to the database, and an index of 0 will be returned at the place corresponding to that record*

Creates several line loads in one call. It is faster than successive calls of the corresponding AddSelfWeight. The lines should be only of the types that can have self weight (i.e. beam, rib or truss). Returns the count of created loads (which can be lower than the length of LineSelfWeigths). Can trigger the following errors through events (errIndexOutOfBounds, errDatabaseNotReady, leInvalidLineType).

Add Structural member loads

long **AddBeamMemberConcentrated** ([i/o] RLoadBeamMemberConcentrated **Data**)

Warning! *This function has become obsolete, was superseded by AddMemberConcentrated*

Data *load data*

Creates a concentrated load on a beam structural member. If successful, returns load index, otherwise returns an error code (loeInvalidLoadCaseType, leInvalidLineType, errIndexOutOfBounds, errDatabaseNotReady, loeInvalidLoad).

IMPORTANT NOTE:

If the added load meets this, then it will be deleted before analysis!

long **AddBeamMemberDistributed** ([i/o] RLoadBeamMemberDistributed **Data**)

Warning! *This function has become obsolete, was superseded by AddMemberDistributed*

Data *load data*

Creates a distributed load on a beam structural member. If successful, returns load index, otherwise returns an error code (loeInvalidLoadCaseType, leInvalidLineType, errIndexOutOfBounds, errDatabaseNotReady, loeInvalidLoad).

IMPORTANT NOTE:

If the added load meets this, then it will be deleted before analysis!

long **AddMemberConcentrated** ([i/o] RLoadMemberConcentrated **Data**)

Data *load data*

Creates a concentrated load on a beam or rib structural member. If successful, returns load index, otherwise returns an error code (loeInvalidLoadCaseType, leInvalidLineType, errIndexOutOfBounds, errDatabaseNotReady, loeInvalidLoad).

IMPORTANT NOTE:

If the added load meets this, then it will be deleted before analysis!

---

long **AddMemberDistributed** ([i/o] RLoadMemberDistributed **Data**)

Data *load data*

Creates a distributed load on a beam or rib structural member. If successful, returns load index, otherwise returns an error code (loeInvalidLoadCaseType, leInvalidLineType, errIndexOutOfBounds, errDatabaseNotReady, loeInvalidLoad).

IMPORTANT NOTE:

If the added load meets this, then it will be deleted before analysis!

---

long **AddRibMemberConcentrated** ([i/o] RLoadRibMemberConcentrated **Data**)

Warning! *This function has become obsolete, was superseded by AddMemberConcentrated*

Data *load data*

Creates a concentrated load on a rib structural member. If successful, returns load index, otherwise returns an error code (loeInvalidLoadCaseType, leInvalidLineType, errIndexOutOfBounds, errDatabaseNotReady, loeInvalidLoad).

IMPORTANT NOTE:

If the added load meets this, then it will be deleted before analysis!

---

long **AddRibMemberDistributed** ([i/o] RLoadRibMemberDistributed **Data**)

Warning! *This function has become obsolete, was superseded by AddMemberDistributed*

Data *load data*

Creates a distributed load on a rib structural member. If successful, returns load index, otherwise returns an error code (loeInvalidLoadCaseType, leInvalidLineType, errIndexOutOfBounds, errDatabaseNotReady, loeInvalidLoad).

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*

Creates a concentrated load on a surface element. If successful, returns load index, otherwise returns an error code (errIndexOutOfBounds, errDatabaseNotReady, leErrorAddingLoad, loeInvalidLoad).

IMPORTANT NOTE: If the added load meets this, then it will be deleted before analysis!

---

long **AddSurfaceDistributed** ([i/o] RLoadSurfaceDistributed **Data**)

Data *load data*

Creates a distributed load on a surface element. If successful, returns load index, otherwise returns an error code (errIndexOutOfBounds, errDatabaseNotReady, loeInvalidLoad).

IMPORTANT NOTE: If the added load meets this, then it will be deleted before analysis!

---

long **AddSurfaceDistributeds** ([in] SAFEARRAY(RLoadSurfaceDistributed)* **SurfaceDistributeds** [out] SAFEARRAY(long)* **Indexes**)

SurfaceDistributeds *list of surface distributed load records*

Indexes *list of the load indexes that were created. The first index is for the first element of SurfaceDistributeds, and so on. If a particular record contained invalid data, that load won't be added to the database, and an index of 0 will be returned at the place corresponding to that record*

Creates several surface distributed loads in one call. It is faster than successive calls to . AddSurfaceDistributed. Returns the count of created loads (which can be lower than the length of SurfaceDistributeds). Can trigger the following errors through events (errIndexOutOfBounds, errDatabaseNotReady, loeInvalidLoadCaseType, loeInvalidLoad).

---

long **AddSurfaceEdge** ([i/o] RLoadSurfaceEdge **Data**)

Data *load data*

Creates an edge load on a surface element. 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 **AddSurfaceFluid** ([i/o] RLoadSurfaceFluid **Data**)

Data *load data*

Creates a fluid load on a surface element. 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 **AddSurfaceSelfWeight** ([in] long **SurfaceId,** [in] long **LoadCaseId**)

SurfaceId *surface element index*
(0 < Index ≤ AxisVMSurfaces.Count)

LoadCaseId *load case index*
(0 < Index ≤ AxisVMLoadCases.Count)

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

---

long **AddSurfaceSelfWeights** ([in] SAFEARRAY(RLoadSurfaceSelfWeigth)* **SurfaceSelfWeigths** [out] SAFEARRAY(long)* **Indexes**)

SurfaceSelfWeigths *list of surface self weight records*

Indexes *list of the load indexes that were created. The first index is for the first element of SurfaceSelfWeigths, and so on. If a particular record contained invalid data, that load won't be added to the database, and an index of 0 will be returned at the place corresponding to that record*

Creates several surface self weight loads in one call. It is faster than successive calls of the corresponding AddSurfaceSelfWeight. Returns the count of created loads (which can be lower than the length of SurfaceSelfWeigths). Can trigger the following errors through events (errIndexOutOfBounds, errDatabaseNotReady).

---

long **AddSurfaceThermal** ([i/o] RLoadSurfaceThermal **Data**)

Data *load data*

Creates a thermal load on a surface element. If successful, returns load index, otherwise returns an error code (errIndexOutOfBounds, errDatabaseNotReady, loeInvalidLoad).

IMPORTANT NOTE: If the added load meets this, then it will be deleted before analysis!

Add Domain loads

long **AddDomainPolyArea** ([in] AxisVMLines3d * **ContourPoly,**
[i/o] RLoadDomainPolyArea **Data**)

ContourPoly *a closed polygon*
Data *load data.* ***Warning!*** *For LoadDistributionType ldtConst only P1 is taken into account, for a load intensity in the Component direction. To define constant load distributions in multiple directions, modify the load created in AddDomainPolyArea with a subsequent call to SetLoad, where the P1, P2, P3 values will be taken into account as described at RLoadDomainPolyArea.*

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).

IMPORTANT NOTE: If the added load meets this, then it will be deleted before analysis!

---

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 (errIndexOutOfBounds, errDatabaseNotReady, loeInvalidLoad).

IMPORTANT NOTE: If the added load meets this, then it will be deleted before analysis!

---

long **AddDomainLinears** ([in] SAFEARRAY(RLoadDomainLinear)* **DomainLinears** [out] SAFEARRAY(long)* **Indexes**)

DomainLinears *list of domain linearly variable surface load records*
Indexes *list of the load indexes that were created. The first index is for the first element of DomainLinears, and so on. If a particular record contained invalid data, that load won't be added to the database, and an index of 0 will be returned at the place corresponding to that record*

Creates several domain linearly variable surface loads in one call. Each load will act on the whole area of the domain. It is faster than successive calls to AddDomainLinear. Returns the count of created loads (which can be lower than the length of DomainLinears). Can trigger the following errors through events (errIndexOutOfBounds, errDatabaseNotReady, loeInvalidLoadCaseType, loeInvalidLoad).

---

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, loeInvalidLoad).

IMPORTANT NOTE: If the added load meets this, then it will be deleted before analysis!

---

long **AddDomainConstant** ([i/o] RLoadDomainConstant **Data**)

Warning! This record has become obsolete, it was superseded by AddDomainConstant_V154

Data *load data*

Creates a distributed load on a domain. If successful, returns load index, otherwise returns an error code (errIndexOutOfBounds, errDatabaseNotReady, loeInvalidLoad).

IMPORTANT NOTE: If the added load meets this, then it will be deleted before analysis!

---

long **AddDomainConstant_V154** ([i/o] RLoadDomainConstant_V154 **Data**)

Data *load data*

Creates a distributed load on a domain, acting on it's whole area. If successful, returns load index, otherwise returns an error code (errIndexOutOfBounds, errDatabaseNotReady, loeInvalidLoad).

IMPORTANT NOTE: If the added load meets this, then it will be deleted before analysis!

---

long **AddDomainConstants** ([in] SAFEARRAY(RLoadDomainConstant_V154)* **DomainConstants** [out] SAFEARRAY(long)* **Indexes**)

DomainConstants *list of domain constant intensity surface load records*
Indexes *list of the load indexes that were created. The first index is for the first element of DomainConstants, and so on. If a particular record contained invalid data, that load won't be added to the database, and an index of 0 will be returned at the place corresponding to that record*

Creates several domain constant intensity surface loads in one call. Each load will act on the whole area of the domain. It is faster than successive calls to AddDomainConstant. Returns the count of created loads (which can be lower than the length of DomainConstants). Can trigger the following errors through events (errIndexOutOfBounds, errDatabaseNotReady, loeInvalidLoadCaseType, loeInvalidLoad).

---

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!

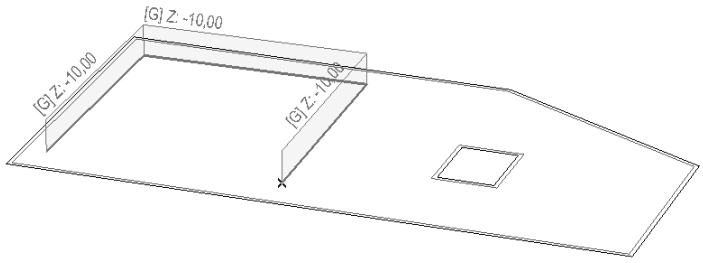
---

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**)

Warning! *This function has become obsolete, was superseded by AddDomainPolyAssoc_V161*

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 might be deleted before analysis!

---

long **AddDomainPolyAssoc_V161** ([i/o] RloadDomainPolyAssoc_V161 **Data**)

Data *load data*

Creates an associative line load on a domain. Load position changes with the location of the member it is associated to. If successful, returns load index, otherwise returns an error code (leErrorAddingLoad, errIndexOutOfBounds, errDatabaseNotReady).

IMPORTANT NOTE: If the added load meets this, then it might be deleted before analysis!

---

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

DomainId *domain index*
(0 < Index ≤ AxisVMDomains.Count)

LoadCaseId *load case index*
(0 < Index ≤ AxisVMLoadCases.Count)

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

---

long **AddDomainSelfWeights** ([in] SAFEARRAY(RLoadDomainSelfWeight)* **DomainSelfWeigths** [out] SAFEARRAY(long)* **Indexes**)

DomainSelfWeigths *list of domain self weight records*

Indexes *list of the load indexes that were created. The first index is for the first element of DomainSelfWeigths, and so on. If a particular record contained invalid data, that load won't be added to the database, and an index of 0 will be returned at the place corresponding to that record*

Creates several domain self weight loads in one call. It is faster than successive calls of the corresponding AddDomainSelfWeight. Returns the count of created loads (which can be lower than the length of DomainSelfWeigths). Can trigger the following errors through events (errIndexOutOfBounds, errDatabaseNotReady).

---

long **AddDomainThermal** ([i/o] RLoadDomainThermal **Data**)

Data *load data*

Creates a thermal laod on a domain. If successful, returns load index, otherwise returns an error code (errIndexOutOfBounds, errDatabaseNotReady, loeInvalidLoad).

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*

Creates a support displacement. If successful, returns load index, otherwise returns an error code (errIndexOutOfBounds, errDatabaseNotReady, loeInvalidLoad).

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*

Adds dynamic load. If successful, returns Index, otherwise returns an error code (errIndexOutOfBounds, errDatabaseNotReady, loeReferenceIndexOutOfBounds, loeInvalidLoadCaseType, loeNotDynamicLoadCase, leLoadCaseIndexOutOfBounds, leInvalidLoadType or loeDYNmoduleNotAvailable, loeInvalidLoad).

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.* ***Warning!*** *For LoadDistributionType ldtConst only P1 is taken into account, for a load intensity in the Component direction. To define constant load distributions in multiple directions, modify the load created with AddLoadPanelPolyArea with a subsequent call to SetLoad, where the P1, P2, P3 values will be taken into account as described at RLoadPanelPolyArea.*

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).

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*

Adds concentrated 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 **AddLoadPanelLinear** ([i/o] RLoadPanelLinear * **Data**)

Data *parameters of linear load on a load panel* ***Warning!*** *For LoadDistributionType ldtConst only P1 is taken into account, for a load intensity in the Component direction. To define constant load distributions in multiple directions, modify the load created in AddLoadPanelLinear with a subsequent call to SetLoad, where the P1, P2, P3 values will be taken into account as described at RLoadPanelLinear.*

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 **AddLoadPanelLinear_V162** ([i/o] RLoadPanelLinear_V162 * **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!

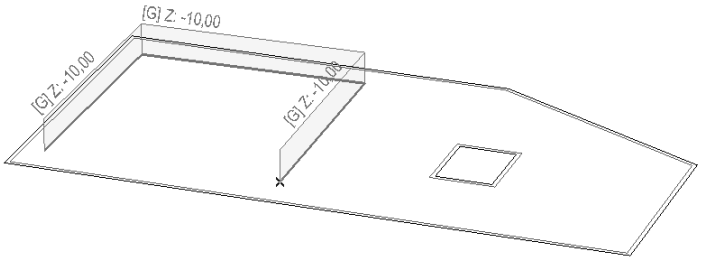
---

long **AddLoadPanelPolyLine** ([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!



Nodal masses

long **AddNodalMass** ([in] RNodalMass * **NodalMass**)

NodalMass *nodal mass definition data*

Creates a new nodal mass. If successful, returns a positive value, otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBounds, loeInvalidLoad).

long **DeleteNodalMass** ([in] long **Node**)

Node *node index (0 < Index ≤* AxisVMNodes.*Count)*

Deletes the nodal mass from a node. If successful, returns a positive value, otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBounds, loeNoMassLoadOnNode).

long **ModifyNodalMass** ([in] RNodalMass * **NodalMass**)

NodalMass *nodal mass definition data*

Modifies an already existing nodal mass. If successful, returns a positive value, otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBounds, loeInvalidLoad, loeNoMassLoadOnNode).

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, loeLoadPanelIndexlListEmpty 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 leThereAreNoSeismicStoreys).

---

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**)

Warning! *This function has become obsolete, see IAxisVMWindLoad for handling wind loads*

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, loeLoadPanelIndexlListEmpty or loeNoWindLoadCase).

---

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

Index *index of the load to delete (0 < Index ≤ 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 loeLoadPanelIndexlListEmpty).

---

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**)

Warning! *This function has become obsolete, see IAxisVMWindLoad for handling wind loads*

LoadPanelIDs *Load panel indexes*

Delete wind loads from the load panels in array.
If successful, returns Index, otherwise returns an error code (errIndexOutOfBounds, errDatabaseNotReady or loeLoadPanelIndexlListEmpty).

---

long **DeleteWindLoadFromAllLoadPanels**

Warning! *This function has become obsolete, see IAxisVMWindLoad for handling wind loads*

Delete wind loads from all load panels where it was applied.
If successful, returns Index, otherwise returns an error code (errDatabaseNotReady).

---

long **GetDomains_Surfaces_LoadPanels** ([in] long **Index,** [out] SAFEARRAY(long)* **DomainIds,**
[out] SAFEARRAY(long)* **SurfaceIds,** [out] SAFEARRAY(long)* **LoadPanelIDs**)

Index *load index (0 < Index ≤ Count)*
DomainIds *index of domains where load is applied*
SurfaceIds *index of surfaces where load is applied*
LoadPanelIDs *index of load panels where load is applied*

Get indexes of domains, surfaces and load panels where load is applied.Can be used with load types ltDomainPolyAssoc and ltDomainPolyArea. If successful, returns index, otherwise returns an error code (leInvalidLoadType, errIndexOutOfBounds, errDatabaseNotReady)

---

long **GetDomainPolyLineItems** ([in] long **Index,**
[out] SAFEARRAY(RLoadDomainPolyLineItem)* **Items**)

Index *load index (0 < Index ≤ Count)*
Items *items (parts) of the load*

Get all load data of the load type ltLoadDomainPolyLine. If successful, returns number of items, otherwise returns an error code (leInvalidLoadType, errIndexOutOfBounds, errDatabaseNotReady)

---

long **GetLoadPanelsOfSnowLoad** ([out] SAFEARRAY(long)* **PitchedLoadPanelIDs,**
[out] SAFEARRAY(long)* **BarrelLoadPanelIDs**)

PitchedLoadPanelIDs *Load panel indexes on pitched roof*
BarrelLoadPanelIDs *Load panel indexes on barrel roof*

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)

---

long **GetLoadPanelsOfWindLoad** ([out] SAFEARRAY(long)* **LoadPanelIDs)**

Warning! *This function has become obsolete, see IAxisVMWindLoad for handling wind loads*

LoadPanelIDs *Load panel indexes on pitched roof*

Get load panel indexes where wind load was applied. If successful, returns length of array of load panels, otherwise returns an error code (errDatabaseNotReady)

---

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

Index *load index (0 < Index ≤ Count)*
LineIds *lines participating in the distribution process according to AxisVMLines*

(only for ltSurfaceToBeam or ltSurfaceToBeamAssoc) Obtains the line indexes participating in the distribution process. If successful, returns Index, otherwise returns an error code (leInvalidLoadType, errIndexOutOfBounds, errDatabaseNotReady)

---

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

Index *load index (0 < Index ≤ Count)*
LoadData *a pointer to the load record in SAFEARRAY format*

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

---

long **GetPoly** ([in] long **Index,** [out] AxisVMLines3d* **Poly**)

Index *load index (0 < Index ≤ Count)*
Poly *the load polygon. For ltDomainPolyArea and ltSurfaceToBeam types it is a cloased polygon, for ltLoadDomainPolyLine it is a polyline.*

Reads the load polygon of load types: ltDomainPolyArea, ltLoadDomainPolyLine and ltSurfaceToBeam types only. If successful, returns Index, otherwise returns an error code(leInvalidLoadType, errIndexOutOfBounds, errDatabaseNotReady)

---

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

ItemIds *list of selected loads*

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

---

long **SetLines** ([in] long **Index,** [in] SAFEARRAY(long)* **LineIds**)

Index *load index (0 < Index ≤ Count)*
LineIds *lines participating in the distribution process according to AxisVMLines*

(only for ltSurfaceToBeam or ltSurfaceToBeamAssoc) Set the line indexes participating in the distribution process. If successful, returns Index, otherwise returns an error code (leInvalidLoadType, leErrorSettingLines, errIndexOutOfBounds, errDatabaseNotReady)

---

long **SetLines_vb** (Visual Basic compatible function of **SetLines**)

---

long **SetLoad** ([in] long **Index,** [in] SAFEARRAY* **LoadData**)

Index *load index (0 < Index ≤ Count)*
LoadData *a pointer to the load record in SAFEARRAY format*

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.

---

long **SetLoad_vb** (Visual Basic compatible function of **SetLoad**)

---

long **SetPoly** ([in] long **Index,** [in] AxisVMLines3d* **Poly**)

Index *load index (0 < Index ≤ Count)*
Poly *the load polygon. For ltDomainPolyArea and ltSurfaceToBeam types it is a closed polygon, for ltLoadDomainPolyLine it is a polyline.*

Set the load polygon only for ltDomainPolyArea, ltLoadDomainPolyLine, ltSurfaceToBeam types. If successful, returns Index, otherwise returns an error code (leInvalidLoadType, leErrorSettingPoly, errIndexOutOfBounds, errDatabaseNotReady)

Properties

long **Count** *Get number of loads in the model*

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

long **LoadCaseId** [long **Index]** • *Get or set load case index of a load*

ELoadType **LoadType** [long **Index**] *Get type of a load.*

IAxisVMLogicalParts


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 = 0,	*Membrane type of domains*
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 rampt (other), only for domains*

Architectural types of domains

Records / structures

REnabledLogicalParts = (

ELongBoolean	**By_Material**	*Enable logical parts based on material*
ELongBoolean	**By_CrossSection**	*Enable logical parts based on cross-section*
ELongBoolean	**By_CrossSection_LineType**	*Enable logical parts based on cross-section and line type*
ELongBoolean	**By_CrossSection_ArchitecturalLineElementType**	*Enable logical parts based on cross-section and architectural line type*
ELongBoolean	**By_DomainType_Thickness**	*Enable logical parts based on domain type and thickness*
ELongBoolean	**By_ArchitecturalDomainElementType_Thickness**	*Enable logical parts based on architectural domain type and thickness*
ELongBoolean	**By_Stories**	*Enable logical parts based on stories*
ELongBoolean	**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] ELongBoolean **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, lpeNotFound or lpeStoreyIdOutOfBounds)

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] 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*
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, lpeNotFound or lpeCrossSectionIdOutOfBounds)

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] 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*
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, lpeNotFound,lpeCrossSectionIdOutOfBounds)

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, lpeNotFound,lpeCrossSectionIdOutOfBounds)

NOTE: If SelectParts = lbTrue the call Refresh function afterwards if not called between functions BeginUpdate and EndUpdate

---

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, lpeNotFound,lpeCrossSectionIdOutOfBounds or lpeInvalidLineType)

NOTE: If SelectParts = lbTrue the call Refresh function afterwards if not called between functions BeginUpdate and EndUpdate

---

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, lpeNotFound,lpeCrossSectionIdOutOfBounds or lpeInvalidLineType)

NOTE: If SelectParts = lbTrue the call Refresh function afterwards if not called between functions BeginUpdate and EndUpdate

---

long **GetBy_DomainElementType_Thickness** ([in] EDomainElementType **DomainElementType,** [in] Double **Thickness,** [in] ELongBoolean **SelectParts,** [in] ESelectMode **SelectMode,** [out] SAFEARRAY(RPartItem)*** PartItems**)

DomainElementType	*Structural 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 struct. type of domain and thickness. Returns number of part items 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_DomainElementType_Thickness_Storey** ([in] EDomainElementType **DomainElementType,** [in] Double **Thickness,** [in] long **StoreyId,**[in] ELongBoolean **SelectParts,** [in] ESelectMode **SelectMode,** [out] SAFEARRAY(RPartItem)*** PartItems**)

DomainElementType	*Structural 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 struct. type of domain, thickness and storey. Ret Returns PartUID if successful, otherwise returns an error code (errDatabaseNotReady, lpeNotFound or lpeStoreyIdOutOfBounds)

NOTE: If SelectParts = lbTrue the call Refresh function afterwards if not called between functions BeginUpdate and EndUpdate

---

long **GetBy_Material** ([in] long **MaterialId,** [in] ELongBoolean **SelectParts,** [in] ESelectMode **SelectMode,** [out] SAFEARRAY(RPartItem)*** PartItems**)

MaterialId	*index of the material*
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 material. Returns PartUID if successful, otherwise returns an error code (errDatabaseNotReady, lpeNotFound, lpeMaterialIdOutOfBounds)

NOTE: If SelectParts = lbTrue the call Refresh function afterwards if not called between functions BeginUpdate and EndUpdate

---

long **GetBy_StructuralGridLineUID** ([in] long **StructuralGridLineUID,** [in] ELongBoolean **SelectParts,** [in] ESelectMode **SelectMode,** [i/o] SAFEARRAY(RPartItem)*** PartItems**)

StructuralGridLineUID	*unique index of the structural grid's 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 StructrualGridUID. Returns PartUID if successful, otherwise returns an error code (errDatabaseNotReady, lpeNotFound, lpeMaterialIdOutOfBounds)

---

long **GetPartItemsByUID** ([in] long **PartUID,** [i/o] SAFEARRAY(RPartItem)*** PartItems**)

PartUID	*unique index of the part*
PartItems	*custom part items*

Get logical part items by PartUID. Returns PartUID if successful, otherwise returns an error code (errDatabaseNotReady)

---

long **GetEnabledLogicalParts** ([i/o] REnabledLogicalParts **EnabledLogicalParts**)

EnabledLogicalParts *Enabled logical parts*

Get enabled logical pars switch. Returns 1 if successful, otherwise returns an error code (errDatabaseNotReady)

---

long **SetEnabledLogicalParts** *([i/o] REnabledLogicalParts* **EnabledLogicalParts***)*

EnabledLogicalParts *Enabled logical parts*

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**] • *Is lbTrue if it is a logical part*

PartUID *Unique index of the logical part*

BSTR **Name** [long **PartUID**] • *Gets the name of the logical part*

PartUID *Unique index of the logical part*

BSTR **FullName** [long **PartUID**] • *Gets the full name of the logical part*

PartUID *Unique index of the logical part*

IAxisVMMaterials

Materials of the model.

Error codes

enum **EMaterialError** = {

meEmtpy_Name = –100001	*material does not have a name*	
meNegative_Nux = –100002	$\nu_x < 0$	
meNegative_Nuy = –100003	$\nu_y < 0$	
meNegative_Nuz = –100004	$\nu_z < 0$	
meGreaterThan05_Nux = –100005	$\nu_x > 0{,}5$	
meGreaterThan05_Nuy = –100006	$\nu_y > 0{,}5$	
meGreaterThan05_Nuz = –100007	$\nu_z > 0{,}5$	
meNegative_Alfax = –100008	$\alpha_x < 0$	
meNegative_Alfay = –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*
	$f_{yd} \leq 0$	*NEN steel*
meNonPositive_Fu = –100019	$f_u \leq 0$	*EC, I, DIN, SIA steel*
	$f_{yt} \leq 0$	*NEN steel*
meNonPositive_Fy40 = –100020	$f_y^* \leq 0$	*EC, I, DIN, SIA steel*
	$f_{yd}^* \leq 0$	*NEN steel*
meNonPositive_Fu40 = –100021	$f_u^* \leq 0$	*EC, I, DIN, SIA steel*
	$f_{yt}^* \leq 0$	*NEN steel*
meNonPositive_R = –100022	$R \leq 0$	*STAS steel*
meNonPositive_Rc = –100023	$R_c \leq 0$	*STAS steel*
meNonPositive_SigmabH = –100024	$\sigma_{bH} \leq 0$	*MSz concrete*
	$R_{bc} \leq 0$	*STAS concrete*
meNonPositive_SigmahH = –100025	$\sigma_{hH} \leq 0$	*MSz concrete*
	$R_{bi} \leq 0$	*STAS concrete*
meNonPositive_Fit = –100026	$\Phi \leq 0$	*STAS, NEN concrete*
	$\Phi_t \leq 0$	*EC, I, DIN, SIA concrete*
meNonPositive_Fck = –100027	$f_{ck} \leq 0$	*EC, I, DIN, SIA concrete*
	$f_{ck}' \leq 0$	*NEN concrete*
meNonPositive_GammaC = –100028	$\gamma_c \leq 0$	*EC, I, DIN, SIA concrete*
meNonPositive_Alfacc = –100029;	$\alpha_{cc} \leq 0$	*EC, I concrete*
	$\alpha \leq 0$	*DIN concrete*
meNonPositive_Fck_cube = –100030	$f_{ck,\,cube} \leq 0$	*DIN concrete*
meInvalid_MaterialType = –100031	*material type is invalid*	
meInvalid_NationalDesignCode = –100032	*national design code is invalid*	
meNameAlreadyExists = –100033	*material name already exists*	

meNonPositive_E005 = -100034,
meNonPositive_Gmean = -100035,
meNonPositive_fmk = -100036,
meNonPositive_ft0k = -100037,
meNonPositive_ft90k = -100038,
meNonPositive_fc0k = -100039,
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, that will appear in the Materials table. Do note that if there is a standardized name for the national code, that will be displayed instead*
MaterialDesignName	*name of the material code, that will appear in the Materials table. For example "EN 206".*
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/m²]*
Nux, Nuy, Nuz	ν *Poisson's ratio in local directions* $0 \le \nu \le 0{,}5$
Alfax, Alfay, Alfaz	α *thermal expansion coefficient in local directions [1/°C]*
Rho	ρ *density [kg/m³]*

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 **Alfax,**
[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 **Alfax,**
[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 **Alfax,**
[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/m²]*

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/m²]*
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/m²]*
Fck_cube $f_{ck,\ cube}$ *characteristic compressive cylinder strength of cube [kN/m²]*
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/m²]*
SigmahH σ_{hH} *resistance for tension [kN/m²]*
Fit Φ *creeping factor*

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

---

long **AddConcrete_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/m²]*
SigmahH R_{bi} *resistance for tension [kN/m²]*
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/m²]*
GammaC γ_c *safety factor of the concrete*
Fit Φ_t *creeping factor*

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

---

long **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**)

NationalDesignCodes *List of national design codes (except ndcOther) of whose materials will be shown in the dialog.*
If nil (null) then materials of current design code are shown.

MaterialTypes *List of material types which will be shown in the dialog*
If nil (null) then all types of materials are shown.

Opens a dialog for adding material. If successful, returns index of the material, otherwise an error (meErrorAdding).

---

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**)

Fy *f_{yd} yield stress [kN/m²]*
Fu *f_{yt} ultimate stress [kN/m²]*
Fy40 *f_{yd}* yield stress if thickness is between 40 and 100 mm [kN/m²]*
Fu40 *f_{yt}* ultimate stress if thickness is between 40 and 100 mm [kN/m²]*

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

---

long **AddSteel_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**)

For the beginning of the parameter list see **Add…**

Fy *f_y yield stress [kN/m²]*
Fu *f_u ultimate stress [kN/m²]*
Fy40 *f_y* yield stress if thickness is between 40 and 100 mm [kN/m²]*
Fu40 *f_u* ultimate stress if thickness is between 40 and 100 mm [kN/m²]*

Adds a steel according to Eurocode to the model.

---

long **AddSteel_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**)

See **AddSteel_EuroCode**

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

---

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 **Alfax,**
[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/m²]*
SigmapH	σ_{pH} *cylinder-jacket resistance [kN/m²]*
Ry	R_y *ultimate stress [kN/m²]*

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

---

long **AddSteel_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 **Alfax,**
[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 **Alfax,**
[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/m²]*
Rc	*Rc resistance [kN/m²]*

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 **Alfax,**
[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 **Alfax,**
[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**

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

---

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

Index *index of the material to delete (0 < Index ≤ Count)*

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

---

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

Name *name of the material*

Finds a metrial by its name. If successful, returns the index of the material, otherwise an error code (errDatabaseNotReady or errNotFound).

Properties

long **Count**
Number of materials in the model.

long **IndexOfUID** [long **UID**] *Get index of the material*

UID *unique index of the material*

AxisVMMaterial* **Item** [long **Index**]
Material interface by index. 1 ≤ Index ≤ Count.

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

Index *index of the material*

IAxisVMMaterial

Material interface in the model or catalog.

Enumerated types

enum **EMaterialType** = {

mtOther = 0x0,	*Other*
mtSteel = 0x1,	*Structural steel*
mtConcrete = 0x2,	*Concrete*
mtTimber = 0x3,	*Timber*
mtAluminium = 0x4,	*Aluminium*
mtBrick = 0x5 }	*Brick*

Material type.

enum **EMaterialColour**= {

mclOther = 44732672,	*Other*
mclOtherContour = 0,	*Other contour*
mclSteel = 39134390,	*Structural steel*
mclSteelContour = 39124992,	*Structural steel contour*
mclConcrete = 44742326,	*Concrete*
mclConcreteContour = 33554432,	*Concrete contour*
mclTimber = 33582518,	*Timber*
mclTimberContour = 33563720,	*Timber contour*
mclAluminium = 50313837,	*Aluminium*
mclAluminiumContour = 39143496,	*Aluminium contour*
mclBrick = 8031416,	*Brick*
mclBrickContour = 3424867 }	*Brick contour*

Material colour.

enum **ETimberType** = {

ttHardwood = 0x0,	*Hardwood*
ttGLULAM = 0x1,	*GLULAM*
ttLVL = 0x2,	*LVL* = Laminated Veneer Lumber
ttSoftWood = 0x3,	*SoftWood*
ttOther = 0x4 }	*Other*

Types of timber

Properties

Please note: All properties are read and write if not noted otherwise

double	**Alfax** • *α thermal expansion coefficient in local x direction [1/°C]*
double	**Alfay** • *α thermal expansion coefficient in local y direction [1/°C]*
double	**Alfaz** • *α thermal expansion coefficient in local z direction [1/°C]*
unsigned long	**ContourColour** • *outline(edge) colour used in rendered view (see IAxisVMMaterials)*
long	**ContourColour_vb** • Visual Basic compatible property of **ContourColour**
double	**Ex** • *Young's modulus of elasticity in local x direction [kN/m^2]*
double	**Ey** • *Young's modulus of elasticity in local y direction [kN/m^2]*
double	**Ez** • *Young's modulus of elasticity in local z direction [kN/m^2]*
unsigned long	**FillColour** • *fill colour used in rendered view (see IAxisVMMaterials) , any number or EmaterialColour*
unsigned long	**FillColour_vb** • Visual Basic compatible property of **FillColour**
BSTR	**MaterialDesignName** • *name of the material code*
EMaterialType	**MaterialType** • *type of the material*

BSTR **Name** • *material name*

ENationalDesignCode **NationalDesignCode** • *national design code*

BSTR **NationalDesignName** • *name of the national design code*

double **Nux** • ν *Poisson's ratio in local x direction* $0 \le \nu \le 0{,}5$

double **Nuy** • ν *Poisson's ratio in local y direction* $0 \le \nu \le 0{,}5$

double **Nuz** • ν *Poisson's ratio in local z direction* $0 \le \nu \le 0{,}5$

double **Rho** • ρ *density [kg/m³]*

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/m²]*

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

double **Fy** • *yield stress*
(EC, DIN, I, SIA: f_y*) (NEN:* f_{yd} *) [kN/m²]*

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

MSz

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

double **SigmaH** • *resistance (*σ_H *) [kN/m²]*

double **SigmapH** • *cylinder-jacket resistance (*σ_{pH} *) [kN/m²]*

STAS

double **R** • *ultimate stress (R) [kN/m²]*

double **Rc** • *yield stress (*R_c *) [kN/m²]*

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/m²]*

double **Fck_cube** • *characteristic compressive cylinder strength of cube*
(DIN: $f_{ck,cube}$*) [kN/m²]*

double **Fit** • *creeping factor (EC, I, DIN, SIA:* Φ_t*) (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/m²]*

double **SigmahH** • *resistance for tension (MSz:* σ_{hH} *, STAS:* R_{bi}*) [kN/m²]*

Design parameters of timber materials

double **E005** • *5% modulus of elasticity parallel to grain (x) [kN/m²]*

double **fmk** • *Characteristic bending strength [kN/m²]*

double **ft0k** • *Characteristic tensile strength parallel to grain [kN/m²]*

double **ft90k** • *Characteristic tensile strength perpendicular to grain [kN/m²]*

double **fc0k** • *Characteristic compression strength parallel to grain [kN/m²]*

double **fc90kz** • *Characteristic compression strength perpendicular to grain (z) [kN/m²]*

double **fvkz** • *Characteristic shear strength (z) [kN/m²]*

double **fvky** • *Characteristic shear strength (y) [kN/m²]*

double **fc90ky** • *Characteristic compression strength perpendicular to grain (y) [kN/m^2]*

double **GammaM** • *Partial factor of the material*

double **GMean** • *Mean shear modulus [kN/m^2]*

double **s** • *Size effect exponent (for LVL materials)*

ETimberType **TimberType** • *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 IAxisVMModel.

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 trigered.

The input of the math composer is a single string describing the calculations.
The string can contain control keywords, text and mathematical formulas. CRLF characters (with code 13 and 10) are ignored. New paragraphs can be started with the `\par` keyword.
String processing is performed according to two control keywords: `\text` and `\math`.

Text mode

Text entered after the `\text` keyword will appear as it is, though certain formatting commands help creating headlines or other styles. These keywords must be followed by a space, e.g. `\fontArial \size8 \b …`

`\n Normal`	Normal
`\i Italic`	*Italic*
`\b Bold`	**Bold**
`\t Bold italic`	***Bold italic***
`\fontLucida.Console`	Font name without a space If font name contains spaces, spaces must be replaced by '.' characters.
`\size18`	Font size
`\colorRed`	Font color by name Accepted names are `Black, Maroon, Green, Olive, Navy, Purple, Teal, Gray, Silver, Red, Lime, Yellow, Blue, Fuchsia, Aqua, White`
`\color$804040`	Color in hexadecimal, byte order: BlueGreenRed

Math mode

Text after the `\math` keyword is interpreted as a mathematical formula.
An example:

```
\text Equations for the projectile motion:\par
\math x(t)=x_0+v_x t \gap \text  and  \math y(t)=y_0+v_y t-{1 \over 2}gt^2
\par t=(x-x_0)\over {v_x}
```

Equations for the projectile motion:

$$x(t) = x_0 + v_x t \text{ and } y(t) = y_0 + v_y t - \frac{1}{2} g t^2$$

$$t = \frac{(x - x_0)}{v_x}$$

`{}`	logical braces (not displayed)
`_` (underscore)	lower index
`^`	upper index
`+ - * / < = >`	math operations
`() []`	braces

`\alpha \beta \khi \delta \epsilon`
`\phi \gamma \eta \iota \varphi`
`\kappa \lambda \mu \nu \omicron`
`\pi \theta \rho \sigma \tau`
`\upsilon \varpi \omega \ksi \psi \zeta`
Greek capitals: `\Alpha \Beta ...`
	$\alpha\,\beta\,\chi\,\delta\,\varepsilon$
	$\phi\,\gamma\,\varepsilon\,\iota\,\varphi$
	$\kappa\,\lambda\,\mu\,\nu\,o$
	$\pi\,\theta\,\rho\,\sigma\,\tau$
	$\upsilon\,\varpi\,\omega\,\xi\,\psi\,\zeta$

`\sum \deg \diameter`	$\sum {}^{o}\,\varnothing$
`\perp \le \leftarrow \rightarrow \pm \ge \neq \equiv \approx \times`	$\perp \le \leftarrow \rightarrow \pm \ge \neq \equiv \approx \times$
`\over`	stands between the numerator and the denominator of a fraction, if the denominator is more than one character it has to be entered within { } braces
`\sqrt`	square root
`\abs`	absolute value
`\ast`	* (asterisk)
`\gap`	space-wide gap in formulas
`\sin \cos \tg \log ...`	mathematical functions must be entered this way

Tables

Commands between `\begintable` and `\endtable` commands are interpreted as tables

```
\begintable 6x2 \header 0
\cellmt 0,0 \hal center "\b Text \par in two lines"
\cellmt 0,1 \hal center "\b Formula"
\cellmt 1,0,2,0 "\i Values"
\cellmt 1,1  \val center "\math \ksi=\sqrt 3 \over 2"'+
\cellmt 2,1 \hal right \val center "\math \omega=\sqrt 5
\over 3"
\cellpic "c:\Axis\bitmap\p1p2.jpg",120,0,3,0
\cellpic "c:\Axis\bitmap\p1p2.jpg",200,0,3,1
\cellmt 4,0 \hal right "Text"
\cellmt 4,1 "\t Cell text"
\cellmt 5,0 "Text"
\cellpic "c:\Axis\bitmap\p1p2.jpg",100,0,5,1
\endtable
```

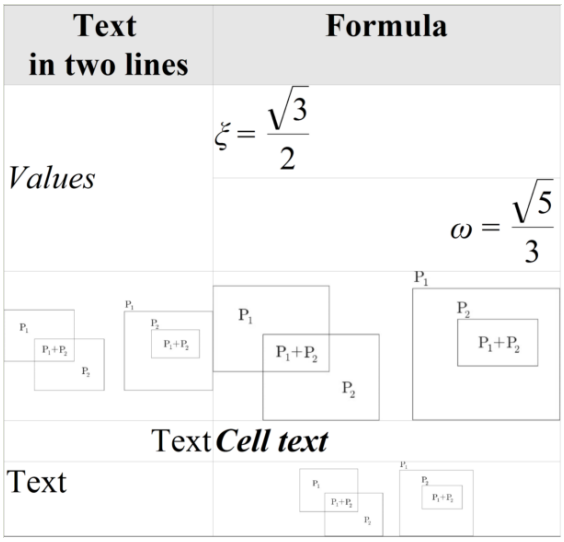


Table cells can contain one or more ‘grid cells’. `\cellmt` and `\cellpic` commands have parameters to specify the cell position. If a table cell covers only one ‘grid cell’ it is enough to specify *row,height.* If the cell spans over multiple ‘grid cells’ enter the top left and right bottom grid cell coordinates, like `\cellmt 1,0,2,0 "\i Values"`

`\begintable` *rows*x*columns*	begins a table and specified the number of grid cell rows x columns
optional:	
`\gridcolor` *color*	The hexadecimal value in *color* (e.g. A0B8B8) sets the color of the table grid (optional)
`\gridoff`	Hides the table grid (optional)
`\header` *n*	Index of the the last header row. If omitted no preformatted header will appear. Header has gray background and displays text in bold by default. (optional)
`\headercolor` *color*	The hexadecimal value in *color* (e.g. A0B8B8) sets the background color of the table header (optional)

\cellmt *position*	Cell displaying text and formulas. If a table cell covers only one ‘grid cell’ it is enough to specify *row,height.* If the cell spans over multiple ‘grid cells’ enter the top left and right bottom grid cell coordinates, like \cellmt 1,0,2,0 "\i Values"
\background *color*	The hexadecimal value in *color* (e.g. A0B8B8) sets the background color of the cell (optional)
\border *values*	values are four numbers encoding the left, top, right, bottom border. E.g. \border 1100,2030,1100,1100 makes all borders 100% black, with thickness 1, except the left one being 30% gray with thickness 2
\hal *align*	Horizontal aligment of the cell content. Possible values of *align* are: *left, center, right.*
\val *align*	Vertical aligment of the cell content. Possible values of *align* are: *top, center, bottom*
\text *"text"*	Text of the cell between " characters, can contain any number of \text and \math sections.
\cellpic *“filename”, width, height, position*	Cell displaying an image. The name of the image file must be entered between " characters. Width and height of the image must be specified in millimeters. One of the *height* and *width* fields can be set to zero. That size will be calculated according to the image aspect ratio. If a table cell covers only one ‘grid cell’ it is enough to specify position as *row,height.* If the cell spans over multiple ‘grid cells’ enter the top left and right bottom grid cell coordinates, like \cellpic c:\Axis\bitmap\p1p2.jpg",120,0,3,0,3,1
\endtable	Ends the table

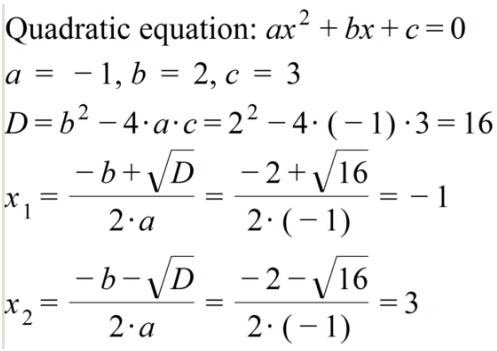
Variables and formula substitution

By defining variables it is possible to create substituted formulas.

\var a = 1;	Variable with a simple assignment
\var D = #b^2-4*#a*#c = 16;	Variable with a formula and an assignment. Other variables must be referred in the formula using # as a prefix. Referred variables will be automatically substituted The result must be calculated and entered after the formula separately. It is recommended to use the sign of multiplication (*) to make substituted formulas readable.
\eval a;	Displays the value of the variable. If the variable has a formula a substitution is also displayed.

Example:

```
\text Quadratic equation: \math ax^2+bx+c=0 \par
\var a =-1;
\var b = 2;
\var c = 3;
\var D = #b^2-4*#a*#c = 16;
\var x_1 = -#b+\sqrt #D\over{2*#a} = -1;
\var x_2 = -#b-\sqrt #D\over{2*#a} = 3;
\eval a;,\gap \eval b;,\gap \eval c; \par
\eval D; \par
\eval x_1; \par
\eval x_2; \par
```



Errors in entering variables do not stop processing. Errors are displayed as ?? characters according to the following rules.

Referring to an undeclared variable
The value of the undeclared variable will be displayed as ??
If the undeclared variable is referred in a formula it will be displayed as ?? and the name of the undeclared variable

input	*output*

`\var x = -1;`
`\var b = 2;`
`\var c = 3;`
`\var D = #b^2-4*#a*#c = 16;`
`\eval a; \par\eval D;`
	$a = ??$
	$D = b^2 - 4 \cdot a \cdot c = 2^2 - 4 \cdot ??a \cdot 3 = 16$

Invalid assignment
The erroneously assigned (non-numerical) value will be displayed with ?? characters:

input	*output*

`\var a = x;`
`\var b = 2;`
`\var c = 3;`
`\var D = #b^2-4*#a*#c = 16;`
`\eval a; \par\eval D;`
	$a = x??$
	$D = b^2 - 4 \cdot a \cdot c = 2^2 - 4 \cdot x?? \cdot 3 = 16$

Error codes

enum **EMathTextError** = {

mteMathTextUIDandAPI_NameDontMatch = -100001,	*MathText was added with different API_Name*
mteAPI_NameNotFound = -100002,	*No math texts found with this API_Name*
mteMathTextUIDnotValid = -100003	*Math texts with this unique index is not found*

}

Functions

long **AddToReport** ([in] long **MathTextUID**, [in] BSTR **API_Name**)

MathTextUID *Unique index of math text*

API_Name *Name of the COM client, used for identification of the client*

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).

---

long **Delete** ([in] long **MathTextUID**, [in] BSTR **API_Name**)

MathTextUID *Unique index of math text*

API_Name *Name of the COM client, used for identification of the client*

Delete the math text. If successful, returns unique index of math text, otherwise an error code (mteMathTextUIDnotValid, mteMathTextUIDandAPI_NameDontMatch, errDatabaseNotReady).

---

long **GetUIDs** ([in] BSTR **API_Name,** [out] SAFEARRAY(long) **UIDs**)

API_Name *Name of the client which is saved to the model file.*

UIDs *Unique indexes of math text which belong to* API_Name

Get math text UIDs by API_Name . If successful, returns unique index of math text, otherwise an error code (mteAPI_NameNotFound, errDatabaseNotReady).

---

long **GetMathText** ([in] long **MathTextUID**, [in] BSTR **API_Name,** [out] BSTR **MathTextTitle,** [out] BSTR **MathText,** [out] ELongBoolean **EnableIfNotPresent**)

MathTextUID	*Unique index of math text*
API_Name	*Name of the COM client, used for identification of the client*
MathTextTitle	*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*
MathText	*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.*
EnableIfNotPresent	*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 trigered for this MathText. Set to lbFalse if you want to update with events.*

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).

---

long **New** ([in] BSTR **API_Name,** [in] BSTR **MathTextTitle,**
[in] BSTR **MathText,** [in] ELongBoolean **EnableIfNotPresent**)

MathTextUID	*Unique index of math text*
API_Name	*Name of the COM client, used for identification of the client*
MathTextTitle	*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*
MathText	*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.*
EnableIfNotPresent	*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 trigered for this MathText. Set to lbFalse if you want to update with events.*

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).

---

long **SetMathText** ([in] long **MathTextUID**, [in] BSTR **API_Name,** [in] BSTR **MathTextTitle,**
[in] BSTR **MathText,** [in] ELongBoolean **EnableIfNotPresent**)

MathTextUID	*Unique index of math text*
API_Name	*Name of the COM client, used for identification of the client*
MathTextTitle	*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*
MathText	*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.*
EnableIfNotPresent	*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 trigered for this MathText. Set to lbFalse if you want to update with events.*

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).

---

long **ShowInWindow** ([in] long **MathTextUID**, [in] BSTR **API_Name,** [in] BSTR **Caption,**
[in] RWindowPosition **Position,** [in] ELongBoolean **Substitution**)

MathTextUID	*Unique index of math text*
API_Name	*Name of the COM client, used for identification of the client*
Caption	*Caption of the form showing the math text*
Position	*Size and the position of the form showing the math text*
Substitution	*If lbTrue then the variables in math text are also shown.*

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).

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 transfered, explained here.

Error codes

enum **EMembersError** = {

mbeEmptyLineList = –100001,	*line list is empty*
mbePropertyNotValidForThisLineType = –100002,	*invalid property for the given line type*
mbeNotBeam = –100003,	*the element is not a beam*
mbeNotRib = –100004	*the element is not a rib*
mbeIllegalServiceClassValue = –100005	*service class value of the timber is incorrect*
mbeDomainIndexOutOfBounds = –100006	*provided DomainID was ioncorrect using functions: IAxisVMLine.DefineAsRibWithAutoExcentricity and IAxisVMLine.DefineAsTimberRibWithAutoExcentricity*
mbeStoreyIdOutOfBounds = –100007	*storey index invalid*
mbeMustBeBeamOrRibOrTuss = –100008	*member must be a beam, rib or truss*
mbeInvalidMemberType = –100009	*invalid member type*
mbeReinforcementParametersNotExsist = –100010	*reinforcement parameters not exsist*
mbeInvalidColumnRebarsId = –100011	*invalid column rebar index*
mbeInvalidConcreteMaterialId = –100012	*invalid concrete material index*
mbeInvalidRebarSteelGradeId = –100013	*invalid rebar steel grade index*
mbeNotTruss = –100014	*the element is not a truss*
mbeInvalidRelease = –100015	*other release error*
mbeInvalidFunctionIDofRelease = –100016	*invalid function index related to the DOF*
mbeReleaseInitAndLimitMustBe0 = –100017	*Initial rotational stiffness and moment resistance of the release must be 0*
mbeFunctionIdMustBe0= –100018	*FunctionId of the release must be 0*
mbeMembersNotContinuous= –100019	*Members do not compose a continuous member*
mbeStartEndCrossSectionTypeIncompatible= –100020	
mbeInvalidRCCheckingParameters = -100021	*at least one of the RC checking parameters is invalid*
mbeRCShrinkageEpsMustBePositive = -100022	*shrinkage strain must be positive*
mbeStirrupParametersAreInvalid = -100023	*stirrup parameters are invalid*
mbeShearCrackAngleIsInvalid = -100024	*defined shear crack angle is too low/high*
mbeInvalidSteelMaterialId = -100025	*invalid steel grade index*
mbeInvalidStiffnessReduction = -100026	*stiffness reduction should be > 0.00 and <= 1.00*
mbeStiffnessReductionNotAllowed = -100027	*design code doesn't allow for stiffness reduction*
mbeInvalidStiffnessReductionMat = -100028	*member's material doesn't allow for stiffness reduction*
mbeRibEccTypeUsedOnBeam = -100029	*that eccentricity is meant only for ribs*
mbeReleaseFunctionIndexError = -100030	*invalid function index for a release*
mbeReleaseInvalidType = -100031	*invalid type for a release*
mbeInvalidRefZ = -100032	*invalid reference*
mbeInvalidLineType = -100033	*LineType field cannot have this value*
mbeReleaseInvalidMaterial = -100034	*invalid material for a release (for now plastic hinges requires steel elements)*
mbeReleaseInvalidComponent = -100035	*release component cannot be of the specified type*
mbeSpringTypeIncompatible = -100036	*spring type isn't in accordance with its role (for example a displacement type spring for a rotational component)*
mbeSpringIndexOutOfBounds = -100037	*spring index is out of bounds of the valid spring indexes (1.. AxisVMSpringParams.Count)*
mbeEccReleaseAutoPosNotAllowed = -100038}	*rptAuto value for PosType is allowed only for eccentricity type leet_Ref)*

Functions

long **Add** ([in] SAFEARRAY(long) **LineIds**)

LineIds *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 **AssembleMembers** ([in] SAFEARRAY(long) **MemberIds)**

MemberIds *list of member indexes to query. Each index must satisfy: (1 ≤ Index ≤ AxisVMMembers.Count)*

Tries to assemble the MemberIds into the largest possible members in accordance with the internal rules for that (i.e. what the Edit->"Assemble structural members" menu point does). As such, it can create one or more new members. If successful, returns the number of created members, otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBounds).

---

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**)

Warning! *This function has become obsolete, was superseded by BulkGetMembers_V161*

MemberIds *list of member indexes to query. Each index must satisfy: (1 ≤ Index ≤ 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 **BulkGetMembers_V161** ([in] SAFEARRAY(long) **MemberIds**, [out] SAFEARRAY (RLineAttr_V161) ***** **MemberAttrs**)

MemberIds *list of member indexes to query. Each index must satisfy: (1 ≤ Index ≤ 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**)

Warning! *This function has become obsolete, was superseded by BulkSetMembers_V161*

MemberIds *list of member indexes to modify. Each index must satisfy: (1 ≤ Index ≤ 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, mbeIllegalServiceClassValue, mbePropertyNotValidForThisLineType) .

---

long **BulkSetMembers_V161** ([in] SAFEARRAY(long) **MemberIds**, [in] SAFEARRAY (RLineAttr_V161) **MemberAttrs**)

MemberIds *list of member indexes to modify. Each index must satisfy: (1 ≤ Index ≤ AxisVMMembers.Count)*

MemberAttrs *list of member attribute records.*

Not all fields of the RLineAttr_V161 are active at the same time.The main factor influencing what is active is LineType, which allways has to be specified.Only the following subset of LineType is valid : ltSimpleLine, ltTruss, ltBeam, ltRib, ltSpring, ltGap.Trying to set for example a LineType of ltLLLink will result in an error.

LineType= ltSimpleLine : no other field is required

LineType=ltTruss : LocalXOrientation, Reference, MaterialIndex, StartCrossSectionIndex=EndCrossSectionIndex (they must be equal), TrussType, Resistance, MaterialColor, ContourColor. If MaterialIndex refers to timber, then also ServiceClass and kdef.

LineType=ltBeam : LocalXOrientation, Reference, MaterialIndex, StartCrossSectionIndex, EndCrossSectionIndex, StartRelease, EndRelease, StartEccentricityType, EndEccentricityType, StartEccentricity, EndEccentricity,

StartAlignementPoint, EndAlignementPoint, EccGroupIndex, StartRefLine, EndRefLine, RefStartAlignementPoint, RefEndAlignementPoint, StartEccRelease, EndEccRelease, Beam7DOF, MaterialColor, ContourColor. If MaterialIndex refers to timber, then also ServiceClass and kdef.

LineType=ltRib : LocalXOrientation, Reference, MaterialIndex, StartCrossSectionIndex, EndCrossSectionIndex, StartRelease, EndRelease, RibAutoEccentricityType, StartEccentricity, EndEccentricity, kx, StartEccentricityType, EndEccentricityType, StartEccentricity, EndEccentricity, StartAlignementPoint, EndAlignementPoint, EccGroupIndex, StartRefLine, EndRefLine, RefStartAlignementPoint, RefEndAlignementPoint, StartEccRelease, EndEccRelease, MaterialColor, ContourColor. If the rib is attached to a domain then Domain1 and situationally Domain2. If MaterialIndex refers to timber, then also ServiceClass and kdef. If StartEccentricityType is leet_RibDomain, then EndEccentricityType must be leet_RibDomain too.

LineType= ltSpring : SpringType, then depending on SpringType either SpringCharacteristics or [SeismicIsolatorIndex and IsolatorD2].

LineType= ltGap : GapType, ActiveStiffness, InactiveStiffness, InitialOpening, MinPenetration, MaxPenetration, AdjustmentRatio.

If Reference is active, it must satisfy: (1 ≤ Reference ≤ AxisVMReferences.Count)
If MaterialIndex is active, it must satisfy: (1 ≤ MaterialIndex ≤ AxisVMMaterials.Count)
If StartCrossSectionIndex is active, it must satisfy: (1 ≤ StartCrossSectionIndex ≤ AxisVMCrossSections.Count)
If EndCrossSectionIndex is active, it must satisfy: (1 ≤ EndCrossSectionIndex ≤ AxisVMCrossSections.Count)
If Domain1 is active, it must satisfy: (0 ≤ Domain1 ≤ AxisVMDomains.Count)
If Domain2 is active, it must satisfy: (0 ≤ Domain2 ≤ AxisVMDomains.Count)
If ServiceClass is active, it must be 1, 2 or 3
If StartRefLine, is active, it must satisfy: (1 ≤ StartRefLine ≤ AxisVMLines.Count)
If EndRefLine, is active, it must satisfy: (1 ≤ EndRefLine ≤ AxisVMLines.Count)

Modifies the properties of several already existing members in one step. It is also used to modify the otherwise inacessible properties of a member (like the eccentricities or the Beam7DOF field of a beam). If successful, returns number of modified members, otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBounds, errInternalException, errOutOfMemory, mbeStartEndCrossSectionTypeIncompatible, mbeIllegalServiceClassValue, mbePropertyNotValidForThisLineType, mbeInvalidLineType, mbeRibEccTypeUsedOnBeam, mbeReleaseFunctionIndexError, mbeReleaseInvalidType, mbeInvalidRefZ, mbeReleaseInvalidMaterial, mbeReleaseInvalidComponent).

---

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

Index *member index (0 < Index ≤ 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 **ClearEccentricities** ([in] SAFEARRAY(long) **MemberIds**)

MemberIds *list of member indexes to modify. Each index must satisfy: (1 ≤ Index ≤ AxisVMMembers.Count)*

Clears the eccentricities of the listed members. If successful, returns zero, otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBounds).

---

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

Index *member index (0 < Index ≤ Count)*

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).

---

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

Index *member index (0 < Index ≤ Count)*

If successful, returns member index, otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBounds)

---

long **DeleteMeshFromSelectedItems**

If successful, returns number of selected members, otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBounds)

---

long **DeleteNameOfAllBeams**

Deletes previously added default name for all beams. If successful returns 1, otherwise returns an error code (errDatabaseNotReady).

---

long **DeleteNameOfAllRibs**

Deletes previously added default name for all ribs. If successful returns 1, otherwise returns an error code (errDatabaseNotReady).

---

long **DeleteNameOfAllTrussses**

Deletes previously added default name for all trusses. If successful returns 1, otherwise returns an error code (errDatabaseNotReady).

---

long **DeleteSelected**

Deletes selected structural members. Their constituent line elements are not deleted. If successful, returns zero, otherwise returns an error code (errDatabaseNotReady).

---

long **GenerateMeshWithParamsOnSelectedItems** ([i/o] RMemberMeshParameters* **MeshParameters**)

MeshParameters *Mesh parameters*

If successful, returns number of selected members, otherwise returns an error code (errDatabaseNotReady)

---

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*

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).

---

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

ItemIds *list of selected members*

If successful, returns the number of selected members otherwise returns an error code

---

long **GetSelectedColumnIds** ([out] SAFEARRAY (long) * **MemberIds)**

MemberIds *list of selected vertical members*

If successful, returns the number of selected vertical members, otherwise returns an error code(errDatabaseNotReady).

---

long **GetSelectedBeamIds** ([out] SAFEARRAY (long) * **MemberIds**)

MemberIds *list of selected horizontal members*

If successful, returns the number of selected horizontal members otherwise returns an error code(errDatabaseNotReady).

---

long **GetSelectedOtherIds** ([out] SAFEARRAY (long) ***** **MemberIds**)

MemberIds *list of selected neither vertical or horizontal members*

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

---

long **IndexOf** ([in] SAFEARRAY(long) **LineIds**)

LineIds *Consecutive line indexes defining the structural member. Must contain all line indexes of the member otherwise function returns 0.*

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 [LineIds array is empty]).

---

long **IndexOf _vb** (Visual Basic compatible function of **IndexOf**)

---

long **RenameSelectedBeams** ([in] long **NewBase,** [in] BSTR **FormatStr**)

NewBase *Start number for renaming, must be a positive number*

FormatStr *Prefix string of the new name + "_" eg "Beam_"*

Rename selected members with membertype.of beam.
Example: NewBase=100 and FormatStr= 'new_' then names are: 'new100', 'new101',…etc.
If successful returns 1, otherwise returns an error code (errDatabaseNotReady).

---

long **RenameSelectedRibs** ([in] long **NewBase,** [in] BSTR **FormatStr**)

NewBase *Start number for renaming, must be a positive number*

FormatStr *Prefix string of the new name + "_" eg "Rib_"*

Rename selected members with membertype.of rib
Example: NewBase=100 and FormatStr= 'new_' then names are: 'new100', 'new101',…etc.
If successful returns 1, otherwise returns an error code (errDatabaseNotReady).

---

long **RenameSelectedTrusses** ([in] long **NewBase,** [in] BSTR **FormatStr**)

NewBase *Start number for renaming, must be a positive number*

FormatStr *Prefix string of the new name + "_" eg "Truss_"*

Rename selected members with membertype.of truss
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 *selection state*

If Select is True, selects all members.
If Select is False, deselects all members.
If successful, returns the number of selected members, 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 members.
If Select is False, deselects all vertical members.
If successful, returns the number of selected vertical members, 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 members
If Select is False, deselects all horizontal members
If successful, returns the number of selected horizontal members, 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 vertical or horizontal members).
If Select is False, deselects all neither vertical or horizontal members .

If successful, returns the number of selected neither vertical or horizontal members, 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 members at storey.
If Select is False, deselects all vertical members at storey.
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).

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*

If Select is True, selects all horizontal members.
If Select is False, deselects all horizontal members.

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).

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*

If Select is True, selects all neither vertical or horizontal members at storey.
If Select is False, deselects all neither vertical or horizontal members at storey.
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).

NOTE: Call Refresh function afterwards if not called between functions BeginUpdate and EndUpdate

---

long **SelectArray** ([in] SAFEARRAY(long) **MemberIds,** [in] ELongBoolean **Select,** [in] ELongBoolean **ClearSelFirst**)

MemberIds *list of member indexes to query. Each index must satisfy: (1 ≤ Index ≤ AxisVMMembers.Count)*

Select *selection state*

ClearSelFirst *if lbTrue, clears the pre-existing selection, which means that only MemberIds will be selected after a call with Select = lbTrue. If lbFalse, the pre-existing state of the selection will not be removed*

Alters the selection state of the members in the MemberIds list to Select. If successful, returns the global number of selected members, otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBounds).

NOTE: Call Refresh function afterwards if not called between functions BeginUpdate and EndUpdate

---

Properties

AxisVMAttachments*	**Attachments** *Get the attachments interface*
AxisVMAttributes*	**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*
ELongBoolean	**LocalX_is_ij** [long **Index**] *Is lbTrue if local x axis of the line is in i-j direction*
	Index *index of the member*
BSTR	**Name** [long **Index**] *Get name of the member*
	Index *index of the member*
ELongBoolean	**Selected** [long **Index**] • *Get or set the selection status of a structural member*
	Index *index of the member*
	NOTE: Call Refresh function afterwards if not called between functions BeginUpdate and EndUpdate
long	**SelCount** *Get number of selected structural members in the model*
double	**StiffnessReduction**[long **Index,** ELineStiffnessReduction **Component**] •
	Get or set stiffness reduction for the given component. Default value is 1. Used only for atLinearStatic and atLinearVibration analysis types. (only valid for Eurocode, SIA, NTS standards)
double	**StiffnessReduction_A**[long **Index**]
	Warning! *This function was extended by StiffnessReduction. 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 Eurocode, SIA, NTS standards)*
double	**StiffnessReduction_I**[long **Index**]
	Warning! *This function was extended by StiffnessReduction. 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 Eurocode, SIA, NTS standards)*
long	**UID** [long **Index**] *Get unique index of the member which remains the same while exists in the model*
	Index *index of the member*

IAxisVMMember

AxisVM structural member interface.

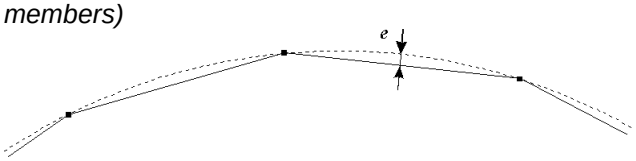
NOTE:

When rib is defined please note that the point of application of defined loads are transfered, 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.

enum **EMemberExcType** = *{*

mexcNone = 0,	*no eccentricity*
mexcDomainRib = 1,	*maximum element size*
mmtSegments = 0x2,	*number of segments*
mmtAngle = 0x3 }	*angle of each segment (only for curved members)*

Eccentricity type of the member.

Records / structures

RMemberMeshParameters = (

EMemberMeshType	**MeshType**	*mesh type*
double	**MeshParam**	*mesh parameter*

)

Functions

long **ClearEccentricity**

Clears the member's eccentricity. If successful, returns the member index, otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBounds).

---

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 < MaterialIndex ≤ 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, if you want to set the eccentricity use IAxisVMMembers.BulkSetMembers_V161 instead)***
EndEccentricity	*eccentricity at the end node **(not used, if you want to set the eccentricity use IAxisVMMembers.BulkSetMembers_V161 instead)***

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**)

MaterialIndex	*index of the material* *(0 < MaterialIndex ≤ 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, if you want to set the eccentricity use IAxisVMMembers.BulkSetMembers_V161 instead)***
EndEccentricity	*eccentricity at the end node **(not used, if you want to set the eccentricity use IAxisVMMembers.BulkSetMembers_V161 instead)***

Defines a structural member as a rib. For ribs with a constant cross-section StartCrossSectionIndex = EndCrossSectionIndex.
If successful, returns the member index according to IAxisVMMembers, otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBounds).

---

long **DefineAsRibWithAutoExcentricity** ([in] long **MaterialIndex,** [in] long **StartCrossSectionIndex,**
[in] long **EndCrossSectionIndex,** [in] EAutoExcentricityType **AutoExcentricityType,**
[in] double **kx,** [in] long **Domain1,** [in] long **Domain2**)

MaterialIndex	*Material Index*
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)*
AutoExcentricityType	*Type of the automatic eccentricity*
kx	*Friction resistance between rib and surface*
Domain1	*Domain index (filled if connected to domain)*
Domain2	*Domain index (filled if rib is between two domains)*

Defines a structural member as a rib, where eccentricity is calculated using domian 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).

---

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 (according to local x direction)*

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

If successful, returns member index, otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBounds, mbeIllegalServiceClassValue).

---

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 (according to local x direction)*

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

Defines a structural member as a timber rib. If successful, returns member index, otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBounds, lneMaterialIndexOutOfBounds, lneCrossSectionIndexOutOfBounds, mbeIllegalServiceClassValue).

---

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 *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)*

AutoExcentricityType *Type of automatic eccentricity*

kx *Friction resistance between rib and surface*

Domain1 *Domain index (filled if connected to domain)*

Domain2 *Domain index (filled if rib is between two domains)*

Defines a structural member as a timber rib, where eccentricity is calculated using domian thicknesses (Domain1, Domain2). If successful, returns member index, otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBounds, mbeDomainIndexOutOfBounds, mbeIllegalServiceClassValue).

---

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 lnlTensionOnly and lnlCompressionOnly types of truss.*

Defines a member as a timber truss element.
If successful, returns the member index, otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBounds, mbeIllegalServiceClassValue).

---

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

MaterialIndex	*index of the material* *(0 < MaterialIndex ≤ AxisVMMaterials.Count)*
CrossSectionIndex	*index of the cross-section* *(0 < CrossSectionIndex ≤ AxisVMCrossSections.Count)*
TrussType	*nonlinear behaviour*
Resistance	*Axial resistance (absolute value) only for lnlTensionOnly and lnlCompressionOnly types of truss*

Defines a member as a truss element.
If successful, returns the member index, otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBounds,).

---

long **DeleteColumnReinforcementParameters**

If successful, returns member index, otherwise returns an error code (errDatabaseNotReady).

---

long **DeleteMesh**

If successful, returns member index, otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBounds)

---

long **DeleteLineElement**

Delete all assigned properties (cross-sections, materials, etc) from the member. If successful, returns member index, otherwise returns an error code (errDatabaseNotReady).

---

long **GenerateMeshWithParams** ([i/o] RMemberMeshParameters* **MeshParameters**)

MeshParameters	*Mesh parameters*

If successful, returns number of existing parts before new meshing, otherwise returns an error code (errDatabaseNotReady)

---

long **GetBeamData** ([out] long **MaterialIndex,** [out] long **StartCrossSectionIndex,**
[out] 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 the local x direction)* *(0 < StartCrossSectionIndex ≤ AxisVMCrossSections.Count)*
EndCrossSectionIndex	*index of the end cross-section (according to the local x direction)* *(0 < EndCrossSectionIndex ≤ AxisVMCrossSections.Count)*
StartEccentricity	*eccentricity at the start node* ***(not used, if you want to get the eccentricity use IAxisVMMembers.BulkGetMembers_V161 instead)***
EndEccentricity	*eccentricity at the end node* ***(not used, if you want to get the eccentricity use IAxisVMMembers.BulkGetMembers_V161 instead)***

Get properties of a beam structural member. If successful, returns the member index according to IAxisVMMembers, otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBounds, mbeNotBeam).

---

long **GetColumnReinforcementParameters** ([i/o] RColumnReinforcementParameters **ColumnReinforcementParameters**)

ColumnReinforcementParameters	*column reinforcement parameters*

If successful, returns the member index, otherwise returns an error code (errDatabaseNotReady, mbeReinforcementParametersNotExsist, mbeInvalidMemberType, mbeEmptyLineList).

---

long **GetEndReleases** ([out] RReleases **EndReleases**)

EndReleases	*end releases*

Get end releases of the structural member. If successful, returns the member index according to IAxisVMMembers, otherwise returns an error code (errDatabaseNotReady).

---

long **GetGeomData** ([i/o] RLineGeomData **Value**)

Get the geometry data for the arc (only if GeomType = lgtCircleArc).
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**)

AnalysisType	*material Index*
MemberSectionId	*section indexof the member* *0 < MemberSectionId ≤ AxisVMMember. SectionsCount*
LineID	*line index*
LineSectionID	*index of the section on the line*

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).

---

long **GetLines** ([out] SAFEARRAY(long) * **LineIds**)

LineIds	*line indexes*

Get the successive line indexes of the constituents.
If successful, returns the array length, otherwise returns an error code (errDatabaseNotReady).

---

long **GetLinesLocX** ([out] SAFEARRAY(long) * **LineIds**)

LineIds	*line indexes*

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).

---

long **GetMeshParameters** ([i/o] RMemberMeshParameters* **MeshParameters**)

MeshParameters	*Mesh parameters*

If successful, returns member index.

---

long **GetRibData** ([out] long **MaterialIndex,** [out] long **StartCrossSectionIndex,** [out] 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 the local x direction)* *(0 < StartCrossSectionIndex ≤ AxisVMCrossSections.Count)*
EndCrossSectionIndex	*index of the end cross-section (according to the local x direction)* *(0 < EndCrossSectionIndex ≤ AxisVMCrossSections.Count)*
StartEccentricity	*eccentricity at the beginning* ***(not used, if you want to get the eccentricity use IAxisVMMembers.BulkGetMembers_V161 instead)***
EndEccentricity	*eccentricity at the end* ***(not used, if you want to get the eccentricity use IAxisVMMembers.BulkGetMembers_V161 instead)***

Get properties of a rib structural member. If successful, returns the member index according to IAxisVMMembers, otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBounds, mbeNotRib).

---

long **GetStartReleases** ([out] RReleases **StartReleases**)

StartReleases	*start releases*

Get start releases of the structural member. If successful, returns the member index according to IAxisVMMembers, otherwise returns an error code (errDatabaseNotReady).

---

long **GetTrussData** ([out] long **MaterialIndex,** [out] long **CrossSectionIndex,**
[out] ELineNonlinearity **TrussType,** [out] double **Resistance**)

MaterialIndex	*index of the material* *(0 < MaterialIndex ≤ AxisVMMaterials.Count)*
CrossSectionIndex	*index of the cross-section* *(0 < CrossSectionIndex ≤ AxisVMCrossSections.Count)*
TrussType	*nonlinear behaviour of the truss*
Resistance	*if nonzero, resistance of the truss* *(only if Truss Type <> lnlTensionAndCompression)*

Get properties of a truss. 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	*Material Index*
ServiceClass	*Service class of timber (valid values are 1, 2, 3)*
kdef	*Kdef value (deformation factor for timber members)*
StartCrossSectionIndex	*index of the start cross-section (according to the local x direction)* *(0 < StartCrossSectionIndex ≤ AxisVMCrossSections.Count)*
EndCrossSectionIndex	*index of the end cross-section (according to the local x direction)* *(0 < EndCrossSectionIndex ≤ AxisVMCrossSections.Count)*
StartExcentricity	*eccentricity at the beginning (according to the local x direction)*
EndExcentricity	*eccentricity at the end (according to the local x direction)*

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	*Material Index*
ServiceClass	*Service class of timber (valid values are 1, 2, 3)*
kdef	*Kdef value (deformation factor for timber members)*
StartCrossSectionIndex	*index of the start cross-section (according to the local x direction)* *(0 < StartCrossSectionIndex ≤ AxisVMCrossSections.Count)*
EndCrossSectionIndex	*index of the end cross-section (according to the local x direction)* *(0 < EndCrossSectionIndex ≤ AxisVMCrossSections.Count)*
StartExcentricity	*eccentricity at the beginning (according to the local x direction)*
EndExcentricity	*eccentricity at the end (according to the local x direction)*

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**)

MaterialIndex	*Material Index*
StartCrossSectionIndex	*index of the start cross-section (according to the local x direction)* *(0 < StartCrossSectionIndex ≤ AxisVMCrossSections.Count)*

EndCrossSectionIndex *index of the end cross-section (according to the local x direction)*
(0 < EndCrossSectionIndex ≤ AxisVMCrossSections.Count)
AutoExcentricityType *Type of the automatic eccentricity*
kx *Friction resistance between rib and surface*
Domain1 *Domain index (filled if connected to domain)*
Domain2 *Domain index (filled if rib is between two domains)*

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**)

MaterialIndex *material Index*
ServiceClass *service class of timber (valid values are 1, 2, 3)*
kdef *Kdef value (deformation factor for timber members)*
StartCrossSectionIndex *index of the start cross-section (according to the local x direction)*
(0 < StartCrossSectionIndex ≤ AxisVMCrossSections.Count)
EndCrossSectionIndex *index of the end cross-section (according to the local x direction)*
(0 < EndCrossSectionIndex ≤ AxisVMCrossSections.Count)
AutoExcentricityType *type of the automatic eccentricity*
kx *Friction resistance between rib and surface*
Domain1 *domain index (filled if connected to domain)*
Domain2 *domain index (filled if rib is between two domains)*

*If successful, returns member index, otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBounds,*lneNotRib *mbeNotRib).*

---

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 *cross-section index*
TrussType *nonlinear behaviour of the truss*
Resistance *if nonzero, resistance of the truss (only if Truss Type <> lnlTensionAndCompression)*

If successful, returns member index, otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBounds, mbeNotTruss).

---

long **GetTrMatrix** ([i/o] RMatrix3x3* **Value**)

Value *the transformation matrix*

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

If successful, returns absolute x position, otherwise returns an 0.

---

long **SetColumnReinforcementParameters** ([i/o] RColumnReinforcementParameters **ColumnReinforcementParameters**)

ColumnReinforcementParameters *column reinforcement parameters*

If successful, returns the member index, otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBounds, mbeInvalidConcreteMaterialId, mbeInvalidConcreteMaterialId, mbeInvalidRebarSteelGradeId, mbeInvalidConcreteMaterialId, mbeInvalidColumnRebarsId, mbeInvalidMemberType, mbeEmptyLineList).

---

long **SetEndReleases** ([i/o] RReleases **Value**)

Set the end releases.
If successful, returns the member index according to IAxisVMMembers, otherwise returns an error code (errDatabaseNotReady).

---

long **SetStartReleases** ([i/o] RReleases **Value**)

Set the start releases.
If successful, returns the member index according to IAxisVMMembers, otherwise returns an error code (errDatabaseNotReady).

Properties

EArchitectElemType **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*

If unsuccessful, returns an error code (errDatabaseNotReady).

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**[ELineStiffnessReduction **Component**]

Get or set stiffness reduction for the given component. Default value is 1. Used only for atLinearStatic and atLinearVibration analysis types. (only valid for Eurocode, SIA, NTS standards)

double **StiffnessReduction_A**

Warning! *This function was extended by StiffnessReduction. 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 Eurocode, SIA, NTS standards)*

double **StiffnessReduction_I**

Warning! *This function was extended by StiffnessReduction. 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 Eurocode, SIA, NTS 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 IAxisVMModel, but moving loads are created through ObjectCreator property in IAxisVMApplication

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 *Index of the moving load*

AxisVMMovingLoadOnDomain *Interface for modifying moving load on domain*

Get moving load on domain interface. If successful, returns moving load index, otherwise an error code (errIndexOutOfBounds, errDatabaseNotReady, errCOMServerInternalError, mleInvalidMovingLoad).

---

long **SetAsMovingLoadOnBeam** ([in] long **Index,** [in] IAxisVMMovingLoadOnBeam **AxisVMMovingLoadOnBeam**)

Index *Index of the moving load*

AxisVMMovingLoadOnBeam *Interface for modifying moving load on beam*

Set as moving load on beam. If successful, returns moving load index, otherwise an error code (errIndexOutOfBounds, errDatabaseNotReady, mleInvalidMovingLoad).

---

long **SetAsMovingLoadOnDomain** ([in] long **Index,** [in] IAxisVMMovingLoadOnDomain **AxisVMMovingLoadOnDomain**)

Index *Index of the moving load*

AxisVMMovingLoadOnDomain *Interface for modifying moving load on domain*

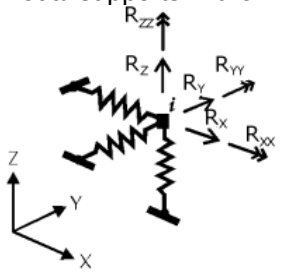
Set as moving load on domain. If successful, returns moving load index, otherwise an error code (errIndexOutOfBounds, errDatabaseNotReady, mleInvalidMovingLoad).

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 ≥ AxisVMNodes.Count*
seLineIndexOutOfBounds = –100002	*line index ≤ 0 or ≥ AxisVMLines.Count*
seReferenceIndexOutOfBounds = –100003	*reference index ≤ 0 or ≥ AxisVMReferences.Count*
seIncompatibleReferences = –100004 }	*Incompatible x and z references for a local support*

Records / structures

RNonlinearity = (

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

)

RResistances = (

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

)

RStiffnesses = (

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

)

Functions

long **AddNodalBeamRelative** ([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 < 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).

long **AddNodalBeamRelative_V153** ([i/o] RNodalSupportSpringParams * NodalSupportSpringParams**,** [in] long **NodeId,** [in] long **BeamId**)

NodalSupportSpringParams *spring characteristics*
NodeId *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 **NodeId,** [in] long **LineId,** [in] long **SurfaceId1,** [in] long **SurfaceId2,** [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*
NodeId *node index*
(0 < Index ≤ AxisVMNodes.Count)
LineId *line index (edge)*
(0 < Index ≤ AxisVMLines.Count)
SurfaceId1 *first connecting surface*
(0 < SurfaceId1 ≤ AxisVMSurfaces.Count)
SurfaceId2 *second connecting surface*
(0 < SurfaceId2 ≤ 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 **NodeId,** [in] long **LineId,** [in] long **SurfaceId1,** [in] long **SurfaceId2,** [in] long **DomainId1,** [in] long **DomainId2,**)

NodalSupportSpringParams *spring characteristics*
NodeId *node index*
(0 < Index ≤ AxisVMNodes.Count)
LineId *line index (edge)*
(0 < Index ≤ AxisVMLines.Count)
SurfaceId1 *first connecting surface*
(0 < SurfaceId1 ≤ AxisVMSurfaces.Count)
SurfaceId2 *second connecting surface*
(0 < SurfaceId2 ≤ 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 **NodeId**)

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*
NodeId	*node index* *(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 **NodeId**)

NodalSupportSpringParams	*spring characteristics*
NodeId	*node index* *(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 **NodeId,** [in] long **ReferenceIdx,** [in] long **ReferenceIdz**)

Warning! *This function has become obsolete, was superseded by AddNodalLocal_V153*

Stiffnesses	*nodal support stiffnesses*
NonLinearity	*nonlinear behaviour of support components*
Resistances	*resistance of support components*
NodeId	*node index* *(0 < Index ≤ AxisVMNodes.Count)*
ReferenceIdx	*reference index for x direction* *(0 < Index ≤ AxisVMReferences.Count)*
ReferenceIdz	*reference index for z direction* *(0 < Index ≤ AxisVMReferences.Count)*

Adds a nodal support to the model, with custom x and z direction. ReferenceIdx defines the x direction, ReferenceIdz 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 **NodeId,** [in] long **ReferenceIdx,** [in] long **ReferenceIdz**)

NodalSupportSpringParams	*spring characteristics*
NodeId	*node index* *(0 < Index ≤ AxisVMNodes.Count)*
ReferenceIdx	*reference index for x direction* *(0 < Index ≤ AxisVMReferences.Count)*
ReferenceIdz	*reference index for z direction* *(0 < Index ≤ AxisVMReferences.Count)*

Adds a nodal support to the model, with custom x and z direction. ReferenceIdx defines the x direction, ReferenceIdz 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, errIndexOutOfBounds, nsePropertyNotValidForThisType, nseInvalidType).

---

long **AddNodalReference** ([i/o] RStiffnesses **Stiffnesses,** [i/o] RNonLinearity **NonLinearity,** [i/o] RResistances **Resistances,** [in] long **NodeId,** [in] long **ReferenceId**)

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*
NodeId	*node index* *(0 < Index ≤ AxisVMNodes.Count)*
ReferenceId	*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 **NodeId,** [in] long **ReferenceId**)

NodalSupportSpringParams	*spring characteristics*
NodeId	*node index* *(0 < Index ≤ AxisVMNodes.Count)*
ReferenceId	*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 **NodeId**)

NodalSupportSpringParams	*spring characteristics*
NodeId	*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 ≤ Index ≤ 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).

Properties

long	**Count** *Get number of nodal supports in the model*
ELongBoolean	**HaveStiffnessCalcParam**[long **Index**] *Returns lbTrue if support has defined parameters for calculating stiffness*
AxisVMNodalSupport*	**Item** [long **Index**] *Get a nodal support interface*
ELongBoolean	**Selected** [long **Index**] • *Get or set the selection status of a nodal support* *NOTE: Call Refresh function afterwards if not called between functions BeginUpdate and EndUpdate*
long	**SelCount** *Get number of selected nodal supports in the model*

IAxisVMNodalSupport

AxisVM nodal support interface.

Enumerated types

enum **ENodalSupportType** = {

nstNodalGlobal = 0x00,	*nodal support in global directions*
nstNodalBeamRelative = 0x01,	*beam / rib relative nodal support*
nstNodalEdgeRelative = 0x02,	*edge relative nodal support*
nstNodalReference = 0x03,	*nodal support in reference direction*
sdNodeRelative = 0x04,	*nodal support in a local system*
sdSeismicIsolator = 0x05 }	*seismic isolator*

Nodal support types

Error codes

enum **ESupportError** = {

seNodeIndexOutOfBounds = –100001	*node index ≤ 0 or ≥ AxisVMNodes.Count*
seLineIndexOutOfBounds = –100002	*line index ≤ 0 or ≥ AxisVMLines.Count*
seReferenceIndexOutOfBounds = –100003 }	*reference index ≤ 0 or ≥ AxisVMReferences.Count*

Records / structures

RNonlinearity = (

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

)

RResistances = (

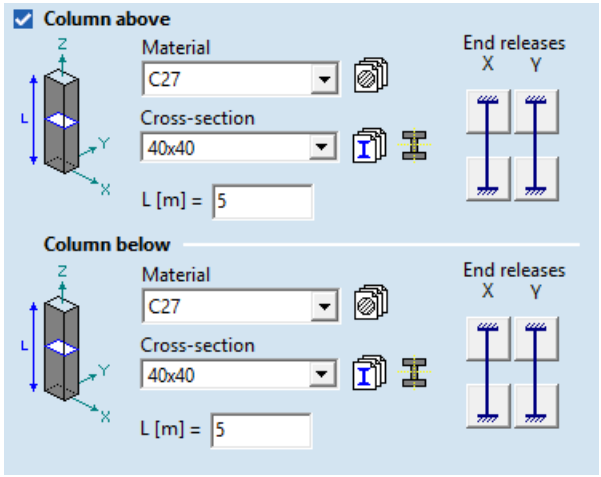
double	**x, y, z,**	*resistance in x, y, z directions [kN]*
	xx, yy, zz	*moment resistances about the x, y, z axis [kNm]*

)

RStiffnesses = (

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

)



Node support calculation

RElementStiffnessParams = (

long	**MaterialID**	*material index. 0 is a valid value, it means this element is not defined*
double	**Height**	*element (column, wall) height*
ElongBoolean	**EndReleaseXTop**	*top of the element is released in glob. x (glob. support) or loc. y direction (beam / rib / edge relative support)*
ElongBoolean	**EndReleaseXBottom**	*bottom of the element is released in glob. x (glob. support) or loc. y direction (beam / rib / edge relative support)*

)

RColumnStiffnessParams = (

RElementStiffnessParams	**CalcParams**	*common calculation parameters*
long	**CrossSectionID**	*index of the cross-section of the supporting element*
ElongBoolean	**EndReleaseYTop**	*Top of the element is released in glob. y (glob. support) or loc. z direction (beam / rib / edge relative support)*
ElongBoolean	**EndReleaseYBottom**	*Bottom of the element is released in glob. y (glob. support) or loc. z direction (beam / rib / edge relative support)*

)

RNodalSupportStiffParams = (

RColumnStiffnessParams	**Top**	*calculation parameters of the supporting element (column) above support*
RColumnStiffnessParams	**Bottom**	*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 parametres 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(errIndexOutOfBounds, errDatabaseNotReady).

long **GetStiffnessCalcParams** ([i/o] RNodalSupportStiffParams **Value**)

Get calculation parameters for calculating the stiffness of the nodal support. If unsuccessful, returns an error code(errIndexOutOfBounds, 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(errIndexOutOfBounds, 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(errIndexOutOfBounds, 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(errIndexOutOfBounds, errDatabaseNotReady).

long **SetStiffnessCalcParams** ([i/o] RNodalSupportStiffParams **Value**)

Set calculation parameters for calculating the stiffness of the nodal support. If unsuccessful, returns an error code(errIndexOutOfBounds, errDatabaseNotReady, nseMaterialIndexOutOfBounds, nseCrossSectionIndexOutOfBounds).

Properties

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

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

EPadFootingType **FootingType** *Type of pad footing if it has been defined in AxisVM*

ELongBoolean **HasFooting** *lbTrue if footing have been defined for the support in AxisVM*

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

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

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

long **NodeId** *node index*

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

RNodalSupportSpringParams **SpringParams** *spring characteristics*

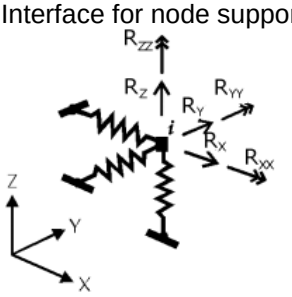
ENodalSupportType **SupportType** *support type*

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

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

IAxisVMNodesSupports

Interface for node supports



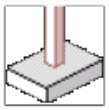
Error codes

enum **ENodesSupportsError** = {

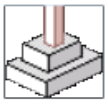
nsePropertyNotValidForThisType = –100001	*LineID and ReferenceID properties can invoke Error event with this code in case it is not applicable.*
nseNodeIndexOutOfBounds = –100002	*node index is our of range*
nseMemberIndexOutOfBounds = –100003	*member index is our of range*
nseReferenceIndexOutOfBounds = –100004	*reference index is our of range*
nseInvalidType = –100005	*not applicable, invalid type of support*
nsePadFootingNotDefined = –100006	*not applicable, pad fotting is not defined*
nseInvalidMemberAndNodeCombination = –100007	*not applicable, node is not attached to the member*
nseStiffnessCalcParamsNotDefined = –100008	*not applicable, stiffness calculation parameters are not defined*
nseMaterialIndexOutOfBounds = –100009	*material index is out of bounds*
nseCrossSectionIndexOutOfBounds = –100010 }	*cross-section index is out of bounds*

Enumerated types

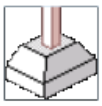
enum **EPadFootingType** = {



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	*rectangular footing in a node*
pft_NodeCircular = 0x01	*circular footing in a node*
pft_Linear = 0x02	*footing along linear elements*

}

General footing class

enum **EPadFootingStepMeasureSource** = *{*

pfsms_Interior = 0x00	*origin of the horizontal delta for the step is the columns base*
pfsms_Edge = 0x01	*origin of the horizontal delta for the step is the outer edge of the footings main element (plate or bottom step)*

}

Origin of the horizontal delta for the step

Records / structures

RSpringParamIndexes = (

long	x	*spring characteristic index for translation in direction x*
long	y	*spring characteristic index for translation in direction y*
long	z	*spring characteristic index for translation in direction z*
long	xx	*spring characteristic index for rotation around ax x*
long	yy	*spring characteristic index for rotation around ax y*
long	zz	*spring characteristic index for rotation around ax z*

)

RNodalSupportSpringParams = (

RSpringParamIndexes	SpringParamIndexes	*Indexes of the spring characteristics (for regular nodal supports, like global node support, domain relative node support, …)*
long	IsolatorParamIndex	*Index of the isolator spring characteristic (only for seismic isolator node support)*
double	IsolatorD2	*Maximum design displacement at ULS [m] (only for seismic isolator node support)*

)

Either SpringParamIndexes or (IsolatorParamIndex, IsolatorD2) has to be filled, depending on the type of the nodal support

RBulkNodalSupportSpringParams = (

ENodalSupportType	SupportType	*specifies how to interpret the rest of the record*
RSpringParamIndexes	SpringParamIndexes	*Indexes of the spring characteristics (only for SupportType nstNodalGlobal, nstNodalBeamRelative, nstNodalEdgeRelative, nstNodalReference, nstTrieder)*
long	IsolatorParamIndex	*Index of the isolator spring characteristic (only for SupportType nstSeismicIsolator)*
double	IsolatorD2	*Maximum design displacement at ULS [m] (only for SupportType nstSeismicIsolator)*
long	NodeId	*node index*
long	MemberId	*member index (only for SupportType nstNodalBeamRelative, nstNodalEdgeRelative)*
long	SurfaceId1	*surface index 1 (only for SupportType nstNodalEdgeRelative)*
long	SurfaceId2	*surface index 2 (only for SupportType nstNodalEdgeRelative)*
long	DomainId1	*domain index 1 (only for SupportType nstNodalEdgeRelative)*
long	DomainId2	*domain index 2 (only for SupportType nstNodalEdgeRelative)*
long	ReferenceId	*reference index (only for SupportType nstNodalReference)*
long	ReferenceIdx	*reference index for local x direction (only for SupportType nstTrieder)*
long	ReferenceIdz	*reference index for local z direction (only for SupportType nstTrieder)*

)

The valid field combinations depending on SupportType are :

SupportType = **nstNodalGlobal**

SpringParamIndexes
NodeId

SupportType = **nstNodalBeamRelative**

SpringParamIndexes
NodeId
MemberId

SupportType = **nstNodalEdgeRelative**

SpringParamIndexes
NodeId

MemberId
SurfaceId1
SurfaceId2
DomainId1
DomainId2

-here the bare minimum is one of fields SurfaceId1 or DomainId1 to be different from zero, the rest from SurfaceId1, SurfaceId2, DomainId1, DomainId2 can be zero. If only one surface is specified it must be SurfaceId1 and SurfaceId2 must be zero. If only one DomainId is specified, it must be DomainId1 and DomainId2 must be zero. If a surface is part of the mesh of a domain, then when SurfaceId is filled, the corresponding DomainId must be filled with the DomainId that contains the surface. If both surfaceIds are filled, the support will be in the direction of the bisector of the two surface planes. If both DomainIds are filled, the support will be in the direction of the bisector of the two domain planes. If SurfaceId1 and DomainId1 is filled, the support will be in the direction of the bisector of the surface and domain planes.

SupportType = **nstNodalReference**
SpringParamIndexes
NodeId
ReferenceId

SupportType = **nstTrieder**
SpringParamIndexes
NodeId
ReferenceIdx
ReferenceIdz

SupportType = **nstSeismicIsolator**
IsolatorParamIndex
IsolatorD2
NodeId

RPadFootingDimensions = (
Warning! *This record has become obsolete, it was superseded by RPadFootingParams_V153 and RLinearFootingParams*

double	UpperThickness	*Thickness of the upper part*
double	LowerThickness	*Thickness of the lower part*
RPoint2D	UpperCornerA	*2D coordinates of point A, Point A is a vertex of rectangular of projected upper part*
RPoint2D	UpperCornerB	*2D coordinates of point B, Point B is a vertex of rectangular of projected upper part*
RPoint2D	LowerCornerA	*2D coordinates of point A, Point A is a vertex of rectangular of projected lower part*
RPoint2D	LowerCornerB	*2D coordinates of point B, Point B is a vertex of rectangular of projected lower part*

)

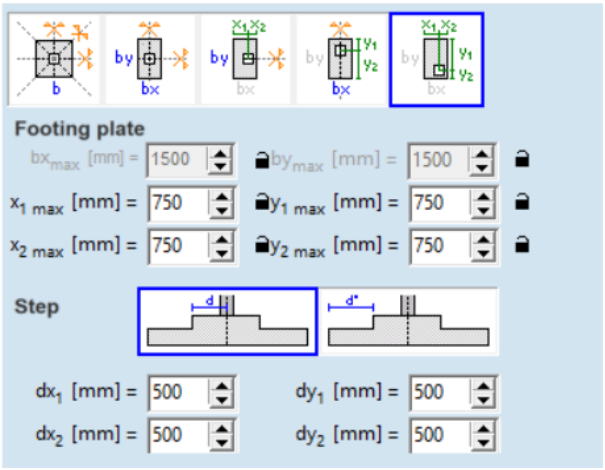
RPadFootingParams = (
Warning! *This record has become obsolete, it was superseded by RPadFootingParams_V153 and RLinearFootingParams*

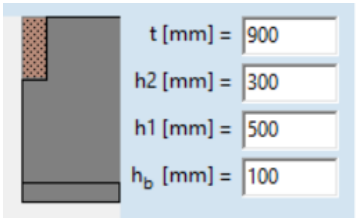
double	HeightTop	*Height of upper part, h2 on form, see pic. below*
double	HeightBottom	*Height of lower part, h1 on form, see pic. below*
double	BaseThickness	*Height of the base, h_b on form, see pic. below*
double	bx	*Total width of the base, negative val. for max., see pic. below*
double	x1	*Left width of the base, negative val. for max., see pic. below*
double	x2	*Right width of the base, negative val. for max., see pic. below*

double	by	*Total length of the base, negative val. for max., node supports only, see pic. below*
double	y1	*Left length of the base, negative val. for max., node supports only, see pic. below*
double	y2	*Right length of the base, negative val. for max., node supports only, see pic. below*
double	dy1	*Left length of the base, negative val. for max., node supports only, see pic. below*
double	dy2	*Right length of the base, negative val. for max., node supports only, see pic. below*
long	MaterialID	*Material index of the concrete footing*
)		

Node supports:

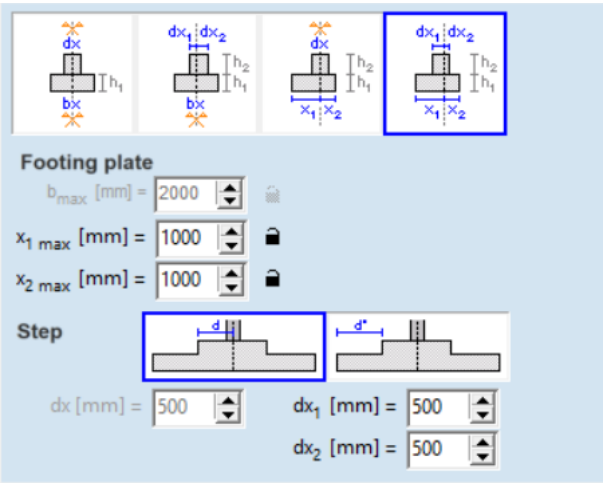
Warning! *This image is related ot an obsolete record*





Member supports:

Warning! *This image is related ot an obsolete record*

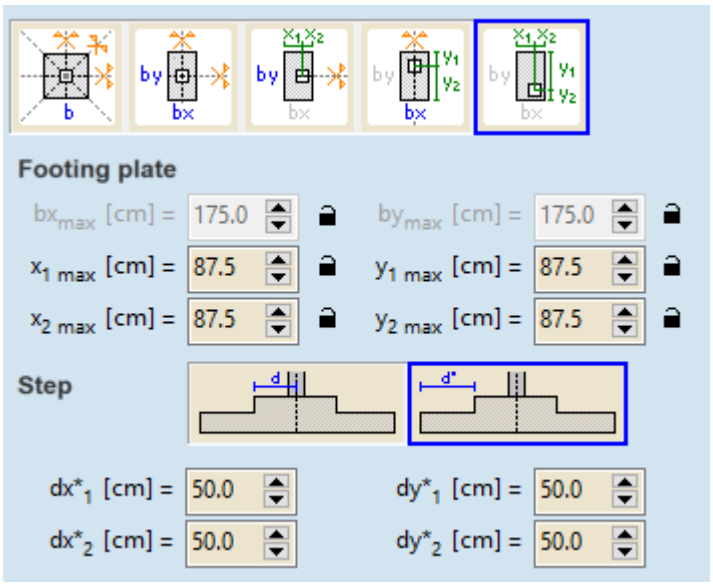


RRectangularFootingSpec = (*see picture below*

ELongBoolean	FixedX1	*x1 is fixed/locked*
ELongBoolean	FixedX2	*x2 is fixed/locked*
double	x1	*value of x1 (left width from the column axis)*
double	x2	*value of x2 (right width from the column axis)*
ELongBoolean	FixedY1	*y1 is fixed/locked*
ELongBoolean	FixedY2	*y2 is fixed/locked*
double	y1	*value of y1 (top width from the column axis)*
double	y2	*value of y2 (bottom width from the column axis)*

EPadFootingStepMeasureSource	StepMeasureSource	*interpretation of the origin of the step horizontal delta. dx1, dx2, dy1, dy2 are interpreted with this rule*
double	dx1	*value of dx1 (step left delta)*
double	dx2	*value of dx2 (step right delta)*
double	dy1	*value of dy1 (step top delta)*
double	dy2	*value of dy2 (step bottom delta)*
	)	

b_x and b_y are derived values, $b_x = x1 + x2$, $b_y = y1 + y2$. Their usage in the user interface can be emulated by specifying their half value in x1, x2, or y1, y2.

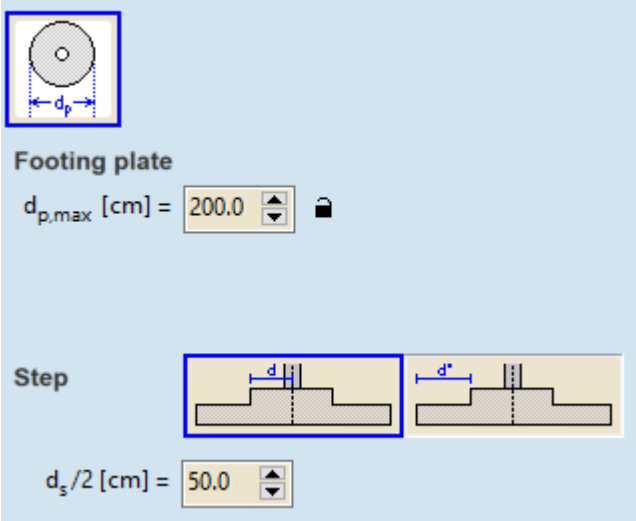


RRectanularFootingCalced = (

ELongBoolean	Calculated	*True if the record contains a valid calculation*
double	x1	*value of x1 (left width from the column axis)*
double	x2	*value of x2 (right width from the column axis)*
double	y1	*value of y1 (top width from the column axis)*
double	y2	*value of y2 (bottom width from the column axis)*
EPadFootingStepMeasureSource	StepMeasureSource	*interpretation of the origin of the step horizontal delta. dx1, dx2, dy1, dy2 are interpreted with this rule*
double	dx1	*value of dx1 (step left delta)*
double	dx2	*value of dx2 (step right delta)*
double	dy1	*value of dy1 (step top delta)*
double	dy2	*value of dy2 (step bottom delta)*
	)	

RCircularFootingSpec = (*see picture below*

ELongBoolean	FixedDiam	*diameter is fixed/locked*
double	Diam	*diameter of the main element (plate or bottom step) (d_p)*
EPadFootingStepMeasureSource	StepMeasureSource	*interpretation of the origin of the step horizontal deltaR*
double	DeltaR	*value of deltaR (step delta) (d_s/2)*
	)	



RCircularFootingCalced = (

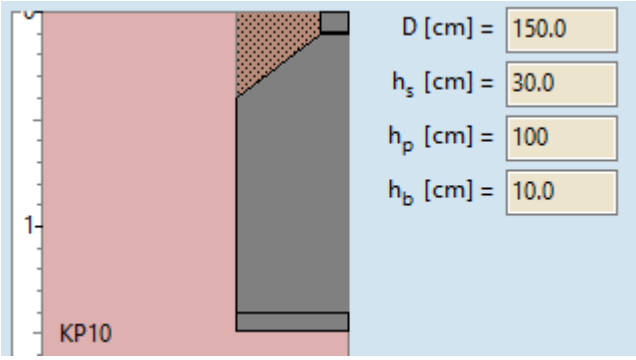
ELongBoolean	Calculated	*True if the record contains a valid calculation*
double	Diam	*diameter of the main element (plate or bottom step)*
EPadFootingStepMeasureSource	StepMeasureSource	*interpretation of the origin of the step horizontal deltaR*
double	DeltaR	*value of deltaR (step delta)*

)

RPadFootingParams_V153 = (*see picture below*

EPadFootingType2	FootingType	*Footing type class*
EPadFootingType	VerticalType	*Step setup*
long	MaterialId	*Material index of the concrete footing*
double	GroundToBottom	*Soil cover over the bottom plane of the footing (D)*
double	HeightMain	*Height of the main element (plate or bottom step) (h_p)*
double	HeightStep	*Step height (h_s)*
double	BlindThickness	*Thickness of the blind concrete (h_b)*
RRectangularFootingSpec	RectangularFootingSpec	*Specified parameters for FootingType = pft_NodeRectangular*
RRectangularFootingCalced	RectanularFootingCalced	*Calculated parameters for FootingType = pft_NodeRectangular*
RCircularFootingSpec	CircularFootingSpec	*Specified parameters for FootingType = pft_NodeCircular*
RCircularFootingCalced	CircularFootingCalced	*Calculated parameters for FootingType = pft_NodeCircular*

)



RLinearFootingParams = (*see picture above*

EPadFootingType	VerticalType	*Step setup*
long	MaterialId	*Material index of the concrete footing*

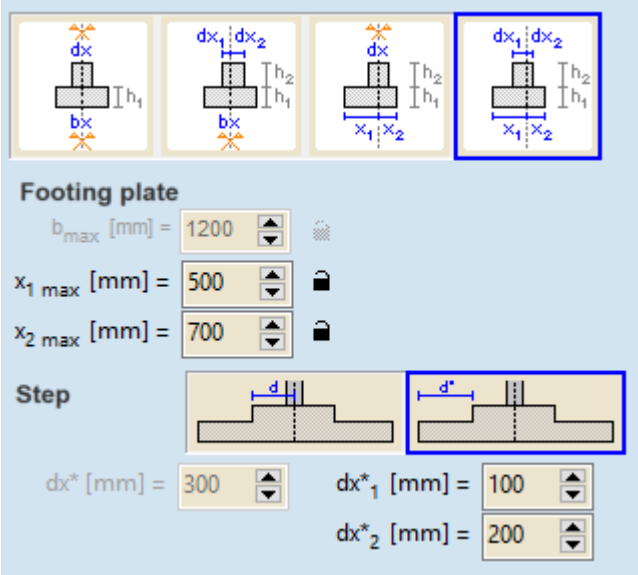
double	GroundToBottom	*Soil cover over the bottom plane of the footing (D)*
double	HeightMain	*Height of the main element (plate or bottom step) (h_p)*
double	HeightStep	*Step height (h_s)*
double	BlindThickness	*Thickness of the blind concrete (h_b)*
RLinearFootingSpec	FootingSpec	*Specified parameters for the line support*
RLinearFootingCalced	FootingCalced	*Calculated parameters for the line support*

)

RLinearFootingSpec = (*see picture below*

ELongBoolean	FixedX1	*x1 is fixed/locked*
ELongBoolean	FixedX2	*x2 is fixed/locked*
double	x1	*value of x1 (left width from the wall axis)*
double	x2	*value of x2 (right width from the wall axis)*
EPadFootingStepMeasureSource	StepMeasureSource	*interpretation of the origin of the step horizontal delta. dx1, dx2 are interpreted with this rule*
double	dx1	*value of dx1 (step left delta)*
double	dx2	*value of dx2 (step right delta)*

)



RLinearFootingCalced = (*See picture above*

ELongBoolean	Calculated	*True if the record contains a valid calculation*
double	x1	*value of x1 (left width from the wall axis)*
double	x2	*value of x2 (right width from the wall axis)*
EPadFootingStepMeasureSource	StepMeasureSource	*interpretation of the origin of the step horizontal delta. dx1, dx2 are interpreted with this rule*
double	dx1	*value of dx1 (step left delta)*
double	dx2	*value of dx2 (step right delta)*

)

Functions

long **AddIsolator** ([i/o] RNodalSupportSpringParams * NodalSupportSpringParams**,** [in] long **NodeId**)

NodalSupportSpringParams *spring characteristics*
NodeId *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 **AddNodalGlobal** ([i/o] RStiffnesses **Stiffnesses,** [i/o] RNonLinearity **NonLinearity,** [i/o] RResistances **Resistances,** [in] long **NodeId**)

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*
NodeId	*node index (0 < Index ≤ 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 **NodeId**)

NodalSupportSpringParams	*spring characteristics*
NodeId	*node index (0 < Index ≤ 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, errIndexOutOfBounds, nsePropertyNotValidForThisType, nseInvalidType).

---

long **AddNodalLocal_V153** ([i/o] RNodalSupportSpringParams * NodalSupportSpringParams**,** [in] long **NodeId,** [in] long **ReferenceIdx,** [in] long **ReferenceIdz**)

NodalSupportSpringParams	*spring characteristics*
NodeId	*node index (0 < Index ≤ AxisVMNodes.Count)*
ReferenceIdx	*reference index for x direction (0 < Index ≤ AxisVMReferences.Count)*
ReferenceIdz	*reference index for z direction (0 < Index ≤ AxisVMReferences.Count)*

Adds a nodal support to the model, with custom x and z direction. ReferenceIdx defines the x direction, ReferenceIdz 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, errIndexOutOfBounds, nsePropertyNotValidForThisType, nseInvalidType).

---

long **AddNodalMemberRelative** ([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 AddNodalMemberRelative_V153*

Stiffnesses	*nodal support stiffnesses*
NonLinearity	*nonlinear behaviour of support components*
Resistances	*resistance of support components*
NodeId	*node index (0 < Index ≤ AxisVMNodes.Count)*
MemberID	*member index (beam or rib) (0 < Index ≤ AxisVMMembers.Count)*

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).

---

long **AddNodalMemberRelative_V153** ([i/o] RNodalSupportSpringParams * NodalSupportSpringParams**,** [in] long **NodeId,** [in] long **BeamId**)

NodalSupportSpringParams	*spring characteristics*
NodeId	*node index (0 < Index ≤ AxisVMNodes.Count)*

MemberID *member index (beam or rib)*
(0 < Index ≤ AxisVMMembers.Count)

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).

---

long **AddNodalDomainRelative** ([i/o] RStiffnesses **Stiffnesses,**
[i/o] RNonLinearity **NonLinearity,** [i/o] RResistances **Resistances,** [in] long **NodeId,**
[in] long **MemberID,** [in] long **DomainId1,** [in] long **DomainId2**)

Warning! *This function has become obsolete, was superseded by AddNodalDomainRelative_V153*

Stiffnesses *nodal support stiffnesses*
NonLinearity *nonlinear behaviour of support components*
Resistances *resistance of support components*
NodeId *node index (0 < Index ≤ AxisVMNodes.Count)*
MemberID *member index (beam or rib) (0 < Index ≤ AxisVMMembers.Count)*
DomainId1 *first connecting domain (0 < DomainId1 ≤ AxisVMDomains.Count)*
DomainId2 *second connecting domain, could be 0 (0 < DomainId2 ≤ 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 **NodeId,** [in] long **MemberID,** [in] long **DomainId1,** [in] long **DomainId2**)

NodalSupportSpringParams *spring characteristics*
NodeId *node index (0 < Index ≤ AxisVMNodes.Count)*
MemberID *member index (beam or rib) (0 < Index ≤ AxisVMMembers.Count)*
DomainId1 *first connecting domain (0 < DomainId1 ≤ AxisVMDomains.Count)*
DomainId2 *second connecting domain, could be 0 (0 < DomainId2 ≤ 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 **NodeId,** [in] long **ReferenceId**)

Warning! *This function has become obsolete, was superseded by AddNodalReferenceRelative_V153*

Stiffnesses *nodal support stiffnesses*
NonLinearity *nonlinear behaviour of support components*
Resistances *resistance of support components*
NodeId *node index (0 < Index ≤ AxisVMNodes.Count)*
ReferenceId *reference index (0 < Index ≤ AxisVMReferences.Count)*

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 **NodeId,** [in] long **ReferenceId**)

NodalSupportSpringParams *spring characteristics*
NodeId *node index (0 < Index ≤ AxisVMNodes.Count)*
ReferenceId *reference index (0 < Index ≤ AxisVMReferences.Count)*

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 **BulkAdd** ([in] SAFEARRAY (RBulkNodalSupportSpringParams) **NodalSupports,** [out] SAFEARRAY (long) *** Indexes**)

NodalSupports *list of nodal support records*
Indexes *list of the support indexes that were created. The first index is for the first element of NodalSupports, and so on. If a particular record contained invalid data, that support won't be added to the database, and an index of 0 will be returned at the place corresponding to that record*

Adds several nodal supports in one call. It is faster than calling the respective singular adds for each support. Returns the number of the successfully created nodal supports (will be zero if none of them were valid). Can trigger the following errors through events (errDatabaseNotReady, nseNodeIndexOutOfBounds, nseInvalidMemberAndNodeCombination, nseReferenceIndexOutOfBounds, nsePropertyNotValidForThisType, nseInvalidType, errIndexOutOfBounds)

---

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*

Index *node support index*
FootingParams *pad footing parameters*

Get the footing design calculation parametres of the support. If successful, returns the index of the line support, otherwise returns an error code(errDatabaseNotReady).

---

long **GetFootingParams_V153** ([in] long **Index**, [i/o] RPadFootingParams_V153 **FootingParams**)

Index *node support index*
FootingParams *pad footing parameters*

Get the specified and calculated parameters of the footing. If successful, returns the index of the line support, otherwise returns an error code(errDatabaseNotReady).

---

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

Index *node support index*

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).

---

long **GetNonLinearity** ([in] long **Index,** [i/o] RNonLinearity **NonLinearity**)

Index *node support index*
NonLinearity *node support non-linearity*

Get the nonlinear behaviour of the node support. If unsuccessful, returns an error code(errIndexOutOfBounds, errDatabaseNotReady).

---

long **GetResistances** ([in] long **Index,** [i/o] RResistances **Resistances**)

Index *node support index*
Resistances *node support resistances*

Get the resistance components of the node support. If unsuccessful, returns an error code(errIndexOutOfBounds, errDatabaseNotReady).

---

long **GetStiffnesses** ([in] long **Index,** [i/o] RStiffnesses **Stiffness**)

Index *node support index*
Stiffness *node support stiffnesses*

Get the stiffness components of the node support. If unsuccessful, returns an error code(errIndexOutOfBounds, errDatabaseNotReady).

---

long **GetStiffnessCalcParams** ([in] long **Index,** [i/o] RNodalSupportStiffParams **StiffnessParams**)

Index *node support index*
StiffnessParams *node support stiffnesses parameters*

Get calculation parameters for calculating the stiffness of the node support. If unsuccessful, returns an error code(errIndexOutOfBounds, errDatabaseNotReady).

---

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

ItemIds *list of selected node supports*

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

---

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

Index *node support index*
TrMatrix *Transformation matrix of the node support*

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

---

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 clculation parameters*
Set calculation parameters for calculating the stiffness of the node support. If unsuccessful, returns an error code(errIndexOutOfBounds, errDatabaseNotReady, nseMaterialIndexOutOfBounds, nseCrossSectionIndexOutOfBounds).

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	**ReferenceId** [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 our of range*
msePadFootingNotDefined = –100002	*not applicable, pad fotting 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*
mseMaterialIndexOutOfBounds = –100006 }	*material index is out of bounds*

Records / structures

RBulkMemberSupport = (

ELineSupportType	**SupportType**	*type of the support*
long	**MemberId**	*identifier of the member (1 ≤ MemberId ≤ AxisVMMembers.Count)*
RStiffnesses	**Stiffnesses**	*stiffnesses components. For support types lstRibElasticFoundation, lstBeamElasticFoundation, only the first 3 components are taken into account : x, y, z*
RNonLinearity	**NonLinearity**	*nonlinearity components. For support types lstRibElasticFoundation, lstBeamElasticFoundation, only the first 3 components are taken into account : x, y, z*
RResistances	**Resistances**	*resistance components. For support types lstRibElasticFoundation, lstBeamElasticFoundation, only the first 3 components are taken into account : x, y, z*
long	**SurfaceId1**	*surface index 1. Applicable only if the member is composed from a single line. Only for support types lstEdgeGlobal, lstEdgeRelative, lstEdgeReference. If it is specified, DomainId1 must be 0.(0 ≤ SurfaceId1 ≤ AxisVMSurfaces.Count)*
long	**SurfaceId2**	*surface index 2. Applicable only if the member is composed from a single line. Only for support types lstEdgeGlobal, lstEdgeRelative, lstEdgeReference. Must be zero if only one surface is connecting to the support. .If it is specified, DomainId1, DomainId2 must be 0. (0 ≤ SurfaceId2 ≤ AxisVMSurfaces.Count)*
long	**DomainId1**	*domain index 1. Only for support types lstEdgeGlobal, lstEdgeRelative, lstEdgeReference. If it is specified, SurfaceId1, SurfaceId2 must be 0. (0 ≤ DomainId1 ≤ AxisVMDomains.Count)*
long	**DomainId2**	*domain index 2. Only for support types lstEdgeGlobal, lstEdgeRelative, lstEdgeReference. Must be zero if only one domain is connecting to the support.(0 ≤ DomainId2 ≤ AxisVMDomains.Count)*
long	**ReferenceId**	*reference index. For support type lstEdgeReference it must be greater than zero, for the other support types it must be zero. (0 ≤ ReferenceId ≤ AxisVMReferences.Count)*

)

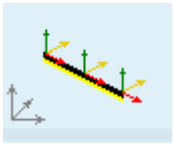
RBulkMemberWSSupport = (

ELineSupportType	**SupportType**	*type of the support*
long	**MemberId**	*identifier of the member (1 ≤ MemberId ≤ AxisVMMembers.Count)*
RStiffnesses	**Stiffnesses**	*stiffnesses components. For support types lstRibElasticFoundation, lstBeamElasticFoundation, only the first 3 components are taken into account : x, y, z*
RNonLinearity	**NonLinearity**	*nonlinearity components. For support types lstRibElasticFoundation, lstBeamElasticFoundation, only the first 3 components are taken into account : x, y, z*
RResistances	**Resistances**	*resistance components. For support types lstRibElasticFoundation, lstBeamElasticFoundation, only the first 3 components are taken into account : x, y, z*
double	**ShearStiffness**	*shear stiffness of the support (noted as RG on the window for supports) [kN]. If it is 0, a traditional Winkler support will be created*
long	**SurfaceId1**	*surface index 1. Applicable only if the member is composed from a single line. Only for support types lstEdgeGlobal, lstEdgeRelative, lstEdgeReference. If it is specified, DomainId1 must be 0.(0 ≤ SurfaceId1 ≤ AxisVMSurfaces.Count)*
long	**SurfaceId2**	*surface index 2. Applicable only if the member is composed from a single line. Only for support types lstEdgeGlobal, lstEdgeRelative, lstEdgeReference. Must be zero if only one surface is connecting to the support. .If it is specified, DomainId1, DomainId2 must be 0. (0 ≤ SurfaceId2 ≤ AxisVMSurfaces.Count)*
long	**DomainId1**	*domain index 1. Only for support types lstEdgeGlobal, lstEdgeRelative, lstEdgeReference. If it is specified, SurfaceId1, SurfaceId2 must be 0. (0 ≤ DomainId1 ≤ AxisVMDomains.Count)*
long	**DomainId2**	*domain index 2. Only for support types lstEdgeGlobal, lstEdgeRelative, lstEdgeReference. Must be zero if only one domain is connecting to the support.(0 ≤ DomainId2 ≤ AxisVMDomains.Count)*
long	**ReferenceId**	*reference index. For support type lstEdgeReference it must be greater than zero, for the other support types it must be zero. (0 ≤ ReferenceId ≤ AxisVMReferences.Count)*

)

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, errIndexOutOfBounds).

---

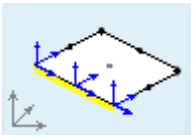
long **AddDomainEdgePasternakSupport** ([in] long **MemberID,** [in] ELineSupportType **SupportType,** [i/o] RStiffnesses **Stiffnesses,** [in] double **ShearStiffness,** [i/o] RNonLinearity **Nonlinearity,** [i/o] RResistances **Resistances,** [in] long **DomainId1,** [in] long **DomainId2**)

MemberID	*index of the member*
SupportType	*type of the member support*
Stiffnesses	*stiffness components in the local system of the edge*
ShearStiffness	*shear stiffness of the support (noted as R_G on the window for supports) [kN]*
Nonlinearity	*nonlinear behaviour of the support*
Resistances	*resistances for stiffness components*
DomainId1	*first connecting domain* *(0 < DomainId1 ≤ AxisVMDomains.Count)*
DomainId2	*second connecting domain, can be 0* *(0 < DomainId2 ≤ AxisVMDomains.Count)*

Adds a Winkler-Pasternak domain edge support. Attached domains are stored so that if all of them are deleted the support is deleted as well. If only the first attached domain is specified (the second one 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 support index, otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBounds).

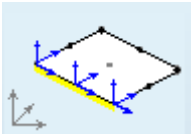
---

long **AddDomainEdgeSupport** ([in] long **MemberID,** [in] ELineSupportType **SupportType,**
[i/o] RStiffnesses **Stiffnesses,** [i/o] RNonLinearity **Nonlinearity,**
[i/o] RResistances **Resistances,** [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*
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 domains are stored so that if all of them are deleted the support is deleted as well. If only the first attached domain is specified (the second one 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 support index, otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBounds).



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 **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*
DomainId1	*first connecting domain (0 < DomainId1 ≤ AxisVMDomains.Count)*
DomainId2	*second connecting domain, can be 0 (0 < DomainId2 ≤ AxisVMDomains.Count)*

Adds an domain edge support where loc. z vector of the support is set by reference. Attached domains are stored so that if all of them are deleted the support is deleted as well. If only the first attached domain is specified (the second one 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 support index, otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBounds).

---

long **AddPasternakSupport** ([in] long **MemberID,** [i/o] RStiffnessesXYZ **StiffnessesXYZ,** [in] double **ShearStiffness,** [i/o] RNonLinearityXYZ **NonlinearityXYZ,** [i/o] RResistancesXYZ **ResistancesXYZ**)

MemberID	*index of the member*
StiffnessesXYZ	*stiffnesses of the elastic foundation in local direction*
ShearStiffness	*shear stiffness of the support (noted as R_G on the window for supports) [kN]*
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, errIndexOutOfBounds).

---

long **BulkAdd** ([in] SAFEARRAY(RBulkMemberSupport) **MemberSupports,**[out] SAFEARRAY(long)* **Indexes**)

MemberSupports	*list of support records to be created*
Indexes	*list of the created support indexes. If a particular support couldn't be created, its corresponding index will be 0. An index of -1 indicates that the support has been created, but its exact index couldn't be determined.*

Creates multiple supports in one step. It is faster than caling the respective singular functions successively. Returns the number created member supports, which can be zero if none were created.

It can trigger the following events in case of an error (some of them may be triggered on a by record case) :

IAxisVMMembersSupportsEvents.Error, with the errorcodes (errDatabaseNotReady)

IAxisVMMembersEvents.Error, with the errorcodes (errIndexOutOfBounds, mbeEmptyLineList, mseInvalidType)

---

long **BulkAddPasternak** ([in] SAFEARRAY(RBulkMemberWSSupport) **MemberSupports,**[out] SAFEARRAY(long)* **Indexes**)

MemberSupports	*list of support records to be created*
Indexes	*list of the created support indexes. If a particular support couldn't be created, its corresponding index will be 0. An index of -1*

indicates that the support has been created, but its exact index couldn't be determined.

Creates multiple supports in one step. If the value of the field ShearStiffness is set to 0 in a record, a traditional Winkler support is created for that. Thus, BulkAddPasternak can be used to create mixed Winkler, Winkler-Pasternak supports. It is faster than caling the respective singular functions successively. Returns the number created member supports, which can be zero if none were created.

It can trigger the following events in case of an error (some of them may be triggered on a by record case) :

IAxisVMMembersSupportsEvents.Error, with the errorcodes (errDatabaseNotReady)

IAxisVMMembersEvents.Error, with the errorcodes (errIndexOutOfBounds, mbeEmptyLineList, mseInvalidType)

---

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

Index *member support index*

Deletes a member support. If successful, returns the number of deleted member supports, otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBounds).

---

long **DeleteSelected**

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

---

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*

Get the calculated dimensions of the footing. If successful, returns the index of the member support, otherwise returns an error code (errIndexOutOfBounds, errDatabaseNotReady).

---

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*

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).

---

long **GetFootingParams_V153** ([in] long **Index**, [i/o] RLinearFootingParams **FootingParams**)

Index *member support index*

FootingParams *pad footing parameters*

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).

---

long **GetLineSupportIds** ([in] long **Index,** [out] SAFEARRAY (long) *** LineSupportIds**)

Index *member support index*

LineSupportIds *list of line support indexes IAxisVMLineSupports*

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).

---

long **GetNonLinearity** ([in] long **Index,** [i/o] RNonLinearity **NonLinearity**)

Index *member support index*

NonLinearity *member support non-linearity*

Get the nonlinear behaviour of the member support. If unsuccessful, returns an error code(errIndexOutOfBounds, errDatabaseNotReady).

---

long **GetPasternakStiffness** ([in] long **Index,** [i/o] double* **Value**)

Index *member support index*
Value *shear stiffness of the member support (noted as R_G on the window for supports) [kN]*

Get the stiffness components of the member support. If unsuccessful, returns an error code(errIndexOutOfBounds, errDatabaseNotReady).

---

long **GetResistances** ([in] long **Index,** [i/o] RResistances **Resistances**)

Index *member support index*
Resistances *nonlinear behaviour of support components*

Get the resistance components of the member support. If unsuccessful, returns an error code(errIndexOutOfBounds, errDatabaseNotReady).

---

long **GetStiffnesses** ([in] long **Index,** [i/o] RStiffnesses **Stiffness**)

Index *member support index*
Stiffness *member support* stiffness

Get the stiffness components of the member support. If unsuccessful, returns an error code(errIndexOutOfBounds, errDatabaseNotReady).

---

long **GetStiffnessCalcParams** ([in] long **Index,** [i/o] RLineSupportStiffParams **Value**)

Index *member support index*
StiffnessParams *member support stiffnesses parameters*

Get calculation parameters for calculating the stiffness of the member support. If unsuccessful, returns an error code(errIndexOutOfBounds, errDatabaseNotReady).

---

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

ItemIds *list of selected member supports*

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

---

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

Index *member support index*
TrMatrix *Transformation matrix of the member support*

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

---

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_"*

Rename selected member 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 member 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 *member support index*
NonLinearity *member support non-linearity*

Set the nonlinear behaviour of the member support. If unsuccessful, returns an error code(errIndexOutOfBounds, errDatabaseNotReady).

---

long **SetPasternakStiffness** ([in] long **Index,** [in] double **Value**)

Index *member support index*

Value *shear stiffness of the member support (noted as R_G on the window for supports) [kN]*

Get the stiffness components of the member support. If unsuccessful, returns an error code(errIndexOutOfBounds, errDatabaseNotReady).

---

long **SetResistances** ([in] long **Index,** [i/o] RResistances **Resistances**)

Index *member support index*
Resistances *nonlinear behaviour of support components*

Set the resistance components of the member support. If unsuccessful, returns an error code(errIndexOutOfBounds, errDatabaseNotReady).

---

long **SetStiffnesses** ([in] long **Index,** [i/o] RStiffnesses **Stiffness**)

Index *member support index*
Stiffness *member support* stiffness

Set the stiffness components of the member support. If unsuccessful, returns an error code(errIndexOutOfBounds, errDatabaseNotReady).

---

long **SetStiffnessCalcParams** ([in] long **Index,**
[i/o] RLineSupportStiffParams **StiffnessParams**)

Index *member support index*
StiffnessParams *member support stiffnesses parameters*

Set calculation parameters for calculating the stiffness of the member support. If unsuccessful, returns an error code(errIndexOutOfBounds, errDatabaseNotReady, mseMaterialIndexOutOfBounds)

Properties

long **Count** *Get number of member 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**] *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 member support UID*

long **MemberID** [long **Index**] *member index*

BSTR **Name** [long **Index**] *member support name*

long **NodeId** [long **Index**] *node index*

long **ReferenceId** [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*

NOTE: Call Refresh function afterwards if not called between functions BeginUpdate and EndUpdate

ELineSupportType **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 member support*

IAxisVMDomainsSupports

Domain supports of the model.

Note:

In this interface you can access supports of the domains.

Functions

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

DomainId	*index of the domain*
StiffnessesXYZ	*stiffnesses of the domain support*
ShearStiffness	*shear stiffness of the domain support (noted as R_G on the window for supports) [kN/m]*
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, errIndexOutOfBounds).

---

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

DomainId	*index of the domain*
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, errIndexOutOfBounds).

---

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

Index	*domain support index*

Deletes a domain support. If successful, returns the number of deleted domain supports, otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBounds).

---

long **DeleteSelected**

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

---

long **GetNonLinearity** ([in] long **Index,** [i/o] RNonLinearity **NonLinearity**)

Index	*domain support index*
NonLinearity	*domain support non-linearity*

Get the nonlinear behaviour of the domain support. If unsuccessful, returns an error code(errIndexOutOfBounds, errDatabaseNotReady).

---

long **GetPasternakStiffness** ([in] long **Index,** [i/o] double* **ShearStiffness**)

Index	*domain support index*
ShearStiffness	*shear stiffness of the domain support (noted as R_G on the window for supports) [kN/m]*

Get the shear stiffness of the domain support. If unsuccessful, returns an error code(errIndexOutOfBounds, errDatabaseNotReady).

---

long **GetResistances** ([in] long **Index,** [i/o] RResistances **Resistances**)

Index	*domain support index*
Resistances	*nonlinear behaviour of support components*

Get the resistance components of the member support. If unsuccessful, returns an error code(errIndexOutOfBounds, errDatabaseNotReady).

---

long **GetStiffnesses** ([in] long **Index,** [i/o] RStiffnesses **Stiffness**)

Index *domain support index*
Stiffness *domain support* stiffness

Get the stiffness components of the domain support. If unsuccessful, returns an error code(errIndexOutOfBounds, errDatabaseNotReady).

---

long **GetStiffnessCalcParams** ([in] long **Index,** [i/o] RLineSupportStiffParams **Value**)

Index *domain support index*
StiffnessParams *domain support stiffnesses parameters*

Get calculation parameters for calculating the stiffness of the domain support. If unsuccessful, returns an error code(errIndexOutOfBounds, errDatabaseNotReady).

---

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

ItemIds *list of selected domain supports*

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

---

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_"*

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

---

long **SetNonLinearity** ([in] long **Index,** [i/o] RNonLinearity **NonLinearity**)

Index *domain support index*
NonLinearity *domain support non-linearity*

Set the nonlinear behaviour of the member support. If unsuccessful, returns an error code(errIndexOutOfBounds, errDatabaseNotReady).

---

long **SetPasternakStiffness** ([in] long **Index,** [in] double **ShearStiffness**)

Index *domain support index*
ShearStiffness *shear stiffness of the domain support (noted as R_G on the window for supports) [kN/m]*

Set the shear stiffness of the domain support. If unsuccessful, returns an error code(errIndexOutOfBounds, errDatabaseNotReady).

---

long **SetResistances** ([in] long **Index,** [i/o] RResistances **Resistances**)

Index *domain support index*
Resistances *nonlinear behaviour of support components*

Set the resistance components of the domain support. If unsuccessful, returns an error code(errIndexOutOfBounds, errDatabaseNotReady).

---

long **SetStiffnesses** ([in] long **Index,** [i/o] RStiffnesses **Stiffness**)

Index *domain support index*
Stiffness *domain support* stiffness

Set the stiffness components of the domain support. If unsuccessful, returns an error code(errIndexOutOfBounds, errDatabaseNotReady).

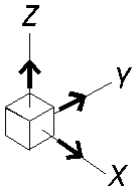
Properties

long **Count** *Get number of domain supports in the model*
long **DomainID** [long **Index**] *domain index*
IAxisVMDomainSupport **Item** [long **Index**] *domain support object by index*
long **IndexOfUID** [long **UID**] *domain support index of domain support UID*

BSTR	**Name** [long **Index**] *domain support name*
long	**SelCount** *Get number of selected domain supports in the model*
ELongBoolean	**Selected** [long **Index**] • *Get or set the selection status of a domain support* *NOTE: Call Refresh function afterwards if not called between functions BeginUpdate and EndUpdate*
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 domain support*

IAxisVMNodes

Nodes in the model.



Enumerated types

enum **EDegreeOfFreedom** = {

dofFree = 0x00	*no constraint (free node)*
dofXfix = 0x01,	*no displacement in X direction*
dofYfix = 0x02,	*no displacement in Y direction*
dofZfix = 0x04,	*no displacement in Z direction*
dofXXfix = 0x08,	*no rotation about X direction*
dofYYfix = 0x10,	*no rotation about Y direction*
dofZZfix = 0x20,	*no rotation about Z direction*
dofwfix = 0x40,	*no warping. Taken into account only for nodes, the diaphragms will ignore it*
dofTrussAndMembraneXZ = 0x3A,	*dofYfix+dofXXfix+dofYYfix+dofZZfix = 2+8+16+32 = 58*
dofFrameYZ = 0x31,	*dofXfix+dofYYfix+dofZZfix=1+16+32= 49*
dofFrameXZ = 0x2A,	*dofYfix+dofXXfix+dofZZfix=2+8+32=42*
dofFrameXY = 0x1C,	*dofZfix+dofXXfix+dofYYfix=4+8+16=28*
dofPlateYZ = 0x0E,	*dofYfix+dofZfix+dofXXfix=2+4+8= 14*
dofPlateXZ = 0x15,	*dofXfix+dofZfix+dofYYfix=1+4+16= 21*
dofPlateXY = 0x23}	*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 safisfy : (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 **BulkDelete** ([in] SAFEARRAY(long) **Indexes**)

Indexes *Array of node indexes. Each index must safisfy : (1 ≤ Index ≤ AxisVMNodes.Count)*

Deletes the nodes in the Indexes list. Use this function if a large number of nodes are to be deleted, it will be significantly faster than calling Delete 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, errIndexOutOfBounds).

---

long **BulkGetCoord** ([in] SAFEARRAY(long) **Indexes,** [out] SAFEARRAY(RPoint3d)* **Coords**)

Indexes *Array of node indexes. Each index must safisfy : (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 Delete for each node individually. The return value on success is the length of Indexes. The possible error codes are(errDatabaseNotReady, errInternalException, errIndexOutOfBounds, errOutOfMemory).

---

long **BulkGetDOF** ([in] SAFEARRAY(long) **Indexes,** [out] SAFEARRAY(long)* **DOFs**)

Indexes *Array of node indexes. Each index must safisfy : (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, errIndexOutOfBounds, errOutOfMemory).

---

long **BulkSelect** ([in] SAFEARRAY(long) **Indexes,** [in] ELongBoolean **Value**)

Indexes *Array of node indexes. Each index must safisfy : (1 ≤ Index ≤ AxisVMNodes.Count)*
Value *Selection state*

Set the selection state of the nodes in the Indexes list to Value. Use this function if a large number of nodes are to be Selected, it will be significantly faster than calling Selected for each node individually. The return value on success is the length of Indexes. The possible error codes are(errDatabaseNotReady, errIndexOutOfBounds).

---

long **BulkSetDOF** ([in] SAFEARRAY(long) **Indexes,** [in] SAFEARRAY(long)* **DOFs**)

Indexes *Array of node indexes. Each index must safisfy : (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 errIndexOutOfBounds will result. The possible error codes are(errDatabaseNotReady, errInternalException, errIndexOutOfBounds, errOutOfMemory).

---

long **BulkSetNodeCoord** ([in] SAFEARRAY(long) **Indexes,** [in] SAFEARRAY(RPoint3d)* **Coords**)

Indexes *Array of node indexes. Each index must safisfy : (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 errIndexOutOfBounds will result. For creating new nodes use the BulkAdd function instead. The possible error codes are(errDatabaseNotReady, errInternalException, errIndexOutOfBounds, 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 ≤ Index ≤ 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)* **LineIdxList**)

Index *node index*
LineIdxList *line indexes according to AxisVMLines*

Obtains indexes of lines connecting to the node. 1 ≤ Index ≤ Count.
If successful, returns the number of lines connecting to the node otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBounds).

---

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

Index *node index*
SurfaceIdxList *surface indexes according to AxisVMSurfaces*

Obtains indexes of surface elements connecting to the node. 1 ≤ Index ≤ Count.
If successful, returns the number of surface elements connecting to the node otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBounds).

---

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 **NodeId,** [out] SAFEARRAY (long) *** LineIds**)

NodeId *node index. It must satisfy: (1 ≤ NodeId ≤ AxisVMNodes.Count)*
LineIds *list of lines having NodeID 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 nodes 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 ≤ Index ≤ 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*

AxisVMAttributes* **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*

NOTE: Call Refresh function afterwards if not called between functions BeginUpdate and EndUpdate

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.

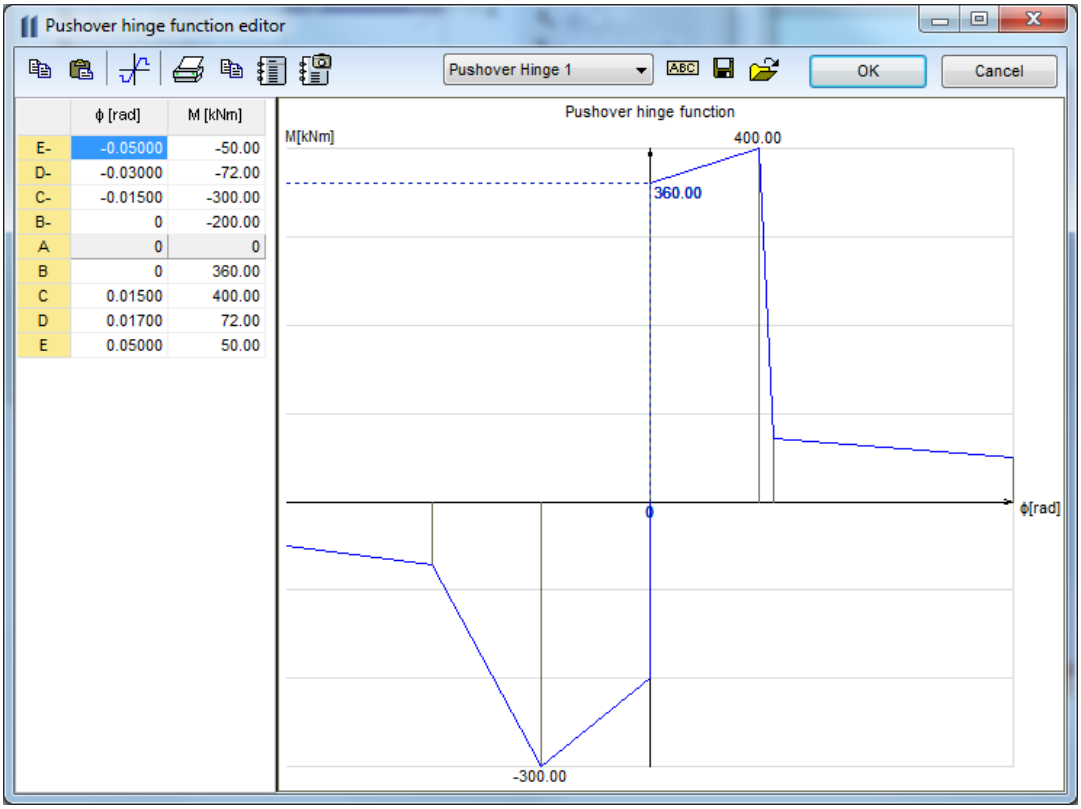
Functions

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 tota,l 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:



	ϕ [rad]	M [kNm]
E-	-0.05000	-50.00
D-	-0.03000	-72.00
C-	-0.01500	-300.00
B-	0	-200.00
A	0	0
B	0	360.00
C	0.01500	400.00
D	0.01700	72.00
E	0.05000	50.00

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*

If successful, returns number of points of the function, otherwise an error code (errDatabaseNotReady, errIndexOutOfBounds)

---

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

Name *Name of the pushover hinge function*

If successful, returns pushover hinge function index, otherwise an error code (errDatabaseNotReady).

---

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.*

If successful, returns function index, 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 *pushover hinge function index*

FileName *Name of the file used for saving without path*

Saves function to AxisVM directory \pohinge with file extension: poh

If successful, returns pushover hinge function index, otherwise an error code (errDatabaseNotReady, errIndexOutOfBounds or fueFileExists).

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 are 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	**RRCBeamCrossSections**	*RC cross section at start and end*
RRCBeamSupports	**RRCBeamSupports**	*RC beam support at start and end*

)
Partial RC Beam design parameters

RRCBeamDesignParameters = (
Warning! *This record was superseded by RRCBeamDesignParameters_V173*

long	**ConcreteMaterial**	*index of the concrete material (index in IAxisVMMaterials)*
double	**Dmax**	*max. size of the aggregate [m]*
long	**RebarMaterial**	*index of the rebar material (index in IAxisVMRebarSteelGrades)*
long	**StirrupMaterial**	*index of the stirrup material(index in AxisVMRebarSteelGrades)*
double	**StirrupDiameter**	*stirrup diameter [m]*
long	**StirrupLegs**	*number of stirrup legs*
ERCBeamShape	**Shape**	*shape of the beam*
double	**c_bottom**	*cover of bottom longitudal bars [m]*
double	**c_top**	*cover of top longitudal bars [m]*
double	**ds_bottom**	*diameter of bottom longitudal bars [m]*
double	**ds_top**	*diameter of top longitudal bars [m]*
ELongBoolean	**TakeConcTensileStrengthNL**	*take tensile strength of the concrete into account in nonlinear analysis*
ELongBoolean	**UsefctmfINL**	*consider flexural tensile strength in nonlinear analysis instead of tensile strength (only for EC and ITA)*
double	**ShrinkageEpsNL**	*shrinkage strain considered in nonlinear analysis [ ]*

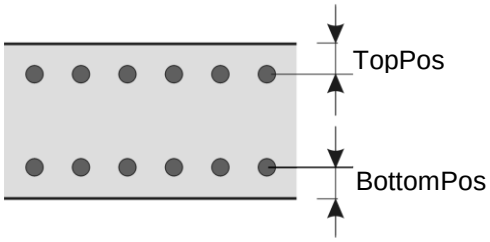
)
General RC beam reinforcement parameters .

RRCBeamDesignParameters_V173 = (

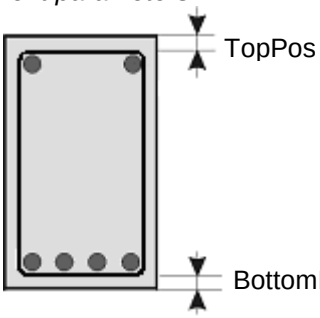
long	**ConcreteMaterial**	*index of the concrete material (index in IAxisVMMaterials)*
double	**Dmax**	*max. size of the aggregate [m]*
long	**RebarMaterial**	*index of the rebar material (index in IAxisVMRebarSteelGrades)*
long	**StirrupMaterial**	*index of the stirrup material(index in AxisVMRebarSteelGrades)*
double	**StirrupDiameter**	*stirrup diameter [m]*
long	**StirrupLegs**	*number of stirrup legs*
ERCBeamShape	**Shape**	*shape of the beam*
double	**c_bottom**	*cover of bottom longitudal bars [m]*
double	**c_top**	*cover of top longitudal bars [m]*
double	**ds_bottom**	*diameter of bottom longitudal bars [m]*
double	**ds_top**	*diameter of top longitudal bars [m]*
double	**ds_torsion**	*diameter of torsional bars [m]*
long	**MaxAppliedRebarCount_Top**	*maximum number of applied top longitudal bars*
long	**MaxAppliedRebarCount_Bottom**	*maximum number of applied bottom longitudal bars*
long	**MaxAppliedStirrupCount**	*maximum number of applied lateral longitudal bars*
double	**StirrupSpacingStep**	*grid for stirrup positioning. Set it to 0 for grid free positioning*
ELongBoolean	**ActualReinforcementDef**	*actual reinforcement is definded when this is set to lbTrue*
double	**ActualDiam_Top**	*actual reinforcement diameter for top longitudal bars [m] (only if ActualReinforcementDef = lbTrue)*
double	**ActualDiam_Bottom**	*actual reinforcement diameter for bottom longitudal bars [m] (only if ActualReinforcementDef = lbTrue)*
long	**ActualCount_Top**	*number of actual top longitudal bars (only if ActualReinforcementDef = lbTrue)*
long	**ActualCount_Bottom**	*number of actual bottom longitudal bars (only if ActualReinforcementDef = lbTrue)*
ELongBoolean	**TakeConcTensileStrengthNL**	*take tensile strength of the concrete into account in nonlinear analysis*
ELongBoolean	**UsefctmfINL**	*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 = (

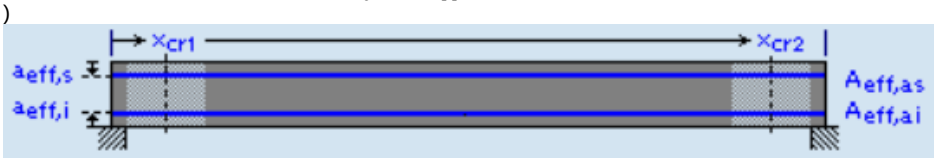
ELongBoolean	**Active**	*platic hinge is active*
ELongBoolean	**AppliedReinforcement**	*use applied reinforcement*
double	**As_Bottom**	*area of reinforcement at the bottom [m^2]*
double	**As_Top**	*area of reinforcement at the top [m^2]*
double	**Depth_Bottom**	*depth of reinforcement at the bottom [m^2]*
double	**Depth_Top**	*depth of reinforcement at the top [m^2]*

)

RRCBeamPlasticHinges = (

ELongBoolean	**EnablePlasticHinges**	*consider plastic hinges*
RRCBeamPlasticHingeParams	**Hinge1**	*parameters of hinge 1*
RRCBeamPlasticHingeParams	**Hinge2**	*parameters of hinge 2*
double	**Pos_Hinge1**	*position of center of the hinge 1 [m]*
double	**Pos_Hinge2**	*position of center of the hinge 2 [m]*
double	**MinRebarDiameter**	*minimal diameter among the considered rebars [m]*
double	**GammaRd**	*safety factor [ ]*

)



RRCBeamDesignParameters_EC = (

ELongBoolean	**VariableAngleTrussMethod**	*variable or fixed (at 45 deg.) strut angle*
double	**Theta**	*angle of the concrete compression strut Θ (in rads.), must be set if VariableAngleTrussMethod = lbFalse [rad]*
double	**fse**	*load factor for seismic forces (default is 1 =>no change) [ ]*
ELongBoolean	**ApplyMinimumCover**	*Calculate concrete covers to all reinforcements based on enviroment and structural class*
ELongBoolean	**CrackWidthCheck**	*Increase reinforcement to limit crack width to MaxCrackWidth_Bottom and MaxCrackWidth_Top values*
double	**MaxCrackWidth_Bottom**	*max. allowed crack width at top [m]*
double	**MaxCrackWidth_Top**	*max. allowed crack width at bottom [m]*
ELongBoolean	**TakeConcTensileStrength**	*Take tensile strength of the concrete into account*
ELongBoolean	**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 [ ]*
EEnvironmentClass	**TopSurface**	*environment class at top*
EEnvironmentClass	**BottomSurface**	*environment class at bottom*
EStructClass_EC	**StructClass**	*structural class*
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 Euro code.

RRCBeamDesignParameters_EC_RO = (

double	**Ksi_RO**	*$\xi_{c0} = x_0 / d$, 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) [ ]*
ELongBoolean	**VariableAngleTrussMethod**	*variable or fixed (at 45 deg.) strut angle*
double	**Theta**	*angle of the concrete compression strut Θ (in rads.), must be set if VariableAngleTrussMethod = lbFalse [rad]*
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 [ ]*
ELongBoolean	**CrackWidthCheck**	*Increase reinforcement to limit crack width to MaxCrackWidth_Bottom and MaxCrackWidth_Top values*
double	**MaxCrackWidth_Bottom**	*max. allowed crack width at top [m]*
double	**MaxCrackWidth_Top**	*max. allowed crack width at bottom [m]*
ELongBoolean	**TakeConcTensileStrength**	*Take tensile strength of the concrete into account*
ELongBoolean	**ShortTerm**	*See RC beam design in AxisVM manual*
RRCBeamPlasticHinges	**PlasticHinges**	*parameters of plastic hinges at the ends of the beam*

)

Beam reinforcement parameters according to the Romanian Euro code.

RRCBeamDesignParameters_ITA = (

ELongBoolean	**VariableAngleTrussMethod**	*variable or fixed (at 45 deg.) strut angle*
double	**Theta**	*angle of the concrete compression strut Θ (in rads.), must be set if VariableAngleTrussMethod = lbFalse [rad]*
double	**fse**	*load factor for seismic forces (default is 1 =>no change) [ ]*
ELongBoolean	**CrackWidthCheck**	*Increase reinforcement to limit crack width to MaxCrackWidth_Bottom and MaxCrackWidth_Top values*
double	**MaxCrackWidth_Bottom**	*max. allowed crack width at top [m]*
double	**MaxCrackWidth_Top**	*max. allowed crack width at bottom [m]*
ELongBoolean	**TakeConcTensileStrength**	*Take tensile strength of the concrete into account*
ELongBoolean	**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 = (

ELongBoolean	**ApplyMinimumCover**	*Calculate concrete covers to all reinforcements based on enviroment and structural class*
EEnvironmentClass	**TopSurface**	*environment class at top*
EEnvironmentClass	**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 [ ]*
ELongBoolean	**VariableAngleTrussMethod**	*variable or fixed (at 45 deg.) strut angle*
double	**Theta**	*angle of the concrete compression strut Θ (in rads.), must be set if VariableAngleTrussMethod = lbFalse [rad]*
ELongBoolean	**CrackWidthCheck**	*Increase reinforcement to limit crack width to MaxCrackWidth_Bottom and MaxCrackWidth_Top values*
double	**MaxCrackWidth_Bottom**	*max. allowed crack width at top [m]*
double	**MaxCrackWidth_Top**	*max. allowed crack width at bottom [m]*
ELongBoolean	**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 = (

Warning! This record has become obsolete.

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 = *(*

Warning! This record has become obsolete.

EEnvironmentClass	**EnvironmentClass**	*environment class*
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*
ELongBoolean	**VariableAngleTrussMethod**	*variable or fixed (at 45 deg.) strut angle*
double	**Theta**	*angle of the concrete compression strut Θ (in rads.), must be set if VariableAngleTrussMethod = lbFalse*
ELongBoolean	**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*
ELongBoolean	**TakeConcTensileStrength**	*Take tensile strength of the concrete into account*

)

Beam reinforcement parameters according to the German design code DIN.

RRCBeamDesignParameters_MSZ = *(*

Warning! This record has become obsolete.

ELongBoolean	**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*
ELongBoolean	**TakeConcTensileStrength**	*Take tensile strength of the concrete into account*
ELongBoolean	**AutoPsi**	*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*

)

Beam reinforcement parameters according to the Hungarian design code MSZ.

RRCBeamDesignReqReinfs = (

RRCBeamDesignReqReinf	**Top**	*reinforcement at top required by design*
RRCBeamDesignReqReinf	**Bottom**	*reinforcement at bottom required by design*
RRCBeamDesignReqReinf	**Top_CC**	*reinforcement at top required to limit crack width (crack control)*
RRCBeamDesignReqReinf	**Bottom_CC**	*reinforcement at bottom required to limit crack width (crack control)*

)

Number of required bars at top and bottom of the cross-section

RRCBeamDesignReqReinf = (

long	**N**	*total number of bars*
long	**Nf**	*number of bars outside of web width (for T and I sections)*
long	**NRow**	*number of rows with rebar*
double	**U**	*distance of rebar's group center of gravity to section's edge [m]*

)

Number of rows and required bars and their position.

RRCBeamDesignCrackResults = (

RRCBeamDesignCrackResult	**Top**	*crack design results at top*
RRCBeamDesignCrackResult	**Bottom**	*crack design results at bottom*

)

Crack results at top and bottom of the cross-section

RRCBeamDesignCrackResult = (

Double	**Wk**	*crack width [m]*
Double	**I1**	*inertia of un-cracked cross section [m⁴]*
Double	**x1**	*distance of neutral axis of un-cracked cross section [m]*
Double	**I2**	*inertia of cracked cross section (according to elastic theory) [m⁴]*
Double	**x2**	*distance of neutral axis of cracked cross section (according to elastic theory) [m]*
Double	**Mcr**	*cracking moment [kNm]*
Double	**Sr_max**)	*maximum distance of cracks [m]*

Details of the crack results

RPartialRCBeamDesignParameters = (

RRCBeamCrossSections	**RRCBeamCrossSections**	*RC cross section at start and end*
RRCBeamSupports	**RRCBeamSupports**)	*RC beam support at start and end*

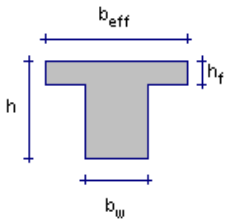
Partial RC Beam design parameters

RRCBeamCrossSections = (

RRCBeamSection	**StartSection**	*section parameter at the start*
RRCBeamSection	**EndSection**)	*section parameters at the end*

Start and end RC beam cross-section parameters

RRCBeamSection = (



double	**bw**	*breadth of the beam [m]*
double	**h**	*height of the section [m]*
double	**hf**	*flange thickness [m]*
double	**beff**)	*width of the flange [m]*

RC beam section parameters

RRCBeamSupports = (

RRCBeamSupport	**StartSupport**	*support parameters at the start*
RRCBeamSupport	**EndSupport**)	*support parameters at the end*

RC beam's supports

NOTE:
To enable/disable shear force reduction at supports set RRCBeamSupport record accordingly and call AddMembers function.

RRCBeamSupport = (

ELongBoolean	**OverWrite**	*if true, overwrite automatically generated values calculated by AxisVM depending on beam and column depths*
double	**ActualHalfWidth**	*actual half-width of the support (from axis to the edge of the support) [m]*
double	**TheoreticalHalfWidth**	*theoretical half-width of the support (from axis to the edge of the support) [m]*
ELongBoolean	**ShearReduction**)	*true if shear reduction enabled around supports. See picture*

Support parameters used for shear reduction

RRCBeamDesignResult = (

double	**x**	*x coordinate of design member [m]*
RRCBeamDesignBendingResult	**Top**	*RC beam bending design results at top*
RRCBeamDesignBendingResult	**Bottom**	*RC beam bending design results at bottom*
double	**s**	*spacing of shear links, s=min(s_v, s_{vmax}, s_t), construction criteria and torsion considered [m]*
double	**sv**	*spacing of shear links for shear design only [m]*
double	**smax**	*maximum spacing of shear links according to construction criteria [m]*
double	**st**	*spacing of shear links for torsion only [m]*
double	**Tsd**	*design torsion [kNm]*
double	**Ted_Trd**	*T_{ed} / T_{rd} utilisation (see national design code) [ ]*
double	**Ved_Vrd**	*V_{ed} / V_{rd} utilisation (see national design code) [ ]*
double	**Vsdred_Min**	*minimum reduced design shear force [kN]*
double	**Vsdred_Max**	*maximum reduced design shear force [kN]*
double	**VRdc**	*Shear resistance of concrete without shear reinforcement (for more information see $V_{Rd,c}$ in design code) [kN]*
double	**VRds**	*Shear resistance of shear reinforcement only (for more information see $V_{Rd,c}$ in design code) [kN]*
long	**VErrorMessage**	*RC beam shear design message code See here*
double	**Astor**)	*calculated area of longitudal reinforcement for torsion only [m^2]*

RC beam design results including shear and torsion reinforcement

RRCBeamDesignBendingResult = (

double	**M**	*design moment [kNm]*
double	**Ast**	*calculated area of tension reinforcement (required by design forces) [m^2]*
double	**Ast_min**	*calculated minimum area of tension reinforcement (required by design code ignoring forces) [m^2]*
double	**Asc**	*calculated area of compression reinforcement (required by design forces) [m^2]*
double	**Asc_min**	*calculated minimum area of compression reinforcement (required by design code ignoring forces) [m^2]*
double	**xc**	*depth of concrete section in compression [m]*
long	**MErrorMessage**)	*RC beam bending design message code See here*

RC beam flexural design resultsfrom bending only

RRCBeamDesignDeflectionResult = (

long	**CombinationOrLoadCaseID**	*Index of load combination or load case*
double	**ez**	*Vertical displacements / deflection [ ]*

Vertical displacements / deflection with corresponding load case or combination index

RRCBeamDesignDeflectionResults = (

RRCBeamDesignDeflectionResult	**Min**	*min. deflection result*
RRCBeamDesignDeflectionResult	**Max**	*max. deflection result*

Deflection results

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_beff, 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_beff, 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*
LoadCaseOrCombinationOrEnvelopeUID	*load case ,combination or EnvelopeUID index, ignored if 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***

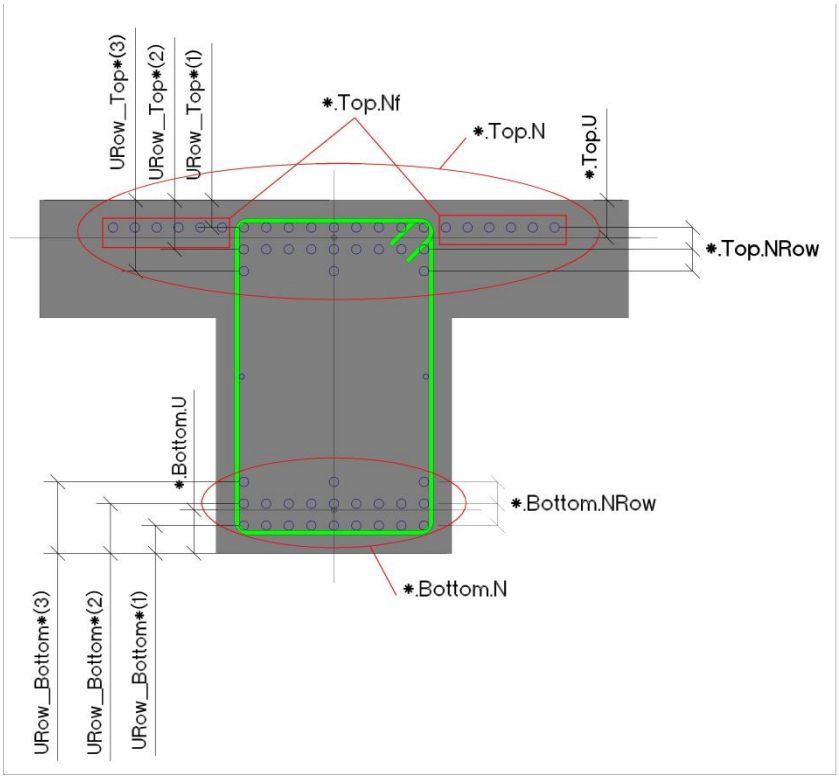
Returns number of results if successful, otherwise returns an error code (errDatabaseNotReady , rcbdeInvalidCombinationOfLoadCaseAndLoadLevel,rcbdeInvalidCombinationOfLoadCombinationAndLoadLevel, rcbdeInvalidLoadCaseId, rcbdeInvalidLoadCombinationId, rcbdeInvalidAnalysisType).

***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(2x6+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) = 0,0105 -> 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 edg

... 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**

Clears all results and setings from the interface.

Returns 1 if successful, otherwise returns an error code (errDatabaseNotReady)

---

long **GetDesignParameters** ([i/o] RRCBeamDesignParameters **RCBeamDesignParameters,**
[out] SAFEARRAY(byte) **DesignCodeParameters**)

Warning! *This function has become obsolete, it was superseded by GetDesignParameters_V173*

RCBeamDesignParameters	*RC beam design parameters*
DesignCodeParameters	*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).*

Returns 1 if successful, otherwise returns an error code (errDatabaseNotReady)

Record type depends on current **NationalDesignCode**

RRCBeamDesignParameters_DIN :
- *ndcGerman_DIN1045_1*

RRCBeamDesignParameters_EC:
- *ndcEuroCode, ndcEuroCode_GER , ndcEuroCode_Austrian, ndcEuroCode_UK, ndcEuroCode_NL , ndcEuroCode_FIN , ndcEuroCode_HU, ndcEuroCode_CZ , ndcEuroCode_B , ndcEuroCode_PL , ndcEuroCode_DK, ndcEuroCode_S*

RRCBeamDesignParameters_EC_RO :
- *ndcEuroCode_RO*

RRCBeamDesignParameters_ITA :
- *ndcItalian*

RRCBeamDesignParameters_MSZ :
- *ndcHungarian_MSZ*

RRCBeamDesignParameters_SIA :
- *ndcSwiss_SIA26x*

RRCBeamDesignParameters_STAS :
- *ndcRomanian_STAS*

---

long **GetDesignParameters_V173** ([i/o] RRCBeamDesignParameters_V173
RCBeamDesignParameters,[i/o] SAFEARRAY(byte) **DesignCodeParameters**)

RCBeamDesignParameters	*RC beam design parameters*
DesignCodeParameters	*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).*

Returns 1 if successful, otherwise returns an error code (errDatabaseNotReady)

Record type depends on current **NationalDesignCode**

RRCBeamDesignParameters_DIN :
- *ndcGerman_DIN1045_1*

RRCBeamDesignParameters_EC:
- *ndcEuroCode, ndcEuroCode_GER , ndcEuroCode_Austrian, ndcEuroCode_UK, ndcEuroCode_NL , ndcEuroCode_FIN , ndcEuroCode_HU, ndcEuroCode_CZ , ndcEuroCode_B , ndcEuroCode_PL , ndcEuroCode_DK, ndcEuroCode_S*

RRCBeamDesignParameters_EC_RO :
- *ndcEuroCode_RO*

RRCBeamDesignParameters_ITA :

- *ndcItalian*

RRCBeamDesignParameters_MSZ :

- *ndcHungarian_MSZ*

RRCBeamDesignParameters_SIA :

- *ndcSwiss_SIA26x*

RRCBeamDesignParameters_STAS :

- *ndcRomanian_STAS*

---

long **GetLines** ([out] SAFEARRAY(long) **LineIds**)

LineIds *array of line indexes for RC beam design*

Returns number of lines.

---

long **SetDesignParameters** ([i/o] RRCBeamDesignParameters **RCBeamDesignParameters,**[in] SAFEARRAY(byte) **DesignCodeParameters**)

Warning! *This function has become obsolete, it was superseded by SetDesignParameters_V173*

RCBeamDesignParameters *RC beam design parameters*

DesignCodeParameters *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)*

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)

---

long **SetDesignParameters_V173** ([i/o] RRCBeamDesignParameters_V173 **RCBeamDesignParameters,**[in] SAFEARRAY(byte) **DesignCodeParameters**)

RCBeamDesignParameters *RC beam design parameters*

DesignCodeParameters *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)*

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)

---

long **SetDesignParameters_vb** ([i/o] RRCBeamDesignParameters **RCBeamDesignParameters,**

[i/o] SAFEARRAY(byte) **DesignCodeParameters**)

Warning! *This function has become obsolete, it was superseded by SetDesignParameters_V173*

Visual Basic compatible function of **SetDesignParameters**

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** = {

rccceInvalidLineId = -100001	*Line index not valid*
rccceInvalidMemberId = -100002	*Member indexes not valid*
rccceInvalidLoadCaseId = -100003	*Load case index invalid*
rccceInvalidLoadCombinationId = -100004	*Load combination index invalid*
rccceInvalidCombinationOfLoadCaseAndLoadLevel = -100005	*Invalid combination of load case and load level*
rccceInvalidCombinationOfLoadCombinationAndLoadLevel = -100006	*Invalid combination of load combination and load level*
rccceColumnReinforcementParametersNotSet = -100007	*Column reinforcement parameters not set*
rccceCapacityCurveCannotBeGenerated = -100008	*Capacity curve can not be generated*
rccceInvalidRebarSteelGradeId = -100009	*Rebar steel grade is invalid*
rccceInvalidConcreteMaterialId = -100010	*Concrete material index is invalid*
rccceInvalidColumnRebarsId = -100011	*Column rebar index is invalid*
rccceCapacityCurveNotYetGenerated = -100012	*Capacity curve has not yet been generated*
rccceDifferentColumnReinforcementParameters = -100013	*Column reinforcement parameters vary*
rccceInvalidArrayLength = -100014	*Invalid length of array*
rccceInvalidAnalysisType = -100015	*Invalid type of analysis*
rccceExcentricity = -100016	*Invalid eccentricity*
rccceColumnReinforcementNoForces= -100017	*Forces for column checking are not available*
rccceInvalidEnvelopeID = -100018	*Invalid EnvelopeID*
rccceInvalidCheckingParameters = -100019	*at least one of the checking parameters is invalid*
rccceShrinkageEpsMustBePositive = -100020	*shrinkage strain must be positive*
rccceVTCheckIsNotSupported = -100021	*shear/torsion check is not supported*
rccceStirrupParametersAreInvalid = -100022	*stirrup parameters are invalid*
rccceShearCrackAngleIsInvalid = -100023	*defined shear crack angle is too low/high*
rccceVTCapacityDesignIsNotSupported = -100024	*capacity design is not supported*
rccceInvalidSteelMaterialId = -100025	*steel grade is invalid*
rccceColumnCheckingIsNotSupported = -100026	*Column checking is not supported (e.g. composite section)*
rccceMixedDesignSituation = -100027	*Variable design situations are included into a single design check*
rccceErrorInFireDesignCalculation = -100028 }	*Error has occurred in fire design calculations (e.g. no fire load has been defined, fire load parameters are variable along a single element, fire design is not supported)*

Records / structures

RMyMz = (

double	**My**	*My bending moment about local y axis [kNm]*
double	**Mz**)	*Mz bending moment about local z axis [kNm]*

Bending moments in the RC column

RMyMzFi = (

double	**My**	*My bending moment about local y axis [kNm]*
double	**Mz**	*Mz bending moment about local z axis [kNm]*
double	**fi**)	*rotation angle of neutral axis [rad]*

Bending moments with fi in the RC column

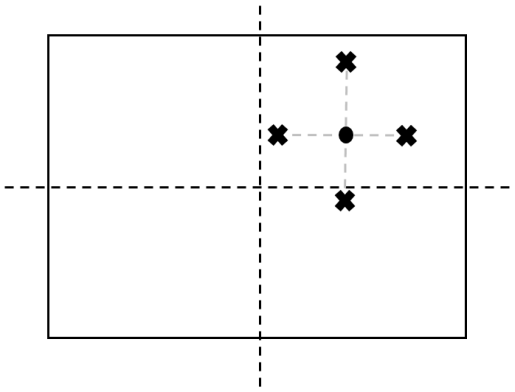
RNMInteractionDiagMinMax = (

double	**NMin**	*min. axial force [kN]*
double	**NMax**	*max. axial force [kN]*
double	**MyMin**	*min. My bending moment about local y axis [kNm]*
double	**MyMax**	*max. My bending moment about local y axis [kNm]*
double	**MzMin**	*min. Mz bending moment about local z axis [kNm]*
double	**MzMax**	*max. Mz bending moment about local z axis [kNm]*

Min. and max. forces in the RC column

RColumnCheckResult =(

double	**xRelPos**	*relative position of the cross section*
ELongBoolean	**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 = const.)*
double	**Eff_Const_e**	*Column's efficiency (M/N = e = 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*
ELongBoolean	**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] RColumnReinforcementParameters **Parameters,** [in] SAFEARRAY(double), **xPos,** [in] SAFEARRAY(RColumnForces) **ColumnForces,**
[out] SAFEARRAY(RColumnCheckResult) **ColumnCheckResults**)

Parameters	*RC column reinforcement parameters*
xPos	*array of positions along the column where check is performed*
ColumnForces	*internal forces on the RC column related to xPos array*
ColumnCheckResults	*check forces on RC column on top, bottom and centre*

If successful returns number of RC column check results, otherwise an error code (errDatabaseNotReady, rccceExcentricity, errCOMServerInternalError, rccceColumnReinforcementParametersNotSet, rccceCapacityCurveCannotBeGenerated, rccceInvalidRebarSteelGradeId, rccceInvalidConcreteMaterialId, rccceInvalidColumnRebarsId, rccceInvalidArrayLength).

---

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(RColumnCheckResult) **ColumnCheckResults**,
[out] SAFEARRAY(BSTR) **Combinations**)

MemberIds	*index of the members*
ResultType	*Type of result*
LoadCaseOrCombinationOrEnvelopeUID	*load case, load combination or unique envelope index*
LoadLevel	*load level (increment) index*
AnalysisType	*Type of Analysis*
Creep	*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…*
ColumnCheckResults	*check forces on RC column* *Note: xRelPos returns always with 0*
Combinations	*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*

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).

---

long **CheckMembers_vb** (Visual Basic compatible function of **CheckMembers**)

---

long **Clear**

Clears all results and setings from the interface.

Returns 1 if successful, otherwise returns an error code (errDatabaseNotReady)

---

long **CheckVTByParameters** ([i/o] RColumnReinforcementParameters **Parameters,** [in] SAFEARRAY(double), **xPos,** [in] SAFEARRAY(RColumnForces) **ColumnForces,**
[out] SAFEARRAY(RColumnVTCheckResult) **ColumnVTCheckResults**)

Parameters	*RC column reinforcement parameters*
xPos	*array of positions along the column where check is performed*
ColumnForces	*internal forces on the RC column related to xPos array*
ColumnVTCheckResults	*results of shear and torsion checks*

If successful returns number of RC column shear/torsion check results, otherwise an error code (errDatabaseNotReady, rccceExcentricity, errCOMServerInternalError, rccceColumnReinforcementParametersNotSet, rccceCapacityCurveCannotBeGenerated, rccceInvalidRebarSteelGradeId, rccceInvalidConcreteMaterialId, rccceInvalidColumnRebarsId, rccceInvalidArrayLength, rccceVTCheckIsNotSupported, rccceStirrupParametersAreInvalid, rccceShearCrackAngleIsInvalid).

---

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(RColumnVTCheckResult) **ColumnVTCheckResults**)

Parameters	*RC column reinforcement parameters*
xPos	*array of positions along the column where check is performed*
ColumnForces	*internal forces on the RC column related to xPos array*
EQColumnForces	*internal forces from seismic effect only on the RC column related to xPos array*
ColumnVTCheckResults	*results of shear and torsion checks*

If successful returns number of RC column shear/torsion check results, otherwise an error code (errDatabaseNotReady, rccceExcentricity, errCOMServerInternalError, rccceColumnReinforcementParametersNotSet, rccceCapacityCurveCannotBeGenerated, rccceInvalidRebarSteelGradeId, rccceInvalidConcreteMaterialId, rccceInvalidColumnRebarsId, rccceInvalidArrayLength, rccceVTCheckIsNotSupported, rccceStirrupParametersAreInvalid, rccceShearCrackAngleIsInvalid, rccceVTCapacityDesignIsNotSupported).

---

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] ELongBoolean **Creep,**
[out] SAFEARRAY(RColumnVTCheckResult) **ColumnVTCheckResults**)

MemberIds	*index of the members*
ResultType	*Type of result*
LoadCaseOrCombinationOrEnvelopeUID	*load case, load combination or unique envelope index*
LoadLevel	*load level (increment) index*
AnalysisType	*Type of Analysis*
Creep	*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…*
ColumnForces	*internal forces on the RC column*
ColumnVTCheckResults	*results of shear and torsion checks Note: xRelPos returns always with 0*

If successful returns number of RC column shear/torsion check results, otherwise an error code (errDatabaseNotReady, rccceInvalidCombinationOfLoadCaseAndLoadLevel, rccceInvalidCombinationOfLoadCombinationAndLoadLevel, rccceInvalidLoadCaseId, rccceInvalidLoadCombinationId, rccceDifferentColumnReinforcementParameters, rccceColumnReinforcementParametersNotSet, errCOMServerInternalError, rccceCapacityCurveCannotBeGenerated, rccceColumnReinforcementParametersNotSet, rccceInvalidArrayLength, rccceInvalidAnalysisType, rccceVTCheckIsNotSupported, rccceStirrupParametersAreInvalid, rccceShearCrackAngleIsInvalid).

long **CheckVTMembers_vb** (Visual Basic compatible function of **CheckVTMembers**)

long **GenererateNMInteractionDiagByLine** ([in] long **LineId,** [out] SAFEARRAY(double) **N,** [out] SAFEARRAY(long) **ItemCounts**, [out] SAFEARRAY(RMyMzFi) * **MyMzFi**)

LineId *index of the line*
N *array with axial forces*
ItemCounts *array with number of moment combinations*
MyMzFi *array with moment combinations*

Returns number of generated N-M interaction diagrams if successful, otherwise returns an error code (errDatabaseNotReady , rccceCapacityCurveCannotBeGenerated, rccceColumnReinforcementParametersNotSet,rccceInvalidLineId).

---

long **GenererateNMInteractionDiagByMember** ([in] long **MemberId,** [out] SAFEARRAY(double) **N,** [out] SAFEARRAY(long) **ItemCounts**, [out] SAFEARRAY(RMyMzFi) * **MyMzFi**)

MemberId *index of the member*
N *array with axial forces*
ItemCounts *array with number of MY,Mz combinations*
MyMzFi *array with moment combinations*

Returns number of generated N-M interaction diagrams if successful, otherwise returns an error code (errDatabaseNotReady , rccceCapacityCurveCannotBeGenerated, rccceColumnReinforcementParametersNotSet, rccceInvalidMemberId, errInternalException).

---

long **GenererateNMInteractionDiagByParameters** ([i/o] RColumnReinforcementParameters **Parameters,** [out] SAFEARRAY(double) **N,** [out] SAFEARRAY(long) **ItemCounts**, [out] SAFEARRAY(RMyMzFi) * **MyMzFi**)

Parameters *RC column reinforcement parameters*
N *array with axial forces*
ItemCounts *array with number of moment combinations*
MyMzFi *array with moment combinations*

Returns number of moment combinations if successful, otherwise returns an error code (errDatabaseNotReady, rccceCapacityCurveCannotBeGenerated, rccceInvalidRebarSteelGradeId, rccceInvalidConcreteMaterialId, rccceInvalidColumnRebarsId).

---

long **GetMyMzPolyByNx** ([in] double **Nx,** [out] SAFEARRAY(RMyMz) **Poly**)

Nx *axial force*
Poly *polygon with moment combination My - Mz*

Returns number of moment combinations if successful, otherwise returns an error code (rccceCapacityCurveNotYetGenerated).

---

long **GetNMInteractionDiagMinMax** ([i/o] RNMInteractionDiagMinMax **NMInteractionDiagMinMax**)

NMInteractionDiagMinMax *polygon with moment combination My - Mz*

Returns 1 if successful, otherwise returns an error code (rccceCapacityCurveNotYetGenerated).

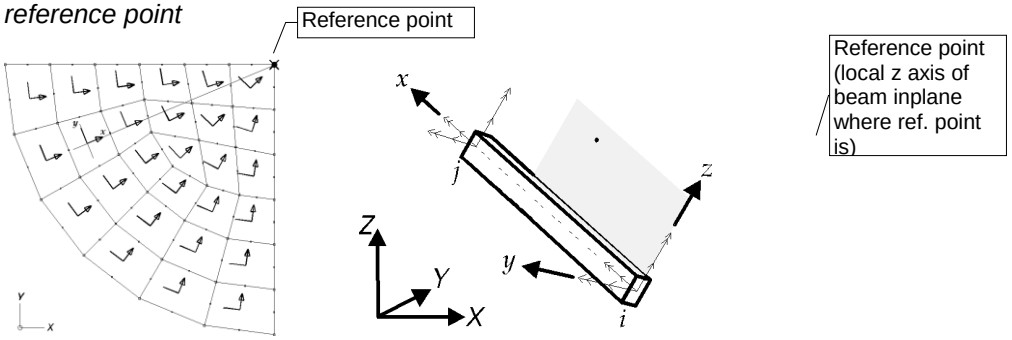
IAxisVMReferences

References in the model

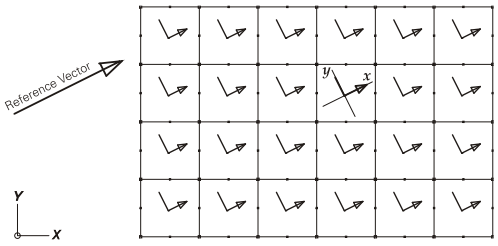
Enumerated types

enum **EReferenceType** = {

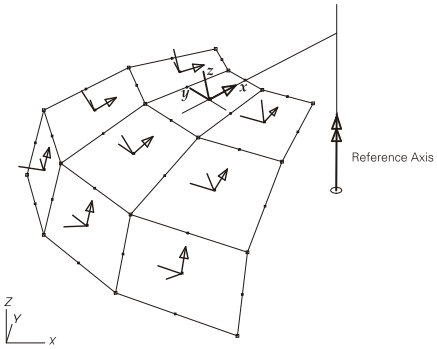
rtPoint = 0x01, *reference point*



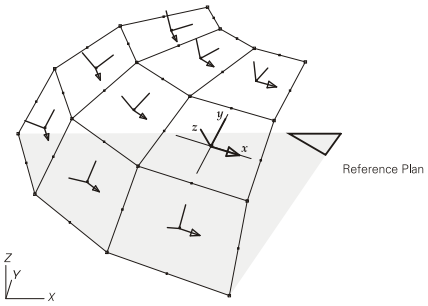
rtVector = 0x02, *reference vector*



rtAxis = 0x03, *reference axis*



rtPlane = 0x04, *reference plane*



rtBeta = 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 [m]*

)

RRefVector = (

RPoint3d	**P1**	*first point of the reference vector [m]*
RPoint3d	**P2**	*second point of the reference vector [m]*

)

RRefAxis = (

RPoint3d	**P1**	*first point of the reference axis [m]*
RPoint3d	**P2**	*second point of the reference axis [m]*

)

RRefPlane = (

RPoint3d	**P1**	*first point of the reference plane [m]*
RPoint3d	**P2**	*second point of the reference plane [m]*
RPoint3d	**P3**	*third point of the reference plane [m]*

)

RRefBeta = (

double	**Beta**	*reference angle [rad]*

)

RRefData = (

RRefPoint	**Point**	*reference point [m] (rtPoint)*
RRefVector	**Vector**	*reference vector [m] (rtVector)*
RRefAxis	**Axis**	*reference axis [m] (rtAxis)*
RRefPlane	**Plane**	*reference plane [m] (rtPlane)*
RRefBeta	**Beta**	*reference angle [rad] (rtBeta)*

)

RReference = (

EReferenceType	**RefenceType**	*reference type*
RRefData	**ReferenceData**	*reference data*

)

Functions

long **Add** ([i/o] RReference **Item**)

Item *the reference*

Adds a reference to the model. If a similar reference already exists, it won't create a new one, but returns the index of the already existing one. 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] ELongBoolean **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).

NOTE: 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*

ELongBoolean **Selected** [long **Index**] • *Get or set the selection status of a reference*

NOTE: 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 < DrawingIndex ≤ IAxisVMDrawingsLibrary.Count*

Adds drawing from library to the 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 ≤ Index ≤ 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 ≤ Index ≤ 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 *index of the report*

FolderPath *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.

ImageIds *array of image indices*

Returns with image indices situated in a given folder of a report wiith ReportIndex. 1 ≤ Index ≤ Count. If successful, returns the number of images otherwise returns an error code (see EGeneralErrors).

---

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

Name *name of the report*

get index of the report by name.If successful, returns Index otherwise returns an error code (see EGeneralErrors).

---

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

Name *name of the report*

FileName *name of the template file, .rep extension*

Creates new report based on template file. If successful, returns the report index otherwise an error code (see EGeneralErrors).

Properties

long **Count** *Get number of reports in the model*

long **ImageCount** [long **Index**]
Index - *report index*
Get number of images in the report

BSTR **ImagePath** [long **ReportIndex,** long **ImageIndex**]
ReportIndex - *report index*
ImageIndex - *image index*
Get the path with caption of an image in the report

BSTR **ImageCaption** [long **ReportIndex,** long **ImageIndex**]
ReportIndex - *report index*
ImageIndex - *image index*
Get the caption of an image in the report

BSTR **Name** [long **Index**] • *Get or set the name of the report*

BSTR **TemplateDescription** [long **Index**] • *Get or set the description of the template used with the report*

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].

linear analysis	*nonlinear analysis*	*vibration analysis*	*buckling analysis*	*dynamic analysis*
[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

LoadCaseId[atNonlinearStatic, 1] = 3
LoadCaseId[atNonlinearStatic, 2] = 5
LoadCaseId[atVibration, 1] = 2
LoadCaseId[atVibration, 2] = 1
LoadCaseId[atBuckling, 1] = 3
LoadCaseId[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	*invalid analysis type*
reResultCaseIndexOutOfBounds = –100002	*result block index is out of bounds*
reResultCaseIsNotLoadCase = –100003	*result block belongs to a load combination*
reResultCaseIsNotLoadCombination = –100004	*result block belongs to a load case*
reLoadCaseIsOutOfBounds = –100005	*load case index is out of bounds*
reLoadCombinationIsOutOfBounds = –100006	*load combination index is out of bounds*
reFrequencyIndexOutOfBounds = –100007	*frequency index is out of bounds*
reModeShapeIndexOutOfBounds = –100008	*mode shape index is out of bounds*
reLoadLevelIndexOutOfBounds = –100009	*load level index is out of bounds*
reInvalidArrayLength = –100010	*length of the array is not the expected value*
reInvalidLoadCaseType = –100011	*invalid load case type*
reInvalidNationalDesignCode = –100012	*Invalid national design code*
reInvalidResponseSpectraParam = –100013	*Invalid response spectrum parameters*
reITAReductionCriterionNotSatisfied = –100014	*ITA reduction criterion not satisfied*
reSteelDesignResultsDisabled = –100015	*extension module SD1 is not available*
reCalculatedReinforcementDisabled = –100016	*extension module RC1 is not available*
reReinforcementCheckDisabled = –100017	*When used design code doesn't support reinforcement checks or extension module RC1 is not available*
reMissingAnalysisResults = –100018	*Results are not available run the analysis first*
reRC3moduleNotAvailable = –100019	*extension module RC3 is not available*
reRC1moduleNotAvailable= –100020	*extension module RC1 is not available*
reTD1moduleNotAvailable= –100021	*extension module TD1 is not available*
reDYNmoduleNotAvailable = –100022	*extension module DYN is not available*
reSE2moduleNotAvailable= –100023	*extension module SE2 is not available*
reNLpackageNotAvailable = –100024	*extension package NL (non-linear) is not available*
rePushoverSpectrumIsNotValid = –100025	*pushover spectrum is invalid or not available*
rePushoverSpectrumIsNotParametric = –100026	*only parametric spectrum is supported for pushover*

}

Enumerated types

enum **EAnalysisType** = {

atLinearStatic = 0x00	*linear analysis*
atNonLinearStatic = 0x01	*nonlinear analysis*
atLinearVibration = 0x02	*linear vibration analysis*
atNonLinearVibration = 0x03	*nonlinear vibration analysis*
atBuckling = 0x04	*buckling analysis*
atDynamic = 0x05	*dynamic analysis*
atNone = 0x06 }	*no specified analysis*

Type of the analysis

enum **EMinMaxType** = {

mtMin = 0x01,	*minimum values*
mtMax = 0x02,	*maximum values*
mtMinMax= 0x03 }	*used only in display parameters (IAxisVMWindows and IAxisVMWindow interfaces)*

Identifies minimum or maximum values for envelope and critical results when reading results.

Alternatively to set display settings in the window(s).

enum **ESurfaceVertexType** = {

svtContourPoint = 0x00,	*surface polygon vertex (node)*
svtContourLineMidPoint = 0x01,	*edge midpoint*
svtCenterPoint = 0x02 }	*surface center point*

Identifies a surface result location

enum **EXYchartFillType** = {

xycftNone = 0x0,	*dataset line only*
xycftSolid = 0x1,	*dataset line with fill*
xycftGradient = 0x2 }	*dataset line with fill colour changing with gradient*

Used for showing the dataset in chart

enum **EXYchartLabelingStyle** = {

xyclsArrange = 0x0,	*labels are rearranged in order show most of them*
xyclsOverlapFilter = 0x1 }	*labels are filtered in order to prioritize important labels*

Used when showing labels in chart

Records / structures

RFrequencyParameters = (

double	**Frequency**	*frequency*
double	**EigenValConv**	*eigenvalue error*
double	**EigenVecConv**	*eigenvector error*
double	**EigenValConvLimit**	*convergence tolerance for the eigenvalue*
double	**EigenVecConvLimit**	*convergence tolerance for the eigenvector*
long	**Iteration**	*number of actual iterations completed*

)

RNcrParameters = (

double	**Ncr**	*critical load parameter*
double	**EigenValConv**	*eigenvalue error*
double	**EigenVecConv**	*eigenvector error*
double	**EigenValConvLimit**	*convergence tolerance for the eigenvalue*
double	**EigenVecConvLimit**	*convergence tolerance for the eigenvector*
long	**Iteration**	*number of actual iterations completed*

)

RNonLinearAnalysisResultInfo = (

long	**Increments**	*number of increments (load levels)*
long	**Iterations**	*number of actual iterations completed*
double	**Lambda**	*load factor of a load level ($0 < Lambda \leq 1$)*
double	**ErrorP**	*relative error of the force convergence*
double	**ErrorU**	*relative error of the displacement convergence*
double	**ErrorE**	*relative error of the work convergence*
double	**ErrorEq**	*solution error*
double	**ControlVal**	*control parameter value*
ELongBoolean	**Convergence**	*if True, the iteration has converged*

)

RResultBlockInfo = (

long	**ResultCase**	*result case index*
long	**LoadLevelOrModeShapeOrTimeStep**	*load level or mode shape or time step index of the result block within the result case. The default value must be 1, unless it is for a case listed below:* *for load level:* *$(0 < LoadLevelOrModeShapeOrTimeStep \leq$ LoadLevelCount* *for mode shape:* *$0 < LoadLevelOrModeShapeOrTimeStep \leq$ ModeShapeCount* *for time step:* *$0 < LoadLevelOrModeShapeOrTimeStep \leq$ TimeStepCount*

)

RSeismicEq = (

double	**seEpsX**	*seismic equivalent coefficient in global x direction (% of participated mass)*
double	**seEpsY**	*seismic equivalent coefficient in global y direction (% of participated mass)*
double	**seEpsZ**	*seismic equivalent coefficient in global z direction (% of participated mass)*

)

RGlobalForces = (

double	**Fx, Fy, Fz**	*x, y, z load components in global directions [kN]*
double	**Mx, My, Mz**	*moment components about the global x, y, z axis [kNm]*

)

RTotalLoads = (

RGlobalForces	**ExternalForces**	*sum of applied loads in all global directions*
RGlobalForces	**Unbalancedloads**	*sum of unbalanced loads in all global directions*

)

JSON strings

OptionsJSON= (

string	**Description**	*Description saved in metafile*
string	**Created by**	*Authors name saved in metafile*
long	**Width**	*width of the metafile in pixels*
long	**Height**	*height of the metafile in pixels*
string	**Main title**	*Main title shown above the chart*
string	**Subtitle**	*Subtitle shown below the chart*
boolean	**Show title**	*Determines whether main title and subtitle are shown*
integer(EGeneralAlignmentHorizontal)	**Label align**	*horizontal align of the labels*
boolean	**Show tick marks**	*Determines whether show tick marks on axis X and Y*
boolean	**Show arrows**	*Determines whether show arrows on axis X and Y*
integer	**Axes width**	*Determines size of arrows on axis X and Y*
string	**Font name**	*name of the used font, must be available in windows*
integer	**Font size**	*size of the used font*
double	**Left margin**	*proportional margin at left (0 < margin < 0,5)*
double	**Rightmargin**	*proportional margin at right (0 < margin < 0,5)*
double	**Top margin**	*proportional margin at top (0 < margin < 0,5)*
double	**Bottommargin**	*proportional margin at bottom (0 < margin < 0,5)*
string	**Axis title X**	*Title of axis x*
string	**Axis title Y**	*Title of axis y*
double	**Axis elongation X**	*proportional (x values) elongation of x axis*
double	**Axis elongation Y**	*proportional (x values) elongation of y axis*
boolean	**Max. labeling preffered**	*IF True,labelling of larger y values is prefered*
boolean	**Tex labels**	*Label strings can contain TeX format strings*
boolean	**Show more labels**	*Additional labels*
integer(EXYchartLabelingStyle)	**Labeling style**	*Style of labelling,length must be same as number of datasets*
array of integer(EXYchartFillType)	**Fill type**	*Style of dataset,length must be same as number of datasets*
array of boolean	**Show labels**	*whether show label,length must be same as number of datasets*
array of boolean	**Absolute labels**	*whether label are absolute values,length must be same as number of datasets*
array of boolean	**Thick dataset**	*Dataset with thick line,length must be same as number of datasets*
array of integer	**Dataset colour**	*Colours of datasets,length must be same as number of datasets*
array of integer	**Fill colour**	*Colours of dataset fills,length must be same as number of datasets*
array of integer	**Label colour**	*Labels of datasets,length must be same as number of datasets*
array of string	**Dataset titles**	*Titles of datasets,length must be same as number of datasets. Empty strings are not shown in the legend*
boolean	**Show legend**	*Determines whether show the legend*
integer(EGeneralAlignmentHorizontal)	**Legend align horizontal**	*Determines horizontal alignment of the legend*
integer(EGeneralAlignmentVertical)	**Legend align vertical**	*Determines vertical alignment of the legend*
boolean	**Show grid**	*Determines whether show the grid*
boolean	**Frame**	*Determines whether show frame around the chart*
double	**Tick spacing X**	*Determines the spacing of tick on axis x*
double	**Tick spacing Y**	*Determines the spacing of tick on axis y*

)

PropertiesJSON= (

integer	**Origin X**	*horizontal pixel coordinate of the origin (0,0)*
integer	**Origin Y**	*vertical pixel coordinate of the origin (0,0)*
double	**Scale X**	*horizontal scale (pixels per unit)*
double	**Scale Y**	*vertical scale (pixels per unit)*
array of double	**X of labels**	*x coordinates (units) of labels*
array of integer	**Dataset IDs of labels**	*dataset index (starts with 1) of shown labels*
array of integer	**Point IDs of labels**	*indexes of the points (starts with 1) in datasets of shown labels*

)

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 < ResultCase ≤ ResultCaseCount[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 < ResultCase ≤ ResultCaseCount[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 < ResultCase ≤ ResultCaseCount[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 < ResultCase ≤ ResultCaseCount[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 < ResultCase ≤ ResultCaseCount[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**)

Warning! *This function has become obsolete, it was superseded by GetBucklingParameters_V162*

ResultCase	*result case index (0 < ResultCase ≤ ResultCaseCount[atBuckling])*
BucklingParameters	*parameters of the buckling analysis*

Retrieves parameters of a buckling analysis.
Possible error code: reResultCaseIndexOutOfBounds or reNLpackageNotAvailable

---

long **GetBucklingParameters_V162** ([in] long **ResultCase,**
[i/o] RBuckling_V162 **BucklingParameters**)

ResultCase	*result case index (0 < ResultCase ≤ 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 **LoadCaseId,**
[in] RSpectrumData **SpectrumData,** [out] SAFEARRAY(double)* **X,**
[out] SAFEARRAY(double)* **Y**)

CapacityCurveType	*Type of capacity curve*
LoadCaseId	*load case index (0 < LoadCaseId ≤ 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, reLoadCaseIsOutOfBounds, reInvalidNationalDesignCode, reSE2moduleNotAvailable)

PLEASE NOTE:

Number of points depends on number of increments set in RPushOverAnalysis.

---

long **GetCapacityCurvePushOver** ([in] ECapacityCurveType **CapacityCurveType,** [in] long **LoadCaseId,** [in] RSpectrumData **SpectrumData,** [out] SAFEARRAY(double)* **X,**
[out] SAFEARRAY(double)* **Y**)

CapacityCurveType	*Type of capacity curve*
LoadCaseId	*load case index (0 < LoadCaseId ≤ 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, reLoadCaseIsOutOfBounds, reInvalidNationalDesignCode, reSE2moduleNotAvailable)

PLEASE NOTE:

Number of points depends on number of increments set in RPushOverAnalysis.

---

long **GetDynamicParameters** ([in] long **ResultCase,**
[i/o] RDynamicAnalysis_V162 **DynamicParameters**)

ResultCase	*result case index (0 < ResultCase ≤ ResultCaseCount[atDynamic])*
DynamicParameters	*parameters of the dynamic analysis*

Retrieves parameters of a dynamic analysis if successful, otherwise an error code (reResultCaseIndexOutOfBounds or reDYNmoduleNotAvailable).

---

long **GetFrequencyParameters** ([in] EAnalysisType **AnalysisType,** [in] long **ResultCase,** [in] long **FrequencyId,** [i/o] RFrequenyParameters **FrequencyParameters**)

AnalysisType	*analysis type* *must be atLinearVibration or atNonLinearVibration*
ResultCase	*result case index* *(0 < ResultCase ≤ ResultCaseCount[AnalysisType])*
FrequencyId	*mode shape (frequency) index* *(0 < FrequencyId ≤ ModeShapeCount[AnalysisType,ResultCase])*
FrequencyParameters	*parameters of the vibration analysis*

Retrieves parameters of a vibration analysis of a given shape.
If the iteration was successful, FrequencyParameters.EigenValConv < FrequencyParameters.EigenValConvLimit,FrequencyParameters.EigenVecConv < FrequencyParameters.EigenVecConvLimit.Error codes are reInvalidAnalysisType, reFrequencyIndexOutOfBounds and reResultCaseIndexOutOfBounds.

---

long **GetNcrParameters** ([in] long **ResultBlock,** [in] long **ModeShapeId,**
[i/o] RNcrParameters **NcrParameters**)

ResultCase	*result case index* *(0 < ResultBlock ≤ ResultBlockCount[atBuckling])*
ModeShapeId	*mode shape (frequency) index* *(0 < ModeShapeId ≤ ModeShapeCount[atBuckling,ResultBlock])*
NcrParameters	*parameters of the buckling analysis*

Retrieves parameters of a buckling analysis of a given buckling shape.
If the iteration was successful,
NcrParameters.EigenValConv < NcrParameters.EigenValConvLimit,
NcrParameters.EigenVecConv < NcrParameters.EigenVecConvLimit.
Error codes are reNLpackageNotAvailable, reModeShapeIndexOutOfBounds or reResultCaseIndexOutOfBounds.

---

long **GetNonLinearAnalysisParameters** ([in] long **ResultCase,**
[i/o] RNonLinearAnalysis **NonLinearAnalysisParameters**)

Warning! *This function has become obsolete, it was superseded by GetNonLinearAnalysisParameters_V162*

ResultCase	*result case index* *(0 < ResultCase ≤ ResultCaseCount[atNonLinearStatic])*
NonLinearAnalysisParameters	*parameters of the nonlinear analysis*

Retrieves parameters of a nonlinear analysis if successful, otherwise an error code (reResultCaseIndexOutOfBounds or reNLpackageNotAvailable).

---

long **GetNonLinearAnalysisParameters_V162** ([in] long **ResultCase,**
[i/o] RNonLinearAnalysis_V162 **NonLinearAnalysisParameters**)

ResultCase	*result case index* *(0 < ResultCase ≤ ResultCaseCount[atNonLinearStatic])*
NonLinearAnalysisParameters	*parameters of the nonlinear analysis*

Retrieves parameters of a nonlinear analysis if successful, otherwise an error code (reResultCaseIndexOutOfBounds or reNLpackageNotAvailable).

---

long **GetNonLinearAnalysisResultInfo** ([in] long **ResultCase,** [in] long **LoadLevel,**
[i/o] RNonLinearAnalysisResultInfo **NonLinearAnalysisResultInfo**)

ResultCase	*result case index* *(0 < ResultCase ≤ ResultCaseCount[atNonLinearStatic])*
LoadLevel	*load level (increment) index* *(0 < LoadLevel ≤ LoadLevelCount[atNonLinearStatic,ResultCase])*
NonLinearAnalysisResultInfo	*result information of a load level*

Retrieves result information for a load level of a nonlinear analysis if successful, otherwise an error code (reNLpackageNotAvailable, reLoadLevelIndexOutOfBounds or reResultCaseIndexOutOfBounds)

---

long **GetModeActive** ([in] EAnalysisType **AnalysisType,** [in] long **ResultCase,** [in] long **FrequencyId,** [out] ELongBoolean **Value**)

AnalysisType	*analysis type*
ResultCase	*result case index* *(0 < ResultCase ≤ ResultCaseCount[atLinearVibration or atNonLinearStatic])*
FrequencyId	*mode shape (frequency) index* *(0 < FrequencyId ≤ ModeShapeCount[AnalysisType,ResultCase])*
Value	*True if mass active for this mode shape (frequency)*

Returns FrequencyId if successful, otherwise an error code (reInvalidAnalysisType, errDatabaseNotReady, reFrequencyIndexOutOfBounds, reResultCaseIndexOutOfBounds)

---

long **GetNonLinearVibrationParameters** ([in] long **ResultCase,**
[i/o] RVibration **NonLinearVibrationParameters**)

Warning! *This function has become obsolete, it was superseded by GetNonLinearVibrationParameters_V162*

ResultCase	*result case index* *(0 < ResultCase ≤ ResultCaseCount[atNonLinearVibration])*
NonLinearVibrationParameters	*parameters of the nonlinear vibration analysis*

Retrieves parameters of a nonlinear vibration analysis.
Possible error code: reResultCaseIndexOutOfBounds or reNLpackageNotAvailable

---

long **GetNonLinearVibrationParameters_V162** ([in] long **ResultCase,**
[i/o] RVibration_V162 **NonLinearVibrationParameters**)

ResultCase	*result case index* *(0 < ResultCase ≤ ResultCaseCount[atNonLinearVibration])*
NonLinearVibrationParameters	*parameters of the nonlinear vibration analysis*

Retrieves parameters of a nonlinear vibration analysis.
Possible error code: reResultCaseIndexOutOfBounds or reNLpackageNotAvailable

---

long **GetPushOverParameters** ([in] long **ResultCase,**
[i/o] RPushOverAnalysis_V162 **PushOverParameters**)

ResultCase	*result case index* *(0 < ResultCase ≤ ResultCaseCount[atNonLinearStatic])*
PushOverParameters	*parameters of the pushover analysis*

Retrieves parameters of a pushover analysis if successful, otherwise an error code (reResultCaseIndexOutOfBounds or reNLpackageNotAvailable).

---

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 **GetTotalLoadsByLoadCaseID** ([in] EAnalysisType **AnalysisType,** [in] long **LoadCaseId,** [out] ElongBoolean **UnbalancedLoadsExist,** [i/o] RTotalLoads **TotalLoads**)

AnalysisType *analysis type must be atLinearVibration or atNonLinearVibration*
LoadCaseId *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**)

Warning! *This function has become obsolete, it was superseded by GetVibrationParameters_V162*

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 **GetVibrationParameters_V162** ([in] long **ResultCase,** [i/o] RVibration_V162 **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)

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.*

Get default properties of the XY chart. Returns number all points from all length of the JSON string.

---

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)

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)*

Generates enhanced metafile containing XY chart from points in datasets. Returns number all points from all datasets. Possible error code: errDatabaseNotReady.

---

long **SetModeActive** ([in] EAnalysisType **AnalysisType,** [in] long **ResultCase,** [in] long **FrequencyId,** [in] ELongBoolean **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 *True if mass active for this mode shape (frequency)*

Returns FrequencyId if successful, otherwise an error code (reInvalidAnalysisType, errDatabaseNotReady, reFrequencyIndexOutOfBounds, reResultCaseIndexOutOfBounds)

---

long **SetUserCreep** ([in] ELongBoolean **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).

---

long **UpdateResults**

Updates all results , to consider changes in input values without re-analysis. Returns 1 if succesfull, otherwise an error code (errDatabaseNotReady)

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*

IAxisVMCalcCrackWidth* **CalcCrackWidth** *Get crack widths of surface elements, based on the calculated reinforcement (extension module RC1 is required)*

IAxisVMCrackWidth* **CrackWidth** *Get crack widths of surface elements, based on the actual reinforcement (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.
AnalysisType must be atLinearVibration or atNonLinearVibration,
(0 < ResultCase ≤ ResultCaseCount[AnalysisType])
(0 < FrequencyID ≤ ModeShapeCount[AnalysisType,ResultCase])
Error codes are reInvalidAnalysisType, reResultCaseIndexOutOfBounds or reFrequencyIndexOutOfBounds.

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 reResultCaseIsNotLoadCase 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 reResultCaseIsNotLoadCombination 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.

AnalysisType must be atBuckling, (0 < ResultCase ≤ ResultCaseCount[atBuckling])
(0 < ModeShapeID ≤ ModeShapeCount[AnalysisType,ResultCase])
Error codes are reNLpackageNotAvailable, reInvalidAnalysisType, reResultCaseIndexOutOfBounds or reModeShapeIndexOutOfBounds .

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**]
Result case index of a given load case for a specific analysis type

(0 < LoadCaseId ≤ IAxisVMLoadCases.Count) or an error code.
Error codes are reInvalidAnalysisType and reLoadCaseIsOutOfBounds.

long **ResultCaseOfLoadCombination** [[in] EAnalysisType **AnalysisType,**
[in] long **LoadCombination**]
Result case index of a given load combination for a specific analysis type

(0 < LoadCombination ≤ IAxisVMLoadCombinations.Count) or an error code.
Error codes are reInvalidAnalysisType and reLoadCombinationIsOutOfBounds.

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**]
Total number of time steps in a dynamic analysis (atDynamic)
(0 < ResultCase ≤ ResultCaseCount[AnalysisType]) or an error code.

	Error codes are reInvalidAnalysisType and reResultCaseIndexOutOfBounds.
IAxisVMTimberDesignResults*	**TimberDesignResults** *Get timber design results of the model (extension module TD1 is required)*
ELongBoolean	**UserCreep** *Returns lbTrue if nonlinear analysis results consider concrete creep.* *More here…*
IAxisVMVelocity*	**Velocity** *Get velocity results of the model (extension module DYN is required)*
IAxisVMVerticalDisplacements*	**VerticalDisplacements** *vertical displacements results of the model. Useable only under special circumstances.*

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 poperties:

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 < LoadLevel ≤ LoadLevelCount[atNonLinearStatic,ResultBlockOfLoadCase[atNonLinearStatic,LoadCaseId]]*
atLinearVibration	*0 < ModeShape ≤ ModeShapeCount[atLinearVibration,ResultBlockOfLoadCase[atLinearVibration,LoadCaseId]]*
atNonLinearVibration	*0 < ModeShape ≤ ModeShapeCount[atNonLinearVibration,ResultBlockOfLoadCase[atNonLinearVibration,LoadCaseId]]*
atBuckling	*0 <ModeShape ≤ ModeShapeCount [atBuckling,ResultBlockOfLoadCase[atBuckling,LoadCaseId]]*
atDynamic	*0 <TimeStep ≤ TimeStepCount[atDynamic,ResultBlockOfLoadCase[atDynamic,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*
*xxx***ByLoadCaseId**	Get *xxx* results for a certain element location in the load case identified by the **LoadCaseId** property in the subject interface (IAxisVMDisplacements, IAxsiVMForces, etc). *LoadLevelOrModeShapeOrTimeStep or LoadLevelOrTimeStep property must be set before calling these functions.*
*xxx***ByLoadCombinationId**	Get *xxx* results for a certain element location identified by the **LoadCombinationId** property in the subject interface (IAxisVMDisplacements, IAxsiVMForces, etc). *LoadLevelOrModeShapeOrTimeStep or LoadLevelOrTimeStep property must be set before calling these functions.*
Envelope*xxx*	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.
Envelope*xxx***2**	Same as **Envelope*xxx*** but also load case or load combination index belonging to resulting value is retrieved.
Critical*xxx*	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.
Critical*xxx***2**	Same as **Critical*xxx*** 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 of 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*

*xxx***sByLoadCaseId**	Get *xxx* results in all locations of a given element in the load case identified by the **LoadCaseId** property in subject interface (IAxisVMDisplacements, IAxsiVMForces, etc). *LoadLevelOrModeShapeOrTimeStep or LoadLevelOrTimeStep property must be set before calling these functions.*
*xxx***sByLoadCombinationId**	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, IAxsiVMForces, etc). *LoadLevelOrModeShapeOrTimeStep or LoadLevelOrTimeStep property must be set before calling these functions.*
Envelope*xxx***s**	Get all envelope values of *xxx* results for a given element according to the **EnvelopeUID,** …**Component** and **MinMaxType** (if available) properties.
Critical*xxx***s**	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.

Multiple element reader functions	*Description*
All*xxx***sByLoadCaseId**	Get *xxx* results for all elements in the load case identified by the **LoadCaseId** property in the subject interface (IAxisVMDisplacements, IAxsiVMForces, etc).. *LoadLevelOrModeShapeOrTimeStep or LoadLevelOrTimeStep property must be set before calling these functions.*
All*xxx***sByLoadCombinationId**	Get *xxx* results for all elements in the load case identified by the **LoadCombinationId** property in subject interface (IAxisVMDisplacements, IAxsiVMForces, etc). *LoadLevelOrModeShapeOrTimeStep or LoadLevelOrTimeStep property must be set before calling these functions.*
AllEnvelope*xxx***s**	Get envelope value of *xxx* results for all elements according to the **EnvelopeUID,** …**Component** and **MinMaxType** (if available) properties.
AllCritical*xxx***s**	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**.

*xxx***sForResultBlocks**	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 **EAccelerationError** = {

aeLoadCaseIdIndexOutOfBounds = –100001	*LoadCaseId is invalid*
aeInvalidCombinationOfLoadCaseAndTimeStep = –100002	*invalid combination of LoadCaseId and TimeStep*
aeInvalidAnalysisType = –100003	*AnalysisType is incompatible with the function*
aeLoadCombinationHasNoDynamicResult = –100004 }	*dynamic results for the LoadCaseId are missing*

Enumerated types

enum **EAcceleration** = {

acX = 0	*acceleration in local x direction*
acY = 1	*acceleration in local y direction*
acZ = 2	*acceleration in local z direction*
acXX = 3	*angular acceleration about local x direction*
acYY = 4	*angular acceleration about local y direction*
acZZ = 5	*angular acceleration about local z direction*
acR = 6	*resultant acceleration*
acRR = 7}	*resultant angular acceleration*

Acceleration types

Records / structures

RAccelerationValues = (

double	**avX**	*acceleration in local x direction [m/s^2]*
double	**avY**	*acceleration in local y direction [m/s^2]*
double	**avZ**	*acceleration in local z direction [m/s^2]*
double	**avXX**	*angular acceleration about local x direction [rad/s^2]*
double	**avYY**	*angular acceleration about local y direction [rad/s^2]*
double	**avZZ**	*angular acceleration about local z direction [rad/s^2]*
double	**avR**	*resultant acceleration [m/s^2]*
double	**avRR**	*resultant angular acceleration [rad/s^2]*

)

Functions

long **NodalAccelerationByLoadCaseId** ([in] long **NodeId,**
[i/o] RAccelerationValues **AccelerationValues,** [out] BSTR **Combination**)

NodeId	*node index or line midpoint index (0 < NodeId ≤ AxisVMModel.Nodes.Count) or a value returned by AxisVMModel.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 l, otherwise an error code (errDatabaseNotReady, errIndexOutOfBounds, aeLoadCaseIdIndexOutOfBounds, aeInvalidCombinationOfLoadCaseAndTimeStep, aeInvalidAnalysisType, aeLoadCombinationHasNoDynamicResult).

long **EnvelopeNodalAcceleration** ([in] long **NodeId,**
[i/o] RAccelerationValues **AccelerationValues,** [out] BSTR **Combination**)

NodeId	*node index or line midpoint index* *(0 < NodeId ≤ AxisVMModel.Nodes.Count) or a value returned by AxisVMModel.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 l, otherwise an error code (errDatabaseNotReady, errIndexOutOfBounds, aeLoadCaseIdIndexOutOfBounds, aeInvalidCombinationOfLoadCaseAndTimeStep, aeInvalidAnalysisType, aeLoadCombinationHasNoDynamicResult).

Properties

EAnalysisType **AnalysisType** • *Get or set type of analysis*

EVelocity **Component** • *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 < LoadCaseId ≤ AxisVMModel.LoadCases.Count)

EMinMaxType **MinMaxType** • *Get or set the if minimum or maximum valuesof 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.MiscelSettings 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** = {

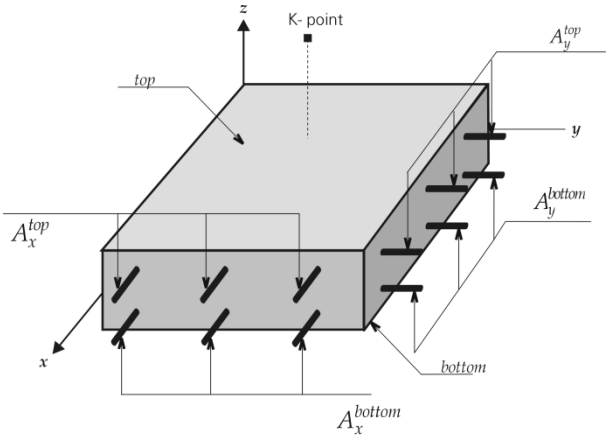
rsOK = 0	*Reinforcement OK*
rsComprReinforcementNeededX = 1	*Compression reinforcement needed in x (or ξ) direction*
rsComprReinforcementNeededY = 2	*Compression reinforcement needed in y (or η) direction*
rsCannotBeReinforcedX = 3	*Cannot be reinforced according to code in x (or ξ) direction*
rsCannotBeReinforcedY = 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	*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*
creInvalidCombinationOfLoadCaseAndLoadLevel = -100006	*Invalid combination LoadCaseID and load level*
creInvalidCombinationOfLoadCombinationAndLoadLevel = -100007	*Invalid combination CombinationID and load level*

}

Records / structures

RReinforcementValues= (

double	**Asbx**	*Bottom reinforcement in x (or ξ) direction [m^2/m]*
double	**Asby**	*Bottom reinforcement in y (or η) direction [m^2/m]*
double	**Astx**	*Top reinforcement in x (or ξ) direction [m^2/m]*
double	**Asty**	*Top reinforcement in y (or η) direction [m^2/m]*
EReinforcementStatus	**AsbxStatus**	*Status of bottom reinforcement in x (or ξ) direction*
EReinforcementStatus	**AsbyStatus**	*Status of bottom reinforcement in y (or η) direction*
EReinforcementStatus	**AstxStatus**	*Status of top reinforcement in x (or ξ) direction*
EReinforcementStatus	**AstyStatus**	*Status of top reinforcement in y (or η) direction*

)

***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**	*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*
RReinforcementValues	**rvCenterPoint**	*Reinforcement at centre point*
RReinforcementValues	**rvContourPoint1**	*Reinforcement at contour point 1*
RReinforcementValues	**rvContourPoint2**	*Reinforcement at contour point 2*
RReinforcementValues	**rvContourPoint3**	*Reinforcement at contour point 3*
RReinforcementValues	**rvContourPoint4**	*Reinforcement at contour point 4*
RReinforcementValues	**rvContourLineMidPoint1**	*Reinforcement at contour line midpoint 1*
RReinforcementValues	**rvContourLineMidPoint2**	*Reinforcement at contour line midpoint 2*
RReinforcementValues	**rvContourLineMidPoint3**	*Reinforcement at contour line midpoint 3*
RReinforcementValues	**rvContourLineMidPoint4**	*Reinforcement at contour line midpoint 4*

)

Functions

long **GetCalculatedReinforcementsByLoadCaseId** ([in] long **SurfaceId,** [in] long **LoadCaseId,** [in] long **LoadLevel**, [in] EAnalyslisType **AnalysisType,** [i/o] RReinforcements* **Reinforcements,** [out] BSTR* **Combination**)

SurfaceId	*index of the surface element*
LoadCaseId	*load case index (0 < LoadCaseId ≤ AxisVMLoadCases.Count)*
LoadLevel	*load level (increment) index*
AnalysisType	*Type of Analysis*
Reinforcements	*Reinforcement of the surface element*
Combination	*String with name of load case.*

If successful returns SurfaceId, otherwise an error code (errDatabaseNotReady, errNotSupportedByNationalDesignCode, errIndexOutOfBounds, creInvalidCombinationOfLoadCaseAndLoadLevel, creInvalidAnalysisType).

---

long **GetCalculatedReinforcementsByLoadCombinationId** ([in] long **SurfaceId,** [in] long **LoadCombinationId,** [in] long **LoadLevel**, [in] EAnalysisType **AnalysisType,** [i/o] RReinforcements* **Reinforcements,** [out] BSTR* **Combination**)

SurfaceId	*index of the surface element*
LoadCombinationId	*load combination index (0 < LoadCombinationId ≤ AxisVMLoadCombinations.Count)*
LoadLevel	*load level (increment) index*
AnalysisType	*Type of Analysis*
Reinforcements	*Reinforcement of the surface element*
Combination	*String with name of combination*

If successful returns SurfaceId, otherwise an error code (errDatabaseNotReady, errNotSupportedByNationalDesignCode, errIndexOutOfBounds, creLoadCombinationIdIndexOutOfBounds, creInvalidCombinationOfLoadCombinationAndLoadLevel, creInvalidAnalysisType).

---

long **GetEnvelopeCalculatedReinforcements** ([in] long **SurfaceId,** [in] EAnalysisType **AnalysisType,** [i/o] RReinforcements* **Reinforcements,** [out] BSTR* **Combination**)

SurfaceId	*index of the surface element*
AnalysisType	*Type of Analysis*
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*

Retrieves envelope results. Envelope is identified by Component , EnvelopeUID and MinMaxType properties.
If successful returns SurfaceId, otherwise an error code (errDatabaseNotReady, errNotSupportedByNationalDesignCode, errIndexOutOfBounds, creInvalidAnalysisType).

---

long **GetEnvelopeCalculatedReinforcements2** ([in] long **SurfaceId,** [in] EAnalysisType **AnalysisType,** [i/o] RReinforcements* **Reinforcements,** [out] **LoadCaseOrCombinationId,** [out] long **LoadLevel**)

SurfaceId	*index of the surface element*
AnalysisType	*Type of Analysis*
Reinforcements	*Reinforcement of the surface element*
LoadCaseOrCombinationId	*load case or load combination index of surface midpoint, if index > IAxisVMLoadcases.count then Load combination index = LoadCaseOrCombinationId – IAxisVMLoadcases.count*
LoadLevel	*load level*

Retrieves envelope results. Envelope is identified by Component , EnvelopeUID and MinMaxType properties.
If successful returns SurfaceId, otherwise an error code (errDatabaseNotReady, errNotSupportedByNationalDesignCode, errIndexOutOfBounds, creInvalidAnalysisType).

---

long **GetCriticalCalculatedReinforcements** ([in] long **SurfaceId,**
[in] ECombinationType **CombinationType,** [in] EAnalysisType **AnalysisType,**
[in] EReinforcement **Component,** [i/o] RReinforcements* **Reinforcements,**
[out] BSTR* **Combination**)

SurfaceId	*index of the surface element*
CombinationType	*Combination Type*
AnalysisType	*Type of Analysis*
Component	*Reinforcement component*
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*

If successful returns SurfaceId, otherwise an error code (errDatabaseNotReady, errNotSupportedByNationalDesignCode, errIndexOutOfBounds, creCombinationTypeNotValidForCurrentNationalDesignCode, creInvalidAnalysisType).

---

long **GetCriticalCalculatedReinforcements2** ([in] long **SurfaceId,**
[in] ECombinationType **CombinationType,** [in] EAnalysisType **AnalysisType,**
[in] EReinforcement **Component,** [i/o] RReinforcements* **Reinforcements,**
[out] ECombinationType **CriticalCombinationType,** [out] SAFEARRAY(double) **Factors,**
[out] SAFEARRAY(long) **LoadCaseIds**)

SurfaceId	*index of the surface element*
CombinationType	*Combination Type*
AnalysisType	*Type of Analysis*
Component	*Reinforcement component*
Reinforcements	*Reinforcement of the surface element*
CriticalCombinationType	*combination type corresponding to critical results*
Factors	*load factors of the critical load combination*
LoadCaseIds	*load case indexes of the critical load combination*

If successful returns SurfaceId, otherwise an error code (errDatabaseNotReady, errNotSupportedByNationalDesignCode, errIndexOutOfBounds, creCombinationTypeNotValidForCurrentNationalDesignCode, creInvalidAnalysisType).

---

long **GetAllCalculatedReinforcementsByLoadCaseId** ([in] long **LoadCaseId,** [in] long **LoadLevel**,
[in] EAnalysisType **AnalysisType,** [out] SAFEARRAY(RReinforcements)* **Reinforcements,**
[out] SAFEARRAY (BSTR)* **Combinations**)

LoadCaseId	*load case index (0 < LoadCaseId ≤ AxisVMLoadCases.Count)*
LoadLevel	*load level (increment) index*
AnalysisType	*Type of Analysis*
Reinforcements	*Array of reinforcements for each surface element*
Combinations	*Array of strings with names of load case for all surface elements.*

If successful returns number of all surface elements, otherwise returns an error code (errDatabaseNotReady, errNotSupportedByNationalDesignCode, creInvalidCombinationOfLoadCaseAndLoadLevel, creInvalidAnalysisType).

---

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 **SurfaceId,**
[i/o] RReinforcements* **Reinforcements,** [out] BSTR* **Combination**)

SurfaceId	*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 SurfaceId, otherwise an error code (errDatabaseNotReady, errNotSupportedByNationalDesignCode, errIndexOutOfBounds, creInvalidCombinationOfLoadCaseAndLoadLevel, creInvalidAnalysisType).

---

long **CalculatedReinforcementsByLoadCombinationId** ([in] long **SurfaceId,**
[i/o] RReinforcements* **Reinforcements,** [out] BSTR* **Combination**)

SurfaceId	*index of the surface element*
Reinforcements	*Reinforcement of the surface element*
Combination	*String with name of combination*

Get calculated reinforcement based on these properties: LoadCombinationID, LoadLevel and AnalysisType. If successful returns SurfaceId, otherwise an error code (errDatabaseNotReady, errNotSupportedByNationalDesignCode, errIndexOutOfBounds, creLoadCombinationIdIndexOutOfBounds, creInvalidCombinationOfLoadCombinationAndLoadLevel, creInvalidAnalysisType).

---

long **EnvelopeCalculatedReinforcements** ([in] long **SurfaceId,**
[i/o] RReinforcements* **Reinforcements,** [out] BSTR* **Combination**)

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*

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, errIndexOutOfBounds, creInvalidAnalysisType).

---

long **EnvelopeCalculatedReinforcements2** ([in] long **SurfaceId,**
[i/o] RReinforcements* **Reinforcements,** [out] long **LoadCaseOrCombinationId,**
[out] long **LoadLevel**)

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*

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, errIndexOutOfBounds, creInvalidAnalysisType).

---

long **CriticalCalculatedReinforcements** ([in] long **SurfaceId,** [i/o] RReinforcements* **Reinforcements,**
[out] BSTR* **Combination**)

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*

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).

---

long **CriticalCalculatedReinforcements2** ([in] long **SurfaceId,** [i/o] RReinforcements* **Reinforcements,** [out] ECombinationType **CriticalCombinationType,** [out] SAFEARRAY(double) **Factors,** [out] SAFEARRAY(long) **LoadCaseIds**)

SurfaceId	*index of the surface element*
Reinforcements	*Reinforcement of the surface element*
CriticalCombinationType	*combination type corresponding to critical results*
Factors	*load factors of the critical load combination for midpoint results*
LoadCaseIds	*load case indexes of the critical load combination for midpoint results*

If successful returns SurfaceId, otherwise an error code (errDatabaseNotReady, errNotSupportedByNationalDesignCode, errIndexOutOfBounds, creCombinationTypeNotValidForCurrentNationalDesignCode, creInvalidAnalysisType).

---

long **AllCalculatedReinforcementsByLoadCaseId** ([out] SAFEARRAY(RReinforcements)* **Reinforcements,** [out] SAFEARRAY (BSTR)* **Combinations**)

Reinforcements	*Reinforcement of the surface element*
Combinations	*Array of string with load case names for all surface elements.*

If successful I returns number of all surface elements, otherwise returns an error code (errDatabaseNotReady, errNotSupportedByNationalDesignCode, errIndexOutOfBounds, creInvalidCombinationOfLoadCaseAndLoadLevel, creInvalidAnalysisType).

---

long **AllCalculatedReinforcementsByLoadCombinationId** ([out] SAFEARRAY(RReinforcements)* **Reinforcements,** [out] SAFEARRAY (BSTR)* **Combinations**)

Reinforcements	*Array of reinforcements for each surface element*
Combinations	*Array of string 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 **AllEnvelopeCalculatedReinforcements** ([out] SAFEARRAY(RReinforcements)* **Reinforcements,** [out] SAFEARRAY (BSTR)* **Combinations**)

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*

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 **AllCriticalCalculatedReinforcements** ([out] SAFEARRAY(RReinforcements)* **Reinforcements,** [out] SAFEARRAY (BSTR)* **Combinations**)

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).

Properties

EAnalysisType	**AnalysisType** • *Get or set the type of analysis*
EReinforcement	**Component** • *Get or set the reinforcement component*
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 the t load case index* *(0 < LoadCaseId ≤ AxisVMLoadCases.Count)*
long	**LoadCombinationId** • *Get or set the load combination index* *(0 < LoadCaseId ≤ AxisVMLoadCombinations.Count)*
long	**LoadLevel** • *Get or set the load level (increment) index*

IAxisVMCalcCrackWidth

Interface for reading crack width results based on the calculated reinforcements. If property returning this interface is null (nil) then the extension module RC1 is not available.

Functions

long **GetCrackWidthsByLoadCaseId** ([in] long **SurfaceId,** [in] long **LoadCaseId,** [in] long **LoadLevel**, [in] EAnalysisType **AnalysisType,** [i/o] RCrackWidths* **CrackWidths,** [out] BSTR* **Combination**)

SurfaceId	*index of the surface element*
LoadCaseId	*load case index (0 < LoadCaseId ≤ 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 SurfaceId, otherwise an error code (errDatabaseNotReady, errIndexOutOfBounds, coeLoadCaseIdIndexOutOfBounds, coeInvalidAnalysisType, coeInvalidCombinationOfLoadCaseAndLoadLevel).

---

long **GetCrackWidthsByLoadCombinationId** ([in] long **SurfaceId,** [in] long **LoadCombinationId,** [in] long **LoadLevel**, [in] EAnalysisType **AnalysisType,** [i/o] RCrackWidths* **CrackWidths,** [out] BSTR* **Combination**)

SurfaceId	*index of the surface element*
LoadCombinationId	*load combination index (0 < LoadCombinationId ≤ 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 SurfaceId, otherwise an error code (errDatabaseNotReady, errIndexOutOfBounds, coeLoadCombinationIdIndexOutOfBounds, coeInvalidAnalysisType, coeInvalidCombinationOfLoadCombinationAndLoadLevel).

---

long **GetEnvelopeCrackWidths** ([in] long **SurfaceId,** [in] EAnalysisType **AnalysisType,** [in] ECrackWidth **Component,** [i/o] RCrackWidths* **CrackWidths,** [out] BSTR* **Combination**)

SurfaceId	*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 SurfaceId, otherwise an error code (errDatabaseNotReady, errIndexOutOfBounds, coeInvalidAnalysisType).

---

long **GetEnvelopeCrackWidths2** ([in] long **SurfaceId,** [in] EAnalysisType **AnalysisType,** [in] ECrackWidth **Component,** [i/o] RCrackWidths* **CrackWidths,** [out] long **LoadCaseOrCombinationId,** [out] long **LoadLevel**)

SurfaceId	*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 SurfaceId, otherwise an error code (errDatabaseNotReady, errIndexOutOfBounds, coeInvalidAnalysisType).

---

long **GetCriticalCrackWidths** ([in] long **SurfaceId,** [in] ECombinationType **CombinationType,**
[in] EAnalysisType **AnalysisType,** [in] ECrackWidth **Component,**
[i/o] RCrackWidths* **CrackWidths,** [out] BSTR* **Combination**)

SurfaceId	*index of the surface element*
CombinationType	*Combination Type*
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*

If successful returns SurfaceId, otherwise an error code (errDatabaseNotReady, errNotSupportedByNationalDesignCode, errIndexOutOfBounds, coeInvalidAnalysisType).

---

long **GetCriticalCrackWidths2** ([in] long **SurfaceId,** [in] ECombinationType **CombinationType,**
[in] EAnalysisType **AnalysisType,** [in] ECrackWidth **Component,** [i/o] RCrackWidths* **CrackWidths,**
[out] ECombinationType **CriticalCombinationType,** [out] SAFEARRAY(double) **Factors,**
[out] SAFEARRAY(long) **LoadCaseIds**)

SurfaceId	*index of the surface element*
CombinationType	*Combination Type*
AnalysisType	*Type of Analysis*
Component	*Location of cracking*
CrackWidths	*Crack widths of the surface element*
CriticalCombinationType	*combination type corresponding to of midpoint critical result*
Factors	*load factors of midpoint's critical load combination*
LoadCaseIds	*load case indexes of midpoint's critical load combination*

If successful returns SurfaceId, otherwise an error code (errDatabaseNotReady, errNotSupportedByNationalDesignCode, errIndexOutOfBounds, coeInvalidAnalysisType).

---

long **GetAllCrackWidthsByLoadCaseId** ([in] long **LoadCaseId,** [in] long **LoadLevel**,
[in] EAnalysisType **AnalysisType,** [out] SAFEARRAY(RCrackWidths)* **CrackWidths,**
[out] SAFEARRAY(BSTR)* **Combinations**)

LoadCaseId	*load case index (0 < LoadCaseId ≤ AxisVMLoadCases.Count)*
LoadLevel	*load level (increment) index*
AnalysisType	*Type of Analysis*
CrackWidths	*Crack widths of all surface elements*
Combinations	*Array of strings with names of load case for all surface elements.*

If successful returns number of surface elements, otherwise an error code (errDatabaseNotReady, , coeLoadCaseIdIndexOutOfBounds, coeInvalidAnalysisType, coeInvalidCombinationOfLoadCaseAndLoadLevel).

---

long **GetAllCrackWidthsByLoadCombinationId** ([in] long **LoadCombinationId,** [in] long **LoadLevel**,
[in] EAnalysisType **AnalysisType,** [out] SAFEARRAY(RCrackWidths)* **CrackWidths,**
[out] SAFEARRAY(BSTR)* **Combinations**)

LoadCombinationId	*load combination index (0 < LoadCombinationId ≤ AxisVMLoadCombinations.Count)*
LoadLevel	*load level (increment) index*
AnalysisType	*Type of Analysis*
CrackWidths	*Crack widths of all surface elements*
Combinations	*Array of strings with combination names for all surface elements.*

If successful returns number of surface elements, otherwise an error code (errDatabaseNotReady, coeLoadCombinationIdIndexOutOfBounds, coeInvalidAnalysisType, coeInvalidCombinationOfLoadCombinationAndLoadLevel).

---

long **GetAllEnvelopeCrackWidths** ([in] EAnalysisType **AnalysisType,** [in] ECrackWidth **Component,** [out] SAFEARRAY(RCrackWidths)* **CrackWidths,** [out] SAFEARRAY(BSTR)* **Combinations**)

AnalysisType	*Type of Analysis*
Component	*Location of cracking*
CrackWidths	*Crack widths of all surface elements*
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 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**)

CombinationType	*Combination Type*
AnalysisType	*Type of Analysis*
Component	*Location of cracking*
CrackWidths	*Crack widths of all surface elements*
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 surface elements, otherwise an error code (errDatabaseNotReady, errNotSupportedByNationalDesignCode, errIndexOutOfBounds, coeInvalidAnalysisType).

---

long **CrackWidthsByLoadCaseId** ([in] long **SurfaceId,** [i/o] RCrackWidths* **CrackWidths,** [out] BSTR* **Combination**)

SurfaceId	*index of the surface element*
CrackWidths	*Crack widths of the surface element*
Combination	*String with name of load case.*

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**)

SurfaceId	*index of the surface element*
CrackWidths	*Crack widths of the surface element*
Combination	*String with name of combination*

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**)

SurfaceId	*index of the surface element*
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 SurfaceId, otherwise an error code (errDatabaseNotReady, errIndexOutOfBounds, coeInvalidAnalysisType).

---

long **EnvelopeCrackWidths2** ([in] long **SurfaceId,** [i/o] RCrackWidths* **CrackWidths,**
[out] long **LoadCaseOrCombinationId,** [out] long **LoadLevel**)

SurfaceId	*index of the surface element*
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 SurfaceId, otherwise an error code (errDatabaseNotReady, errIndexOutOfBounds, coeInvalidAnalysisType).

---

long **CriticalCrackWidths** ([in] long **SurfaceId,** [i/o] RCrackWidths* **CrackWidths,** [out] BSTR* **Combination**)

SurfaceId	*index of the surface element*
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*

If successful returns SurfaceId, otherwise an error code (errDatabaseNotReady, errNotSupportedByNationalDesignCode, errIndexOutOfBounds, coeInvalidAnalysisType).

---

long **CriticalCrackWidths2** ([in] long **SurfaceId,** [i/o] RCrackWidths* **CrackWidths,**
[out] ECombinationType **CriticalCombinationType,** [out] SAFEARRAY(double) **Factors,**
[out] SAFEARRAY(long) **LoadCaseIds**)

SurfaceId	*index of the surface element*
CrackWidths	*Crack widths of the surface element*
CriticalCombinationType	*combination type corresponding to of surface midpoint's critical result*
Factors	*load factors of midpoint's critical load combination*
LoadCaseIds	*load case indexes of midpoint's critical load combination*

If successful returns SurfaceId, otherwise an error code (errDatabaseNotReady, errNotSupportedByNationalDesignCode, errIndexOutOfBounds, coeInvalidAnalysisType).

---

long **AllCrackWidthsByLoadCaseId** ([out] SAFEARRAY(RCrackWidths)* **CrackWidths,**
[out] SAFEARRAY(BSTR)* **Combinations**)

CrackWidths	*Crack widths of all surface elements*
Combinations	*Array of strings with names of load case for all surface elements.*

If successful returns number of surface elements, otherwise an error code (errDatabaseNotReady, , coeLoadCaseIdIndexOutOfBounds, coeInvalidAnalysisType, coeInvalidCombinationOfLoadCaseAndLoadLevel).

---

long **AllCrackWidthsByLoadCombinationId** ([out] SAFEARRAY(RCrackWidths)* **CrackWidths,** [out] SAFEARRAY(BSTR)* **Combinations**)

CrackWidths	*Crack widths of all surface elements*
Combinations	*Array of strings with combination names for all surface elements.*

If successful returns number of surface elements, otherwise an error code (errDatabaseNotReady, coeLoadCombinationIdIndexOutOfBounds, coeInvalidAnalysisType, coeInvalidCombinationOfLoadCombinationAndLoadLevel).

---

long **AllEnvelopeCrackWidths** ([out] SAFEARRAY(RCrackWidths)* **CrackWidths,**
[out] SAFEARRAY(BSTR)* **Combinations**)

CrackWidths *Crack widths of all surface elements*
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*

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**)

CrackWidths *Crack widths of all surface elements*
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 surface elements, otherwise an error code (errDatabaseNotReady, errNotSupportedByNationalDesignCode, errIndexOutOfBounds, coeInvalidAnalysisType).

---

long **SetUserCreep** ([in] ELongBoolean **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 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 < LoadCaseId ≤ AxisVMLoadCombinations.Count)

long **LoadLevel** •
Get or set load level (increment) index

ELongBoolean **UserCreep**
Returns lbTrue if nonlinear analysis results consider concrete creep. More here…

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 — *at bottom*
cwTop = 1 } — *at top*
Position of the crack

enum **ERebarType** = {
rtRibbed = 0 — *ribbed rebar*
rtWelded = 1 — *welded rebar*
rtSmooth = 2 } — *smooth rebar*
Type of rebar

Error codes

enum **ECrackWidthsError**= {

cweCOMError = -100001	*COM server error*
cweLoadCaseIdIndexOutOfBounds = -100002	*Invalid LoadCaseID while reading results*
cweLoadCombinationIdIndexOutOfBounds = -100003	*Invalid CombinationID while reading results*
cweInvalidAnalysisType = -100004	*Invalid type of results for used analysis*
cweCombinationTypeNotValidForCurrentNationalDesignCode = -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*
cweInvalidCombinationOfLoadCaseAndLoadLevel = -100006	*Invalid combination LoadCaseID and load level*
cweInvalidCombinationOfLoadCombinationAndLoadLevel = -100007 }	*Invalid combination CombinationID and load level or this*

Records / structures

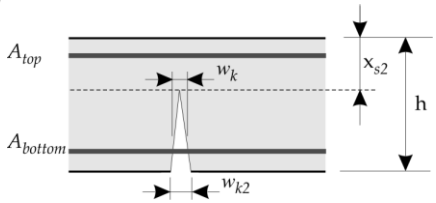
RCrackWidthValues = (

double	**Asbx**	*Bottom reinforcement in x (or ξ) direction [m^2/m]*
double	**Asby**	*Bottom reinforcement in y (or η) direction [m^2/m]*
double	**Astx**	*Top reinforcement in x (or ξ) direction [m^2/m]*
double	**Asty**	*Top reinforcement in y (or η) direction [m^2/m]*
double	**Nx**	*Axial force in x direction [kN/m]*
double	**Ny**	*Axial force in y direction [kN/m]*
double	**Nxy**	*Axial force in xy direction [kN/m]*
double	**Mx**	*Bending moment in x direction [kNm/m]*
double	**My**	*Bending moment in y direction [kNm/m]*
double	**Mxy**	*Bending moment inx y direction [kNm/m]*
double	**wR_p**	*Angle of primary cracks relative to the local x direction [rad]*
double	**wR_s**	*Angle of secondary cracks relative to the local x direction [rad]*
double	**wk_p**	*Primary crack width at the axis of the rebar [m]*
double	**wk_s**	*Secondary crack width at the axis of the rebar [m]*
double	**wk2_p**	*Primary crack width at the surface level [m]*
double	**wk2_s**	*Secondary crack width at the surface level [m]*
double	**xs2_p**	*Position of neutral axis of primary crack relative to the edge on the compression side [m]*

double	**xs2_s**	*Position of neutral axis of secondary crack relative to the edge on the compression side [m]*
double	**Ss2_p**	*σ_{s2} Stress in reinforcement in primary direction [kN/m²]*
double	**Ss2_s**	*σ_{s2} Stress in reinforcement in secondary direction [kN/m²]*
double	**Sb1_p**	*σ_b Stress in concrete in primary direction [kN/m²]*
double	**Sb1_s**	*σ_b Stress in concrete in secondary direction [kN/m²]*

)

see crack width calculation in AxisVM manual



RCrackWidths = (

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*
RCrackWidthValues	**cwvCenterPoint_Bottom**	*Crack width at bottom of centre point*
RCrackWidthValues	**cwvCenterPoint_Top**	*Crack width at top of of centre point*
RCrackWidthValues	**cwvContourPoint1_Bottom**	*Crack width at bottom of contour point 1*
RCrackWidthValues	**cwvContourPoint1_Top**	*Crack width at top of contour point 1*
RCrackWidthValues	**cwvContourPoint2_Bottom**	*Crack width at bottom of contour point 2*
RCrackWidthValues	**cwvContourPoint2_Top**	*Crack width at top of contour point 2*
RCrackWidthValues	**cwvContourPoint3_Bottom**	*Crack width at bottom of contour point 3*
RCrackWidthValues	**cwvContourPoint3_Top**	*Crack width at top of contour point 3*
RCrackWidthValues	**cwvContourPoint4_Bottom**	*Crack width at bottom of contour point 4*
RCrackWidthValues	**cwvContourPoint4_Top**	*Crack width at top of contour point 4*
RCrackWidthValues	**cwvContourLineMidPoint1_Bottom**	*Crack width at bottom of contour line midpoint 1*
RCrackWidthValues	**cwvContourLineMidPoint1_Top**	*Crack width at top of contour line midpoint 1*
RCrackWidthValues	**cwvContourLineMidPoint2_Bottom**	*Crack width at bottom of contour line midpoint 2*
RCrackWidthValues	**cwvContourLineMidPoint2_Top**	*Crack width at top of contour line midpoint 2*
RCrackWidthValues	**cwvContourLineMidPoint3_Bottom**	*Crack width at bottom of contour line midpoint 3*
RCrackWidthValues	**cwvContourLineMidPoint3_Top**	*Crack width at top of contour line midpoint 3*
RCrackWidthValues	**cwvContourLineMidPoint4_Bottom**	*Crack width at bottom of contour line midpoint 4*
RCrackWidthValues	**cwvContourLineMidPoint4_Top**	*Crack width at top of contour line midpoint 4*

)

Functions

long **GetCrackWidthsByLoadCaseId** ([in] long **SurfaceId,** [in] long **LoadCaseId,** [in] long **LoadLevel**, [in] EAnalysisType **AnalysisType,** [i/o] RCrackWidths* **CrackWidths,** [out] BSTR* **Combination**)

SurfaceId	*index of the surface element*
LoadCaseId	*load case index (0 < LoadCaseId ≤ 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 SurfaceId, otherwise an error code (errDatabaseNotReady, errIndexOutOfBounds, coeLoadCaseIdIndexOutOfBounds, coeInvalidAnalysisType, coeInvalidCombinationOfLoadCaseAndLoadLevel).

---

long **GetCrackWidthsByLoadCombinationId** ([in] long **SurfaceId,** [in] long **LoadCombinationId,** [in] long **LoadLevel**, [in] EAnalysisType **AnalysisType,** [i/o] RCrackWidths* **CrackWidths,** [out] BSTR* **Combination**)

SurfaceId	*index of the surface element*
LoadCombinationId	*load combination index (0 < LoadCombinationId ≤ 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 SurfaceId, otherwise an error code (errDatabaseNotReady, errIndexOutOfBounds, coeLoadCombinationIdIndexOutOfBounds, coeInvalidAnalysisType, coeInvalidCombinationOfLoadCombinationAndLoadLevel).

---

long **GetEnvelopeCrackWidths** ([in] long **SurfaceId,** [in] EAnalysisType **AnalysisType,** [in] ECrackWidth **Component,** [i/o] RCrackWidths* **CrackWidths,** [out] BSTR* **Combination**)

SurfaceId	*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 SurfaceId, otherwise an error code (errDatabaseNotReady, errIndexOutOfBounds, coeInvalidAnalysisType).

---

long **GetEnvelopeCrackWidths2** ([in] long **SurfaceId,** [in] EAnalysisType **AnalysisType,** [in] ECrackWidth **Component,** [i/o] RCrackWidths* **CrackWidths,** [out] long **LoadCaseOrCombinationId,** [out] long **LoadLevel**)

SurfaceId	*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 SurfaceId, otherwise an error code (errDatabaseNotReady, errIndexOutOfBounds, coeInvalidAnalysisType).

---

long **GetCriticalCrackWidths** ([in] long **SurfaceId,** [in] ECombinationType **CombinationType,**
[in] EAnalysisType **AnalysisType,** [in] ECrackWidth **Component,**
[i/o] RCrackWidths* **CrackWidths,** [out] BSTR* **Combination**)

SurfaceId	*index of the surface element*
CombinationType	*Combination Type*
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*

If successful returns SurfaceId, otherwise an error code (errDatabaseNotReady, errNotSupportedByNationalDesignCode, errIndexOutOfBounds, coeInvalidAnalysisType).

---

long **GetCriticalCrackWidths2** ([in] long **SurfaceId,** [in] ECombinationType **CombinationType,**
[in] EAnalysisType **AnalysisType,** [in] ECrackWidth **Component,** [i/o] RCrackWidths* **CrackWidths,**
[out] ECombinationType **CriticalCombinationType,** [out] SAFEARRAY(double) **Factors,**
[out] SAFEARRAY(long) **LoadCaseIds**)

SurfaceId	*index of the surface element*
CombinationType	*Combination Type*
AnalysisType	*Type of Analysis*
Component	*Location of cracking*
CrackWidths	*Crack widths of the surface element*
CriticalCombinationType	*combination type corresponding to of midpoint critical result*
Factors	*load factors of midpoint's critical load combination*
LoadCaseIds	*load case indexes of midpoint's critical load combination*

If successful returns SurfaceId, otherwise an error code (errDatabaseNotReady, errNotSupportedByNationalDesignCode, errIndexOutOfBounds, coeInvalidAnalysisType).

---

long **GetAllCrackWidthsByLoadCaseId** ([in] long **LoadCaseId,** [in] long **LoadLevel**,
[in] EAnalysisType **AnalysisType,** [out] SAFEARRAY(RCrackWidths)* **CrackWidths,**
[out] SAFEARRAY(BSTR)* **Combinations**)

LoadCaseId	*load case index (0 < LoadCaseId ≤ AxisVMLoadCases.Count)*
LoadLevel	*load level (increment) index*
AnalysisType	*Type of Analysis*
CrackWidths	*Crack widths of all surface elements*
Combinations	*Array of strings with names of load case for all surface elements.*

If successful returns number of surface elements, otherwise an error code (errDatabaseNotReady, , coeLoadCaseIdIndexOutOfBounds, coeInvalidAnalysisType, coeInvalidCombinationOfLoadCaseAndLoadLevel).

---

long **GetAllCrackWidthsByLoadCombinationId** ([in] long **LoadCombinationId,** [in] long **LoadLevel**,
[in] EAnalysisType **AnalysisType,** [out] SAFEARRAY(RCrackWidths)* **CrackWidths,**
[out] SAFEARRAY(BSTR)* **Combinations**)

LoadCombinationId	*load combination index (0 < LoadCombinationId ≤ AxisVMLoadCombinations.Count)*
LoadLevel	*load level (increment) index*
AnalysisType	*Type of Analysis*
CrackWidths	*Crack widths of all surface elements*
Combinations	*Array of strings with combination names for all surface elements.*

If successful returns number of surface elements, otherwise an error code (errDatabaseNotReady, coeLoadCombinationIdIndexOutOfBounds, coeInvalidAnalysisType, coeInvalidCombinationOfLoadCombinationAndLoadLevel).

---

long **GetAllEnvelopeCrackWidths** ([in] EAnalysisType **AnalysisType,** [in] ECrackWidth **Component,**
[out] SAFEARRAY(RCrackWidths)* **CrackWidths,** [out] SAFEARRAY(BSTR)* **Combinations**)

AnalysisType	*Type of Analysis*
Component	*Location of cracking*
CrackWidths	*Crack widths of all surface elements*
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 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**)

CombinationType	*Combination Type*
AnalysisType	*Type of Analysis*
Component	*Location of cracking*
CrackWidths	*Crack widths of all surface elements*
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 surface elements, otherwise an error code (errDatabaseNotReady, errNotSupportedByNationalDesignCode, errIndexOutOfBounds, coeInvalidAnalysisType).

---

long **CrackWidthsByLoadCaseId** ([in] long **SurfaceId,** [i/o] RCrackWidths* **CrackWidths,**
[out] BSTR* **Combination**)

SurfaceId	*index of the surface element*
CrackWidths	*Crack widths of the surface element*
Combination	*String with name of load case.*

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**)

SurfaceId	*index of the surface element*
CrackWidths	*Crack widths of the surface element*
Combination	*String with name of combination*

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**)

SurfaceId	*index of the surface element*
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 SurfaceId, otherwise an error code (errDatabaseNotReady, errIndexOutOfBounds, coeInvalidAnalysisType).

---

long **EnvelopeCrackWidths2** ([in] long **SurfaceId,** [i/o] RCrackWidths* **CrackWidths,**
[out] long **LoadCaseOrCombinationId,** [out] long **LoadLevel**)

SurfaceId	*index of the surface element*
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 SurfaceId, otherwise an error code (errDatabaseNotReady, errIndexOutOfBounds, coeInvalidAnalysisType).

---

long **CriticalCrackWidths** ([in] long **SurfaceId,** [i/o] RCrackWidths* **CrackWidths,** [out] BSTR* **Combination**)

SurfaceId	*index of the surface element*
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*

If successful returns SurfaceId, otherwise an error code (errDatabaseNotReady, errNotSupportedByNationalDesignCode, errIndexOutOfBounds, coeInvalidAnalysisType).

---

long **CriticalCrackWidths2** ([in] long **SurfaceId,** [i/o] RCrackWidths* **CrackWidths,**
[out] ECombinationType **CriticalCombinationType,** [out] SAFEARRAY(double) **Factors,**
[out] SAFEARRAY(long) **LoadCaseIds**)

SurfaceId	*index of the surface element*
CrackWidths	*Crack widths of the surface element*
CriticalCombinationType	*combination type corresponding to of surface midpoint's critical result*
Factors	*load factors of midpoint's critical load combination*
LoadCaseIds	*load case indexes of midpoint's critical load combination*

If successful returns SurfaceId, otherwise an error code (errDatabaseNotReady, errNotSupportedByNationalDesignCode, errIndexOutOfBounds, coeInvalidAnalysisType).

---

long **AllCrackWidthsByLoadCaseId** ([out] SAFEARRAY(RCrackWidths)* **CrackWidths,**
[out] SAFEARRAY(BSTR)* **Combinations**)

CrackWidths	*Crack widths of all surface elements*
Combinations	*Array of strings with names of load case for all surface elements.*

If successful returns number of surface elements, otherwise an error code (errDatabaseNotReady, , coeLoadCaseIdIndexOutOfBounds, coeInvalidAnalysisType, coeInvalidCombinationOfLoadCaseAndLoadLevel).

---

long **AllCrackWidthsByLoadCombinationId** ([out] SAFEARRAY(RCrackWidths)* **CrackWidths,** [out] SAFEARRAY(BSTR)* **Combinations**)

CrackWidths	*Crack widths of all surface elements*
Combinations	*Array of strings with combination names for all surface elements.*

If successful returns number of surface elements, otherwise an error code (errDatabaseNotReady, coeLoadCombinationIdIndexOutOfBounds, coeInvalidAnalysisType, coeInvalidCombinationOfLoadCombinationAndLoadLevel).

---

long **AllEnvelopeCrackWidths** ([out] SAFEARRAY(RCrackWidths)* **CrackWidths,**
[out] SAFEARRAY(BSTR)* **Combinations**)

CrackWidths *Crack widths of all surface elements*
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*

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**)

CrackWidths *Crack widths of all surface elements*
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 surface elements, otherwise an error code (errDatabaseNotReady, errNotSupportedByNationalDesignCode, errIndexOutOfBounds, coeInvalidAnalysisType).

---

long **SetUserCreep** ([in] ELongBoolean **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 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 < LoadCaseId ≤ AxisVMLoadCombinations.Count)

long **LoadLevel** •
Get or set load level (increment) index

ELongBoolean **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.

Some functions use global parameters, specified as properties. This way their parameter list is shorter, but the necessary global parameters must be set prior to their call.

Error codes

enum **EDisplacementError** = {

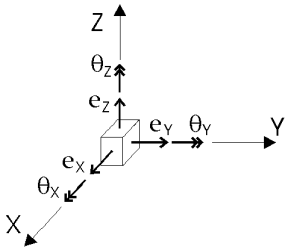
deNodeIndexOutOfBounds = –100001	*node index is out of bounds*
deLoadCaseIndexOutofBounds = –100002	*LoadCaseId is out of bounds*
deLoadCombinationIndexOutofBounds= –100003	*LoadCombinationId is out of bounds*
deCombinationTypeNotValidForCurrentNationalDesignCode= –100004	*used CombinationType is incompatible with the current design code*
deCOMError= –100005	*internal COM error when calling GetRecordInfoFromGUIDs,SafeArrayCreateEx, SafeArrayAccesssData*
deNoNodesInTheModel = –100006	*AxisVMModel.Nodes.Count = 0*
deNoLoadCasesInTheModel = –100007	*AxisVMModel.LoadCases.Count = 0*
deNoLoadCombinationsInTheModel = –100008	*AxisVMModel.LoadCombinations.Count = 0*
deSectionIndexOutOfBounds = –100009	*cross-section index is out of bounds*
deLineIndexOutOfBounds = –100010	*line index is out of bounds*
deLineHasNoSections = –100011	*line element has no sections*
deNoValidLinesInTheModel = –100012	*no valid lines in the model*
deInvalidAnalysisType = –100013	*AnalysisType is incompatible with the function*
deInvalidCombinationOfLoadCaseAndLoadLevel = –100014	*no result is available for the given LoadCaseId and LoadLevelOrModeShape or this*CreepOfConcrete
deInvalidCombinationOfLoadCombinationAndLoadLevel = –100015	*no result is available for the given LoadCombinationId and LoadLevelOrModeShape or this*
deNoResultBlocksInTheModel = –100016	*no result blocks in the model*
deInvalidLineType = –100017 ,	*Line type is not valid for the selected operation*
deMemberIndexOutOfBounds = –100018 ,	*member index is out of bounds*
deVirtualBeamIndexOutOfBounds = –100019 ,	*virtual beam or strip index is out of bounds*
deVirtualBeamChainIndexOutOfBounds = –100020 ,	*index of virtual beam or strip chain is out of bounds*
deVirtualBeamSectionIndexOutOfBounds = –100021 }	*ndex of virtual beam or strip section is out of bounds*

Enumerated types

enum **EDisplacement** = {

	nodal displacements	**line and member displacements**
d_eX = 0x01	*displacement in global X direction*	*displacement in global or local x direction (depends on DisplacementSystem property)*
d_eY = 0x02	*displacement in global Y direction*	*displacement in global or local y direction (depends on DisplacementSystem property)*
d_eZ = 0x03	*displacement in global Z direction*	*displacement in global or local z direction (depends on DisplacementSystem property)*
d_fX = 0x04	*rotation about the global X axis*	*rotation about the global or the local x axis (depends on DisplacementSystem property)*
d_fY = 0x05	*rotation about the global Y axis*	*rotation about the global or the local y axis (depends on DisplacementSystem property)*
d_fZ = 0x06	*rotation about the global Z axis*	*rotation about the global or the local z axis (depends on DisplacementSystem property)*
d_eR = 0x07	*global resultant displacement*	*global or local resultant displacement (depends on DisplacementSystem property)*
d_fR = 0x08 }	*global resultant rotation*	*global or local resultant rotation (depends on DisplacementSystem property)*

Displacement component identifiers [m]



enum **EDisplacementSystem** = {

dsLocal = 0x01 *line or member displacements in local system*
dsGlobal = 0x02 } *line or member displacements in global system*

Records / structures

RDisplacementValues = (

		nodal displacements	line and member displacements
double	**Ex**	*displacement in global X direction*	*displacement in global or local x direction (depends on DisplacementSystem property)*
double	**Ey**	*displacement in global Y direction*	*displacement in global or local y direction (depends on DisplacementSystem property)*
double	**Ez**	*displacement in global Z direction*	*displacement in global or local z direction (depends on DisplacementSystem property)*
double	**Fx**	*rotation about the global X axis*	*rotation about the global or the local x axis (depends on DisplacementSystem property)*
double	**Fy**	*rotation about the global Y axis*	*rotation about the global or the local y axis (depends on DisplacementSystem property)*
double	**Fz**	*rotation about the global Z axis*	*rotation about the global or the local z axis (depends on DisplacementSystem property)*
double	**eR**	*global resultant displacement*	*global or local resultant displacement (depends on DisplacementSystem property)*
double	**fR**	*global resultant rotation*	*global or local resultant rotation (depends on DisplacementSystem property)*

)

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 **GetNodalDisplacementByLoadCaseId** ([in] long **NodeId,** [in] long **LoadCaseId,** [in] long **LoadLevelOrModeShapeOrTimeStep,** [in] EAnalysisType **AnalysisType,** [in] ELongBoolean **WithReinforcement,** [i/o] RDisplacementValues **Displacement,** [out] BSTR **Combination**)

NodeId	*node index or line midpoint index (0 < NodeId ≤ AxisVMModel.Nodes.Count) or a value returned by AxisVMModel.Lines.MidpointId[LineIndex]*
LoadCaseId	*load case index (0 < LoadCaseId ≤ AxisVMModel.LoadCases.Count)*
LoadLevelOrModeShapeOrTimeStep	*load level or mode shape or time step, according to the type of analysis. See LoadLevelOrModeShapeOrTimeStep parameter.*
AnalysisType	*analysis type*
WithReinforcement	*determines which displacements results should be read. Considering reinforcement (lbTrue) or not (lbFalse) (this setting is taken into account only if AnalysisType = atNonLinearStatic)*
Displacement	*displacement results*
Combination	*name of the load case*

Retrieves displacements of a node or line midpoint according to the parameters. Returns NodeId or an error code (errDatabaseNotReady or see EDisplacementError).

---

long **GetNodalDisplacementByLoadCombinationId** ([in] long **NodeId,** [in] long **LoadCombinationId,** [in] long **LoadLevelOrModeShape,** [in] EAnalysisType **AnalysisType,** [in] ELongBoolean **WithReinforcement,** [i/o] RDisplacementValues **Displacement,** [out] BSTR **Combination**)

NodeId	*node index or line midpoint index (0 < NodeId ≤ AxisVMModel.Nodes.Count) or a value returned by AxisVMModel.Lines.MidpointId[LineIndex]*
LoadCombinationId	*load combination index (0 < LoadCombinationId ≤ AxisVMModel.LoadCombinations.Count)*
LoadLevelOrModeShape	*load level or mode shape, according to the type of analysis. See LoadLevelOrModeShapeOrTimeStep parameter.*
AnalysisType	*analysis type*
WithReinforcement	*determines which displacements results should be read. Considering reinforcement (lbTrue) or not (lbFalse) (this*

	setting is taken into account only if AnalysisType = atNonLinearStatic)
Displacement	*displacement results*
Combination	*name of the load combination*

Retrieves displacements of a node or line midpoint according to the parameters. Returns NodeId or an error code (errDatabaseNotReady or see EDisplacementError).

---

long **GetEnvelopeNodalDisplacement** ([in] long **NodeId,** [in] EMinMaxType **MinMaxType,** [in] EAnalysisType **AnalysisType,** [in] ELongBoolean **WithReinforcement,** [in] EDisplacement **Component,** [i/o] RDisplacementValues **Displacement,** [out] BSTR **Combination**)

NodeId	*node index or line midpoint index (0 < NodeId ≤ AxisVMModel.Nodes.Count) or a value returned by AxisVMModel.Lines.MidpointId[LineIndex])*
MinMaxType	*minmax type*
AnalysisType	*analysis type*
WithReinforcement	*determines which displacements results should be read. Considering reinforcement (lbTrue) or not (lbFalse) (this setting is taken into account only if AnalysisType = atNonLinearStatic)*
Component	*displacement component*
Displacement	*displacement results*
Combination	*name of the load case or combination in which Component has its minimum or maximum*

Retrieves envelope results. The EnvelopUID is specified by the global EnvelopUID property. Returns NodeId or an error code (errDatabaseNotReady or see EDisplacementError).

---

long **GetEnvelopeNodalDisplacement2** ([in] long **NodeId,** [in] EMinMaxType **MinMaxType,** [in] EAnalysisType **AnalysisType,** [in] ELongBoolean **WithReinforcement,** [in] EDisplacement **Component,** [i/o] RDisplacementValues **Displacement,** [out] long **LoadCaseOrCombinationId,** [out] long **LoadLevel**)

NodeId	*node index or line midpoint index (0 < NodeId ≤ AxisVMModel.Nodes.Count) or a value returned by AxisVMModel.Lines.MidpointId[LineIndex])*
MinMaxType	*minmax type*
AnalysisType	*analysis type*
WithReinforcement	*determines which displacements results should be read. Considering reinforcement (lbTrue) or not (lbFalse) (this setting is taken into account only if AnalysisType = atNonLinearStatic)*
Component	*displacement component*
Displacement	*displacement results*
LoadCaseOrCombinationId	*load case or load combination index, if index is > IAxisVMLoadcases.count then Load combination index = LoadCaseOrCombinationId – IAxisVMLoadcases.count*
LoadLevel	*load level*

Retrieves envelope results. The EnvelopUID is specified by the global EnvelopUID property. Similar to GetEnvelopeNodalDisplacement, but instead of a mere string detailed combination info is retrieved.
Returns NodeId or an error code (errDatabaseNotReady or see EDisplacementError).

---

long **GetCriticalNodalDisplacement** ([in] long **NodeId,** [in] EMinMaxType **MinMaxType,** [in] ECombinationType **CombinationType,** [in] EAnalysisType **AnalysisType,** [in] ELongBoolean **WithReinforcement,** [in] EDisplacement **Component,** [i/o] RDisplacementValues **Displacement,** [out] BSTR **Combination**)

NodeId	*node index or line midpoint index (0 < NodeId ≤ AxisVMModel.Nodes.Count) or a value returned by AxisVMModel.Lines.MidpointId[LineIndex])*
MinMaxType	*minmax type*
CombinationType	*combination type (mostly for Eurocode only)*
AnalysisType	*analysis type*

WithReinforcement	*determines which displacements results should be read. Considering reinforcement (lbTrue) or not (lbFalse) (this setting is taken into account only if AnalysisType = atNonLinearStatic)*
Component	*displacement component*
Displacement	*displacement results*
Combination	*critical combination in which Component has its minimum or maximum*

Retrieves critical displacements of a node or line midpoint according to the parameters. Returns NodeId or an error code (errDatabaseNotReady or see EDisplacementError).

---

long **GetCriticalNodalDisplacement2** ([in] long **NodeId,** [in] EMinMaxType **MinMaxType,** [in] ECombinationType **CombinationType,** [in] EAnalysisType **AnalysisType,** [in] ELongBoolean **WithReinforcement,** [in] EDisplacement **Component,** [i/o] RDisplacementValues **Displacement,** [out] ECombinationType **CriticalCombinationType,** [out] SAFEARRAY(double) **Factors,** [out] SAFEARRAY(long) **LoadCaseIds**)

NodeId	*node index or line midpoint index (0 < NodeId ≤ AxisVMModel.Nodes.Count) or a value returned by AxisVMModel.Lines.MidpointId[LineIndex])*
MinMaxType	*minmax type*
CombinationType	*combination type (mostly for Eurocode only)*
AnalysisType	*analysis type*
WithReinforcement	*determines which displacements results should be read. Considering reinforcement (lbTrue) or not (lbFalse) (this setting is taken into account only if AnalysisType = atNonLinearStatic)*
Component	*displacement component*
Displacement	*displacement results*
CriticalCombinationType	*combination type corresponding to critical load combination*
Factors	*load factors of the critical load combination*
LoadCaseIds	*load case indexes of the critical load combination*

Retrieves critical displacements of a node or line midpoint according to the parameters. Similar to GetCriticalNodalDisplacement, but retrieves detailed combination information instead of a mere string

---

long **NodalDisplacementByLoadCaseId** ([in] long **NodeId,** [i/o] RDisplacementValues **Displacement,** [out] BSTR **Combination**)

NodeId	*node index or line midpoint index (0 < NodeId ≤ AxisVMModel.Nodes.Count) or a value returned by AxisVMModel.Lines.MidpointId[LineIndex]*
Displacement	*displacement results*
Combination	*name of the load case*

Retrieves displacements of a node or line midpoint. Uses the following global properties : LoadCaseId, LoadLevelOrModeShapeOrTimeStep, AnalysisType, WithReinforcement. Set those parameters at some point before the call. 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 ≤ AxisVMModel.Nodes.Count) or a value returned by AxisVMModel.Lines.MidpointId[LineIndex]*
Displacement	*displacement results*
Combination	*name of the load combination*

Retrieves displacements of a node or line midpoint. Uses the following global properties : LoadCombinationId, LoadLevelOrModeShapeOrTimeStep, AnalysisType, WithReinforcement. Set

those parameters at some point before the call. 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 ≤ AxisVMModel.Nodes.Count) or a value returned by AxisVMModel.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. Uses the following global properties : MinMaxType, AnalysisType, WithReinforcement, Component, EnvelopeUID.
Returns NodeId or an error code (errDatabaseNotReady or see EDisplacementError).

---

long **EnvelopeNodalDisplacement2** ([in] long **NodeId,** [i/o] RDisplacementValues **Displacement,** [out] long **LoadCaseOrCombinationId,** [out] long **LoadLevel**)

NodeId	*node index or line midpoint index (0 < NodeId ≤ AxisVMModel.Nodes.Count) or a value returned by AxisVMModel.Lines.MidpointId[LineIndex])*
Displacement	*displacement results*
LoadCaseOrCombinationId	*load case or load combination index, if index is > IAxisVMLoadcases.count then Load combination index = LoadCaseOrCombinationId – IAxisVMLoadcases.count*
LoadLevel	*load level*

Retrieves envelope results. Uses the following global properties : MinMaxType, AnalysisType, WithReinforcement, Component, EnvelopeUID. Similar to EnvelopeNodalDisplacement, but instead of a mere string detailed combination info is retrieved.
Returns NodeId or an error code (errDatabaseNotReady or see EDisplacementError).

---

long **CriticalNodalDisplacement** ([in] long **NodeId,** [i/o] RDisplacementValues **Displacement,** [out] BSTR **Combination**)

NodeId	*node index or line midpoint index (0 < NodeId ≤ AxisVMModel.Nodes.Count) or a value returned by AxisVMModel.Lines.MidpointId[LineIndex])*
Displacement	*displacement results*
Combination	*critical combination in which Component has its minimum or maximum*

Retrieves critical displacements of a node or line midpoint. Uses the following global properties : MinMaxType, CombinationType, AnalysisType, WithReinforcement, Component. Returns NodeId or an error code (errDatabaseNotReady or see EDisplacementError).

---

long **CriticalNodalDisplacement2** ([in] long **NodeId,** [i/o] RDisplacementValues **Displacement,** [out] ECombinationType **CriticalCombinationType,** [out] SAFEARRAY(double) **Factors,** [out] SAFEARRAY(long) **LoadCaseIds**)

NodeId	*node index or line midpoint index (0 < NodeId ≤ AxisVMModel.Nodes.Count) or a value returned by AxisVMModel.Lines.MidpointId[LineIndex])*
Displacement	*displacement results*
CriticalCombinationType	*combination type corresponding to critical load combination*
Factors	*load factors of the critical load combination*
LoadCaseIds	*load case indexes of the critical load combination*

Retrieves critical displacements of a node or line midpoint. Uses the following global properties : MinMaxType, CombinationType, AnalysisType, WithReinforcement, Component. Similar to CriticalNodalDisplacement, but retrieves detailed combination information instead of a mere string

---

Multiple element reader functions

long **GetAllNodalDisplacementsByLoadCaseId** ([in] long **LoadCaseId,** [in] long **LoadLevelOrModeShapeOrTimeStep,** [in] EAnalysisType **AnalysisType,** [in] ELongBoolean **WithReinforcement,** [i/o] SAFEARRAY(RDisplacementValues) **Displacements**)

LoadCaseId	*load case index* *(0 < LoadCaseId ≤ AxisVMModel.LoadCases.Count)*
LoadLevelOrModeShapeOrTimeStep	*load level or mode shape or time step, according to the type of analysis. See LoadLevelOrModeShapeOrTimeStep parameter.*
AnalysisType	*analysis type*
WithReinforcement	*determines which displacements results should be read. Considering reinforcement (lbTrue) or not (lbFalse) (this setting is taken into account only if AnalysisType = atNonLinearStatic)*
Displacements	*displacement results for all nodes* *Length of the array = AxisVMModel.Nodes.Count + AxisVMModel.Lines.MidpointCount*

Retrieves displacements of all nodes and line midpoints. Returns the total number of displacement records in the array or an error code (errDatabaseNotReady or see EDisplacementError).

---

long **GetAllNodalDisplacementsByLoadCombinationId** ([in] long **LoadCombinationId,** [in] long **LoadLevelOrModeShape,** [in] EAnalysisType **AnalysisType,** [in] ELongBoolean **WithReinforcement,** [i/o] SAFEARRAY(RDisplacementValues) **Displacements**)

LoadCombinationId	*load case index* *(0 < LoadCombinationId ≤ AxisVMModel.LoadCombinations.Count)*
LoadLevelOrModeShape	*load level or mode shape, according to the type of analysis. See LoadLevelOrModeShapeOrTimeStep parameter.*
AnalysisType	*analysis type*
WithReinforcement	*determines which displacements results should be read. Considering reinforcement (lbTrue) or not (lbFalse) (this setting is taken into account only if AnalysisType = atNonLinearStatic)*
Displacements	*displacement results for all nodes* *Length of the array = AxisVMModel.Nodes.Count + AxisVMModel.Lines.MidpointCount*

Retrieves displacements of all nodes and line midpoints. Returns the total number of displacement records in the array or an error code (errDatabaseNotReady or see EDisplacementError).

---

long **GetAllEnvelopeNodalDisplacements** ([in] EMinMaxType **MinMaxType,** [in] EAnalysisType **AnalysisType,** [in] ELongBoolean **WithReinforcement,** [in] EDisplacement **Component,** [i/o] SAFEARRAY(RDisplacementValues) **Displacements**)

MinMaxType	*minmax type*
AnalysisType	*analysis type*
WithReinforcement	*determines which displacements results should be read. Considering reinforcement (lbTrue) or not (lbFalse) (this setting is taken into account only if AnalysisType = atNonLinearStatic)*
Component	*displacement component*
Displacements	*array of envelope nodal displacements for all nodes and all midpoints consecutively.* *Length of the array is AxisVMModel.Nodes.Count+AxisVMModel.Lines.MidpointCount*

Retrieves envelope results of all nodes and line midpoints. The EnvelopUID is specified by the global EnvelopUID property. Returns the total number of displacement records in the array or an error code (errDatabaseNotReady or see EDisplacementError).

---

long **GetAllCriticalNodalDisplacements** ([in] EMinMaxType **MinMaxType,** [in] ECombinationType **CombinationType,** [in] EAnalysisType **AnalysisType,** [in] ELongBoolean **WithReinforcement,** [in] EDisplacement **Component,** [i/o] SAFEARRAY(RDisplacementValues) **Displacements**)

MinMaxType	*minmax type*
CombinationType	*combination type (mostly for Eurocode only)*
AnalysisType	*analysis type*
WithReinforcement	*determines which displacements results should be read. Considering reinforcement (lbTrue) or not (lbFalse) (this setting is taken into account only if AnalysisType = atNonLinearStatic)*
Component	*displacement component*
Displacements	*array of critical nodal displacements for all nodes and all midpoints consecutively.* *Length of the array is AxisVMModel.Nodes.Count+AxisVMModel.Lines.MidpointCount*

Retrieves critical displacements of all nodes and line midpoints. Returns the total number of displacement records in the array or an error code (errDatabaseNotReady or see EDisplacementError).

---

long **AllNodalDisplacementsByLoadCaseId** ([i/o] SAFEARRAY(RDisplacementValues) **Displacements**)

Displacements	*displacement results for all nodes* *Length of the array = AxisVMModel.Nodes.Count + AxisVMModel.Lines.MidpointCount*

Retrieves displacements of all nodes and line midpoints. Uses the following global properties : LoadCaseId, LoadLevelOrModeShapeOrTimeStep, AnalysisType, WithReinforcement. 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 = AxisVMModel.Nodes.Count + AxisVMModel.Lines.MidpointCount*

Retrieves displacements of all nodes and line midpoints. Uses the following global properties : LoadCombinationId, LoadLevelOrModeShapeOrTimeStep, AnalysisType, WithReinforcement. 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 AxisVMModel.Nodes.Count+AxisVMModel.Lines.MidpointCount*

Retrieves envelope results of all nodes and line midpoints. Uses the following global properties : MinMaxType, AnalysisType, WithReinforcement, EnvelopeUID, Component. 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 AxisVMModel.Nodes.Count+AxisVMModel.Lines.MidpointCount

Retrieves critical displacements of all nodes and line midpoints. Uses the following global properties : MinMaxType, CombinationType, AnalysisType, WithReinforcement, Component. Returns the total number of displacement records in the array or an error code (errDatabaseNotReady or see EDisplacementError).

Multiple BLOCK reader functions

long **GetNodalDisplacementsForResultBlocks** ([in] long **NodeId,** [in] EAnalysisType **AnalysisType,** [in] ELongBoolean **WithReinforcement,** i/o] SAFEARRAY(RResultBlockInfo) **ResultBlockInfo,** [i/o] SAFEARRAY(RDisplacementValues) **Displacements**)

NodeId *node index or line midpoint index (0 < NodeId ≤ AxisVMModel.Nodes.Count) or a value returned by AxisVMModel.Lines.MidpointId[LineIndex])*
AnalysisType *analysis type*
WithReinforcement *determines which displacements results should be read. Considering reinforcement (lbTrue) or not (lbFalse) (this setting is taken into account only if AnalysisType = atNonLinearStatic)*
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. Returns the common length of ResultBlockInfo and Displacements arrays or an error code (errDatabaseNotReady or see EDisplacementError).

---

long **NodalDisplacementsForResultBlocks** ([in] long **NodeId,** [i/o] SAFEARRAY(RResultBlockInfo) **ResultBlockInfo,** [i/o] SAFEARRAY(RDisplacementValues) **Displacements**)

NodeId *node index or line midpoint index (0 < NodeId ≤ AxisVMModel.Nodes.Count) or a value returned by AxisVMModel.Lines.MidpointId[LineIndex])*
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. Uses the following global properties : AnalysisType, WithReinforcement. 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 **LineId** 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 **GetLineDisplacementByLoadCaseId** ([in] long **LineId,** [in] long **SectionId,** [in] long **LoadCaseId,** [in] long **LoadLevelOrTimeStep,** [in] EAnalysisType **AnalysisType,** [in] ELongBoolean **WithReinforcement,** [i/o] RDisplacementValues **Displacement,** [out] double **PosX,** [out] BSTR **Combination**)

LineId *line index (0 < LineId ≤ AxisVMModel.Lines.Count)*
SectionId *section index (0 < SectionId ≤ AxisVMModel.Lines[LineId].SectionCount)*
LoadCaseId *load case index*
(0 < LoadCaseId ≤ AxisVMModel.LoadCases.Count)

LoadLevelOrTimeStep	*load level or time step, according to the type of analysis. See LoadLevelOrModeShapeOrTimeStep parameter*
AnalysisType	*analysis type*
WithReinforcement	*determines which displacements results should be read. Considering reinforcement (lbTrue) or not (lbFalse) (this setting is taken into account only if AnalysisType = atNonLinearStatic)*
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. Returns LineId or an error code (errDatabaseNotReady or see EDisplacementError).

---

long **GetLineDisplacementByLoadCombinationId** ([in] long **LineId,** [in] long **SectionId,** [in] long **LoadCombinationId,** [in] long **LoadLevel,** [in] EAnalysisType **AnalysisType,** [in] ELongBoolean **WithReinforcement,** [i/o] RDisplacementValues **Displacement,** [out] double **PosX,** [out] BSTR **Combination**)

LineId	*line index (0 < LineId ≤ AxisVMModel.Lines.Count)*
SectionId	*section index (0 < SectionId ≤ AxisVMModel.Lines[LineId].SectionCount)*
LoadCombinationId	*load combination index (0 < LoadCombinationId ≤ AxisVMModel.LoadCombinations.Count)*
LoadLevel	*load level, according to the type of analysis. See LoadLevelOrModeShapeOrTimeStep parameter*
AnalysisType	*analysis type*
WithReinforcement	*determines which displacements results should be read. Considering reinforcement (lbTrue) or not (lbFalse) (this setting is taken into account only if AnalysisType = atNonLinearStatic)*
Displacement	*displacement results*
PosX	*position of SectionId in m according to the local x direction*
Combination	*name of the load combination*

Retrieves displacements at a section of a line element. Returns LineId or an error code (errDatabaseNotReady or see EDisplacementError).

---

long **GetEnvelopeLineDisplacement** ([in] long **LineId,** [in] long **SectionId,** [in] EMinMaxType **MinMaxType,** [in] EAnalysisType **AnalysisType,** [in] ELongBoolean **WithReinforcement,** [in] EDisplacement **Component,** [i/o] RDisplacementValues **Displacement,** [out] double **PosX,** [out] BSTR **Combination**)

LineId	*line index (0 < LineId ≤ AxisVMModel.Lines.Count)*
SectionId	*section index (0 < SectionId ≤ AxisVMModel.Lines[LineId].SectionCount)*
MinMaxType	*minmax type*
AnalysisType	*analysis type*
WithReinforcement	*determines which displacements results should be read. Considering reinforcement (lbTrue) or not (lbFalse) (this setting is taken into account only if AnalysisType = atNonLinearStatic)*
Component	*displacement component*
Displacement	*displacement 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*

Read envelope results at a section of a line element. The EnvelopUID is specified by the global EnvelopUID property. Returns LineId or an error code (errDatabaseNotReady or see EDisplacementError).

---

long **GetEnvelopeLineDisplacement2** ([in] long **LineId,** [in] long **SectionId,** [in] EMinMaxType **MinMaxType,** [in] EAnalysisType **AnalysisType,** [in] ELongBoolean **WithReinforcement,** [in] EDisplacement **Component,** [i/o] RDisplacementValues **Displacement,** [out] double **PosX,** [out] long **LoadCaseOrCombinationId,** [out] long **LoadLevel**)

LineId	*line index (0 < LineId ≤ AxisVMModel.Lines.Count)*
SectionId	*section index (0 < SectionId ≤ AxisVMModel.Lines[LineId].SectionCount)*
MinMaxType	*minmax type*
AnalysisType	*analysis type*

WithReinforcement	*determines which displacements results should be read. Considering reinforcement (lbTrue) or not (lbFalse) (this setting is taken into account only if AnalysisType = atNonLinearStatic)*
Component	*displacement component*
Displacement	*displacement 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 results. The EnvelopUID is specified by the global property. Similar to GetEnvelopeLineDisplacement, but retrieves detailed combination data instead of a mere string. Returns LineId or an error code (errDatabaseNotReady or see EDisplacementError).

---

long **GetCriticalLineDisplacement** ([in] long **LineId,** [in] long **SectionId,** [in] EMinMaxType **MinMaxType,** [in] ECombinationType **CombinationType,** [in] EAnalysisType **AnalysisType,** [in] ELongBoolean **WithReinforcement,** [in] EDisplacement **Component,** [i/o] RDisplacementValues **Displacement,** [out] double **PosX,** [out] BSTR **Combination**)

LineId	*line index (0 < LineId ≤ AxisVMModel.Lines.Count)*
SectionId	*section index (0 < SectionId ≤ AxisVMModel.Lines[LineId].SectionCount)*
MinMaxType	*minmax type*
CombinationType	*combination type (mostly for Eurocode only)*
AnalysisType	*analysis type*
WithReinforcement	*determines which displacements results should be read. Considering reinforcement (lbTrue) or not (lbFalse) (this setting is taken into account only if AnalysisType = atNonLinearStatic)*
Component	*displacement component*
Displacement	*displacement 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 displacements at a section of a line element. Displacement contains the result of the critical combination in which Component is maximal or minimal. Returns LineId or an error code (errDatabaseNotReady or see EDisplacementError).

---

long **GetCriticalLineDisplacement2** ([in] long **LineId,** [in] long **SectionId,** [in] EMinMaxType **MinMaxType,** [in] ECombinationType **CombinationType,** [in] EAnalysisType **AnalysisType,** [in] ELongBoolean **WithReinforcement,** [in] EDisplacement **Component,** [i/o] RDisplacementValues **Displacement,** [out] double **PosX,** [out] ECombinationType **CriticalCombinationType,** [out] SAFEARRAY(double) **Factors,** [out] SAFEARRAY(long) **LoadCaseIds**)

LineId	*line index (0 < LineId ≤ AxisVMModel.Lines.Count)*
SectionId	*section index (0 < SectionId ≤ AxisVMModel.Lines[LineId].SectionCount)*
MinMaxType	*minmax type*
CombinationType	*combination type (mostly for Eurocode only)*
AnalysisType	*analysis type*
WithReinforcement	*determines which displacements results should be read. Considering reinforcement (lbTrue) or not (lbFalse) (this setting is taken into account only if AnalysisType = atNonLinearStatic)*
Component	*displacement component*
Displacement	*displacement results*
PosX	*position of SectionId in m according to the local x direction*
CriticalCombinationType	*combination type corresponding to critical load combination*
Factors	*load factors of the critical load combination*
LoadCaseIds	*load case indexes of the critical load combination*

Retrieves critical displacements at a section of a line element. Displacement contains the result of the critical combination in which Component is maximal or minimal. Similar to GetCriticalLineDisplacement, but retrieves detailed combination data instead of a mere string. Returns LineId or an error code (errDatabaseNotReady or see EDisplacementError).

---

long **LineDisplacementByLoadCaseId** ([in] long **LineId,** [in] long **SectionId,**
[i/o] RDisplacementValues **Displacement,** [out] double **PosX,** [out] BSTR **Combination**)

LineId	*line index (0 < LineId ≤ AxisVMModel.Lines.Count)*
SectionId	*section index (0 < SectionId ≤ AxisVMModel.Lines[LineId].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 Uses the following global properties : LoadCaseId, LoadLevelOrModeShapeOrTimeStep, AnalysisType, WithReinforcement. Returns LineId or an error code (errDatabaseNotReady or see EDisplacementError).

---

long **LineDisplacementByLoadCombinationId** ([in] long **LineId,** [in] long **SectionId,**
[i/o] RDisplacementValues **Displacement,** [out] double **PosX,** [out] BSTR **Combination**)

LineId	*line index (0 < LineId ≤ AxisVMModel.Lines.Count)*
SectionId	*section index (0 < SectionId ≤ AxisVMModel.Lines[LineId].SectionCount)*
Displacement	*displacement results*
PosX	*position of SectionId in m according to the local x direction*
Combination	*name of the load combination*

Retrieves displacements at a section of a line element. Uses the following global properties : LoadCombinationId, LoadLevelOrModeShapeOrTimeStep, AnalysisType, WithReinforcement. Returns LineId or an error code (errDatabaseNotReady or see EDisplacementError).

---

long **EnvelopeLineDisplacement** ([in] long **LineId,** [in] long **SectionId,**
[i/o] RDisplacementValues **Displacement,** [out] double **PosX,** [out] BSTR **Combination**)

LineId	*line index (0 < LineId ≤ AxisVMModel.Lines.Count)*
SectionId	*section index (0 < SectionId ≤ AxisVMModel.Lines[LineId].SectionCount)*
Displacement	*displacement 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*

Read envelope results at a section of a line element. Uses the following global properties : EnvelopUID, MinMaxType, AnalysisType, WithReinforcement, Component. Returns LineId or an error code (errDatabaseNotReady or see EDisplacementError).

---

long **EnvelopeLineDisplacement2** ([in] long **LineId,** [in] long **SectionId,**
[i/o] RDisplacementValues **Displacement,** [out] double **PosX,** [out] long
LoadCaseOrCombinationId, [out] long **LoadLevel**)

LineId	*line index (0 < LineId ≤ AxisVMModel.Lines.Count)*
SectionId	*section index (0 < SectionId ≤ AxisVMModel.Lines[LineId].SectionCount)*
Displacement	*displacement 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 results. Uses the following global properties : EnvelopUID, MinMaxType, AnalysisType, WithReinforcement, Component. Similar to EnvelopeLineDisplacement, but retrieves detailed combination data instead of a mere string. Returns LineId or an error code (errDatabaseNotReady or see EDisplacementError).

---

long **CriticalLineDisplacement** ([in] long **LineId,** [in] long **SectionId,**
[i/o] RDisplacementValues **Displacement,** [out] double **PosX,** [out] BSTR **Combination**)

LineId	*line index (0 < LineId ≤ AxisVMModel.Lines.Count)*
SectionId	*section index (0 < SectionId ≤ AxisVMModel.Lines[LineId].SectionCount)*
Displacement	*displacement 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 displacements at a section of a line element. Uses the following global properties : MinMaxType, CombinationType, AnalysisType, WithReinforcement, Component. Displacement contains the result of the critical combination in which Component is maximal or minimal. Returns LineId or an error code (errDatabaseNotReady or see EDisplacementError).

---

long **CriticalLineDisplacement2** ([in] long **LineId,** [in] long **SectionId,**
[i/o] RDisplacementValues **Displacement,** [out] double **PosX,**
[out] ECombinationType **CriticalCombinationType,** [out] SAFEARRAY(double) **Factors,**
[out] SAFEARRAY(long) **LoadCaseIds**)

LineId	*line index (0 < LineId ≤ AxisVMModel.Lines.Count)*
SectionId	*section index (0 < SectionId ≤ AxisVMModel.Lines[LineId].SectionCount)*
Displacement	*displacement results*
PosX	*position of SectionId in m according to the local x direction*
CriticalCombinationType	*combination type corresponding to critical load combination*
Factors	*load factors of the critical load combination*
LoadCaseIds	*load case indexes of the critical load combination*

Retrieves critical displacements at a section of a line element. Uses the following global properties : MinMaxType, CombinationType, AnalysisType, WithReinforcement, Component. Displacement contains the result of the critical combination in which Component is maximal or minimal. Similar to CriticalLineDisplacement, but retrieves detailed combination data instead of a mere string. Returns LineId or an error code (errDatabaseNotReady or see EDisplacementError).

Single ELEMENT reader functions

long **GetLineDisplacementsByLoadCaseId** ([in] long **LineId,** [in] long **LoadCaseId,** [in] long **LoadLevelOrTimeStep,** [in] EAnalysisType **AnalysisType,** [in] ELongBoolean **WithReinforcement,** [i/o] SAFEARRAY(RDisplacementValues) **Displacements,** [out] SAFEARRAY(double) **PosX**)

LineId	*line index (0 < LineId ≤ AxisVMModel.Lines.Count)*
LoadCaseId	*load case index (0 < LoadCaseId ≤ AxisVMModel.LoadCases.Count)*
LoadLevelOrTimeStep	*load level or time step, according to the type of analysis. See LoadLevelOrModeShapeOrTimeStep parameter*
AnalysisType	*analysis type*
WithReinforcement	*determines which displacements results should be read. Considering reinforcement (lbTrue) or not (lbFalse) (this setting is taken into account only if AnalysisType = atNonLinearStatic)*
Displacements	*array of line displacements for the line element. Length of the array is SectionCount[LineId]*
PosX	*array of cross-section positions in m according to the local x direction Length of the array is SectionCount[LineId]*

Retrieves displacements for each cross-section of a given element Returns the total number of cross-sections or an error code (errDatabaseNotReady or see EDisplacementError).

---

long **GetLineDisplacementsByLoadCombinationId** ([in] long **LineId,** [in] long **LoadCombinationId,** [in] long **LoadLevel,** [in] EAnalysisType **AnalysisType,** [in] ELongBoolean **WithReinforcement,** [i/o] SAFEARRAY(RDisplacementValues) **Displacements,** [out] SAFEARRAY(double) **PosX**)

LineId	*line index (0 < LineId ≤ AxisVMModel.Lines.Count)*
LoadCombinationId	*load combination index (0 < LoadCombinationId ≤ AxisVMModel.LoadCombinations.Count)*
LoadLevel	*load level according to the type of analysis. See LoadLevelOrModeShapeOrTimeStep parameter*
AnalysisType	*analysis type*
WithReinforcement	*determines which displacements results should be read. Considering reinforcement (lbTrue) or not (lbFalse) (this setting is taken into account only if AnalysisType = atNonLinearStatic)*

Displacements *array of line displacements for the line element.*
Length of the array is SectionCount[LineId]
PosX *array of cross-section positions in m according to the local x direction*
Length of the array is SectionCount[LineId]

Retrieves displacements for each cross-section of a given element. Returns the total number of cross-sections or an error code (errDatabaseNotReady or see EDisplacementError).

---

long **GetEnvelopeLineDisplacements** ([in] long **LineId,** [in] EMinMaxType **MinMaxType,** [in] EAnalysisType **AnalysisType,** [in] ELongBoolean **WithReinforcement,** [in] EDisplacement **Component,** [i/o] SAFEARRAY(RDisplacementValues) **Displacements,** [out] SAFEARRAY(double) **PosX**)

LineId *line index (0 < LineId ≤ AxisVMModel.Lines.Count)*
MinMaxType *minmax type*
AnalysisType *analysis type*
WithReinforcement *determines which displacements results should be read. Considering reinforcement (lbTrue) or not (lbFalse) (this setting is taken into account only if AnalysisType = atNonLinearStatic)*
Component *displacement component*
Displacements *array of envelope line displacements for the line element.*
Length of the array is SectionCount[LineId]
PosX *array of cross-section positions in m according to the local x direction*
Length of the array is SectionCount[LineId]

Retrieves envelope results. The EnvelopUID is specified by the global EnvelopUID property. Displacements array contains the result of the load case or combination in which Component is maximal or minimal for the given section. To determine the load case or combination for a specific section, use the single section reader function for that section (GetEnvelopeLineDisplacement or GetEnvelopeLineDisplacement2). Returns the total number of cross-sections or an error code (errDatabaseNotReady or see EDisplacementError).

---

long **GetCriticalLineDisplacements** ([in] long **LineId,** [in] EMinMaxType **MinMaxType,** [in] ECombinationType **CombinationType,** [in] EAnalysisType **AnalysisType,** [in] ELongBoolean **WithReinforcement,** [in] EDisplacement **Component,** [i/o] SAFEARRAY(RDisplacementValues) **Displacements,** [out] SAFEARRAY(double) **PosX**)

LineId *line index (0 < LineId ≤ AxisVMModel.Lines.Count)*
MinMaxType *minmax type*
CombinationType *combination type (mostly for Eurocode only)*
AnalysisType *analysis type*
WithReinforcement *determines which displacements results should be read. Considering reinforcement (lbTrue) or not (lbFalse) (this setting is taken into account only if AnalysisType = atNonLinearStatic)*
Component *displacement component*
Displacements *array of critical line displacements for the line element.*
Length of the array is SectionCount[LineId]
PosX *array of cross-section positions in m according to the local x direction*
Length of the array is SectionCount[LineId]

Retrieves critical displacements in each cross-section of a given element. Displacements array contains the result of the load case or combination in which Component is maximal or minimal for the given section. To determine the load combination for a specific section, use the single section reader function for that section (GetCriticalLineDisplacement or GetCriticalLineDisplacement2). Returns the total number of cross-sections or an error code (errDatabaseNotReady or see EDisplacementError).

---

long **LineDisplacementsByLoadCaseId** ([in] long **LineId,** [i/o] SAFEARRAY(RDisplacementValues) **Displacements,** [out] SAFEARRAY(double) **PosX**)

LineId *line index (0 < LineId ≤ AxisVMModel.Lines.Count)*
Displacements *array of line displacements for the line element.*
Length of the array is SectionCount[LineId]
PosX *array of cross-section positions in m according to the local x direction*
Length of the array is SectionCount[LineId]

Retrieves displacements for each cross-section of a given element. Uses the following global properties : LoadCaseId, LoadLevelOrModeShapeOrTimeStep , AnalysisType, WithReinforcement. Returns the total number of cross-sections or an error code (errDatabaseNotReady or see EDisplacementError).

---

long **LineDisplacementsByLoadCombinationId** ([in] long **LineId,**
[i/o] SAFEARRAY(RDisplacementValues) **Displacements,** [out] SAFEARRAY(double) **PosX**)

LineId	*line index (0 < LineId ≤ AxisVMModel.Lines.Count)*
Displacements	*array of line displacements for the line element.* *Length of the array is SectionCount[LineId]*
PosX	*array of cross-section positions in m according to the local x direction* *Length of the array is SectionCount[LineId]*

Retrieves displacements for each cross-section of a given element. Uses the following global properties : LoadCombinationId, LoadLevelOrModeShapeOrTimeStep , AnalysisType, WithReinforcement. Returns the total number of cross-sections or an error code (errDatabaseNotReady or see EDisplacementError).

---

long **EnvelopeLineDisplacements** ([in] long **LineId,**
[i/o] SAFEARRAY(RDisplacementValues) **Displacements,** [out] SAFEARRAY(double) **PosX**)

LineId	*line index (0 < LineId ≤ AxisVMModel.Lines.Count)*
Displacements	*array of envelope line displacements for the line element.* *Length of the array is SectionCount[LineId]*
PosX	*array of cross-section positions in m according to the local x direction* *Length of the array is SectionCount[LineId]*

Retrieves envelope results. Uses the following global properties : EnvelopUID, MinMaxType, AnalysisType, WithReinforcement, Component. To determine the load case or combination for a specific section, use the single section reader function for that section (EnvelopeLineDisplacement or EnvelopeLineDisplacement2). Returns the total number of cross-sections or an error code (errDatabaseNotReady or see EDisplacementError).

---

long **CriticalLineDisplacements** ([in] long **LineId,**
[i/o] SAFEARRAY(RDisplacementValues) **Displacements,** [out] SAFEARRAY(double) **PosX**)

LineId	*line index (0 < LineId ≤ AxisVMModel.Lines.Count)*
Displacements	*array of critical line displacements for the line element.* *Length of the array is SectionCount[LineId]*
PosX	*array of cross-section positions in m according to the local x direction* *Length of the array is SectionCount[LineId]*

Retrieves critical displacements in each cross-section of a given element. Uses the following global properties : MinMaxType, CombinationType, AnalysisType, WithReinforcement, Component. Displacements contains the result of the critical combination in which Component is maximal or minimal. To determine the combination for a specific section, use the single section reader function for that section (CriticalLineDisplacement or CriticalLineDisplacement2). Returns the total number of cross-sections or an error code (errDatabaseNotReady or see EDisplacementError).

Multiple element reader functions

long **GetAllLineDisplacementsByLoadCaseId** ([in] long **LoadCaseId,** [in] long **LoadLevelOrTimeStep,** [in] EAnalysisType **AnalysisType,** [in] ELongBoolean **WithReinforcement,** [out] SAFEARRAY(long) **SectionCounts,** [i/o] SAFEARRAY(RDisplacementValues) **Displacements,** [out] SAFEARRAY(double) **PosX**)

LoadCaseId	*load case index* *(0 < LoadCaseId ≤ AxisVMModel.LoadCases.Count)*
LoadLevelOrTimeStep	*load level or time step, according to the type of analysis. See LoadLevelOrModeShapeOrTimeStep parameter*
AnalysisType	*analysis type*
WithReinforcement	*determines which displacements results should be read. Considering reinforcement (lbTrue) or not (lbFalse) (this setting is taken into account only if AnalysisType = atNonLinearStatic)*
SectionCounts	*array containing the section counts of line elements.* *Length of the array = AxisVMModel.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. Returns the total number of displacement records in the array or an error code (errDatabaseNotReady or see EDisplacementError).

---

long **GetAllLineDisplacementsByLoadCombinationId** ([in] long **LoadCombinationId,** [in] long **LoadLevel,** [in] EAnalysisType **AnalysisType,** [in] ELongBoolean **WithReinforcement,** [out] SAFEARRAY(long) **SectionCounts,** [i/o] SAFEARRAY(RDisplacementValues) **Displacements,** [out] SAFEARRAY(double) **PosX**)

LoadCombinationId *load combination index*
(0 < LoadCombinationId ≤ AxisVMModel.LoadCombinations.Count)
LoadLevel *load level, according to the type of analysis. See LoadLevelOrModeShapeOrTimeStep parameter*
AnalysisType *analysis type*
WithReinforcement *determines which displacements results should be read. Considering reinforcement (lbTrue) or not (lbFalse) (this setting is taken into account only if AnalysisType = atNonLinearStatic)*
SectionCounts *array containing the section counts of line elements.*
Length of the array = AxisVMModel.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 Returns the total number of displacement records in the array or an error code (errDatabaseNotReady or see EDisplacementError).

---

long **GetAllEnvelopeLineDisplacements** ([in] EMinMaxType **MinMaxType,** [in] EAnalysisType **AnalysisType,** [in] ELongBoolean **WithReinforcement,** [in] EDisplacement **Component,** [out] SAFEARRAY(long) **SectionCounts,**
[i/o] SAFEARRAY(RDisplacementValues) **Displacements,** [out] SAFEARRAY(double) **PosX**)

MinMaxType *minmax type*
AnalysisType *analysis type*
WithReinforcement *determines which displacements results should be read. Considering reinforcement (lbTrue) or not (lbFalse) (this setting is taken into account only if AnalysisType = atNonLinearStatic)*
Component *displacement component*
SectionCounts *array containing the section counts of line elements.*
Length of the array = AxisVMModel.Lines.Count
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
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 results of all line elements. The EnvelopUID is specified by the global EnvelopUID property. Displacements array contains the result of the load case or combination in which Component is maximal or minimal. To determine the load case or combination for a specific section, use the single section reader function for that section (EnvelopeLineDisplacement or EnvelopeLineDisplacement2). Returns the total number of cross-sections or an error code (errDatabaseNotReady or see EDisplacementError).

---

long **GetAllCriticalLineDisplacements** ([in] EMinMaxType **MinMaxType,** [in] ECombinationType **CombinationType,** [in] EAnalysisType **AnalysisType,** [in] ELongBoolean **WithReinforcement,** [in] EDisplacement **Component,** [out] SAFEARRAY(long) **SectionCounts,**
[i/o] SAFEARRAY(RDisplacementValues) **Displacements,** [out] SAFEARRAY(double) **PosX**)

MinMaxType *minmax type*
CombinationType *combination type (mostly for Eurocode only)*

AnalysisType *analysis type*
WithReinforcement *determines which displacements results should be read. Considering reinforcement (lbTrue) or not (lbFalse) (this setting is taken into account only if AnalysisType = atNonLinearStatic)*
Component *displacement component*
SectionCounts *array containing the section counts of line elements.*
Length of the array = AxisVMModel.Lines.Count
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
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 displacements of all line elements. Displacements array contains the result of the critical combination in which Component is maximal or minimal. To determine the load combination for a specific section, use the single section reader function for that section (GetCriticalLineDisplacement or GetCriticalLineDisplacement2). Returns the total number of cross-sections or an error code (errDatabaseNotReady or see EDisplacementError).

---

long **AllLineDisplacementsByLoadCaseId** ([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 = AxisVMModel.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 Uses the following global properties : LoadCaseId, LoadLevelOrModeShapeOrTimeStep, AnalysisType, WithReinforcement. 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 = AxisVMModel.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. Uses the following global properties : LoadCombinationId, LoadLevelOrModeShapeOrTimeStep, AnalysisType, WithReinforcement. 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**)

SectionCounts *array containing the section counts of line elements.*
Length of the array = AxisVMModel.Lines.Count
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
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 results of all line elements. Uses the following global properties : EnvelopUID, MinMaxType, AnalysisType, WithReinforcement, Component. Displacements array contains the result of the load case or combination in which Component is maximal or minimal. To determine the load case or combination for a specific section, use the single section reader function for that section (EnvelopeLineDisplacement or EnvelopeLineDisplacement2). Returns the total number of cross-sections or an error code (errDatabaseNotReady or see EDisplacementError).

---

long **AllCriticalLineDisplacements** ([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 = AxisVMModel.Lines.Count*
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*
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 displacements of all line elements. Uses the following global properties : MinMaxType, CombinationType, AnalysisType, WithReinforcement, Component. Displacements array contains the result of the critical combination in which Component is maximal or minimal. To determine the combination for a specific section, use the single section reader function for that section (CriticalLineDisplacement or CriticalLineDisplacement2). Returns the total number of cross-sections or an error code (errDatabaseNotReady or see EDisplacementError).

Multiple BLOCK reader functions

long **GetLineDisplacementsForResultBlocks** ([in] long **LineId,** [in] long **SectionId,** [in] EAnalysisType **AnalysisType,** [in] ELongBoolean **WithReinforcement,** [i/o] SAFEARRAY(RResultBlockInfo) **ResultBlockInfo,**
[i/o] SAFEARRAY(RDisplacementValues) **Displacements,** [out] double **PosX**)

LineId	*line index (0 < LineId ≤ AxisVMModel.Lines.Count)*
SectionId	*section index (0 < SectionId ≤ AxisVMModel.Lines[LineId].SectionCount)*
AnalysisType	*analysis type*
WithReinforcement	*determines which displacements results should be read. Considering reinforcement (lbTrue) or not (lbFalse) (this setting is taken into account only if AnalysisType = atNonLinearStatic)*
ResultBlockInfo	*array of description records for result blocks*
Displacements	*displacement results for all result blocks*
PosX	*cross-section position according to the local x direction of each line element*

Retrieves displacements at the given line and cross-section in all result blocks consecutively. Returns the common length of ResultBlockInfo, Displacements and PosX arrays or an error code (errDatabaseNotReady or see EDisplacementError).

---

long **LineDisplacementsForResultBlocks** ([in] long **LineId,** [in] long **SectionId,**
[i/o] SAFEARRAY(RResultBlockInfo) **ResultBlockInfo,**
[i/o] SAFEARRAY(RDisplacementValues) **Displacements,** [out] double **PosX**)

LineId	*line index (0 < LineId ≤ AxisVMModel.Lines.Count)*
SectionId	*section index (0 < SectionId ≤ AxisVMModel.Lines[LineId].SectionCount)*
ResultBlockInfo	*array of description records for result blocks*
Displacements	*displacement results for all result blocks*
PosX	*cross-section position according to the local x direction of each line element*

Retrieves displacements at the given line and cross-section in all result blocks consecutively. Uses the following global properties : AnalysisType, WithReinforcement. Returns the common length of ResultBlockInfo, Displacements and PosX arrays or an error code (errDatabaseNotReady or see EDisplacementError).

Single ELEMENT reader functions

long **VirtualBeamOrStripDisplacementsByLoadCaseId** ([in] long **Index,** [in] long **ChainId,** [i/o] SAFEARRAY(RDisplacementValues) **Displacements,** [i/o] SAFEARRAY(RPoint3d)* **Pos**)

Index *virtual beam/strip index (0 < Index ≤ AxisVMModel.VirtualBeams.Count)*

ChainId *virtual beam's/strip's chain index (0 < ChainId ≤ AxisVMModel.VirtualBeams.ChainCount[Index])*

Displacements *array of virtual beam/strip displacements Length of the array is AxisVMModel.VirtualBeams.SectionCount[Index, ChainId]*

Pos *array of section positions in m according to the global coordinate system Length of the array is AxisVMModel.VirtualBeams.SectionCount[Index, ChainId]*

Retrieves displacements for each sections of a given chain of a virtual beam/strip according to the LoadCaseId (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 *virtual beam/strip index (0 < Index ≤ AxisVMModel.VirtualBeams.Count)*

ChainId *virtual beam's/strip's chain index (0 < ChainId ≤ AxisVMModel.VirtualBeams.ChainCount[Index])*

Displacements *array of virtual beam/strip displacements Length of the array is AxisVMModel.VirtualBeams.SectionCount[Index, ChainId]*

Pos *array of section positions in m according to the global coordinate system Length of the array is AxisVMModel.VirtualBeams.SectionCount[Index, ChainId]*

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 *virtual beam/strip index (0 < Index ≤ AxisVMModel.VirtualBeams.Count)*

ChainId *virtual beam's/strip's chain index (0 < ChainId ≤ AxisVMModel.VirtualBeams.ChainCount[Index])*

Displacements *array of virtual beam/strip displacements Length of the array is AxisVMModel.VirtualBeams.SectionCount[Index, ChainId]*

Pos *array of section positions in m according to the global coordinate system Length of the array is AxisVMModel.VirtualBeams.SectionCount[Index, ChainId]*

Combination *load cases or combinations in which Component has its minimum or maximum*

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/o] SAFEARRAY(RDisplacementValues) **Displacements,** [i/o] SAFEARRAY(RPoint3d)* **Pos,** [out] SAFEARRAY(long)* **LoadCaseOrCombinationIds,** [out] SAFEARRAY(long)* **LoadLevels**)

Index	*virtual beam/strip index (0 < Index ≤ AxisVMModel.VirtualBeams.Count)*
ChainId	*virtual beam's/strip's chain index (0 < ChainId ≤ AxisVMModel.VirtualBeams.ChainCount[Index])*
Displacements	*array of virtual beam/strip displacements Length of the array is AxisVMModel.VirtualBeams.SectionCount[Index, ChainId]*
Pos	*array of section positions in m according to the global coordinate system 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*

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).

---

long **CriticalVirtualBeamOrStripDisplacements** ([in] long **Index,** [in] long **ChainId,** [i/o] SAFEARRAY(RDisplacementValues) **Displacements,** [i/o] SAFEARRAY(RPoint3d)* **Pos,** [out] SAFEARRAY(BSTR)* **Combination**)

Index	*virtual beam/strip index (0 < Index ≤ AxisVMModel.VirtualBeams.Count)*
ChainId	*virtual beam's/strip's chain index (0 < ChainId ≤ AxisVMModel.VirtualBeams.ChainCount[Index])*
Displacements	*array of virtual beam/strip displacements Length of the array is AxisVMModel.VirtualBeams.SectionCount[Index, ChainId]*
Pos	*array of section positions in m according to the global coordinate system Length of the array is AxisVMModel.VirtualBeams.SectionCount[Index, ChainId]*
Combination	*load cases or combinations in which Component has its minimum or maximum*

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).

---

long **CriticalVirtualBeamOrStripDisplacement** ([in] long **Index,** [in] long **ChainId,** [in] long **SectionId,** [i/o] RDisplacementValues **Displacements,** [i/o] RPoint3d **Pos,** [out] BSTR **Combination**)

Index	*virtual beam/strip index (0 < Index ≤ AxisVMModel.VirtualBeams.Count)*
ChainId	*virtual beam's/strip's chain index (0 < ChainId ≤ AxisVMModel.VirtualBeams.ChainCount[Index])*
SectionId	*chain's section index (0 < ChainId ≤ AxisVMModel.VirtualBeams.SectionCount[Index, ChainId])*
Displacements	*virtual beam displacements*
Pos	*section position in m according to the global coordinate system*
Combination	*critical combination in which Component has its minimum or maximum*

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).

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)* **LoadCaseIds**)

Index	*virtual beam/strip index (0 < Index ≤ AxisVMModel.VirtualBeams.Count)*
ChainId	*virtual beam's/strip's chain index (0 < ChainId ≤ AxisVMModel.VirtualBeams.ChainCount[Index])*
SectionId	*chain's section index (0 < ChainId ≤ AxisVMModel.VirtualBeams.SectionCount[Index, ChainId])*
Displacements	*virtual beam displacements*
Pos	*section position in m according to the global coordinate system*
CriticalCombinationType	*combnation type corresponding to critical load combination*
Factors	*load factors of the critical load combination*
LoadCaseIds	*load case indexes of the critical load combination*

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).

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	*name of the file with extension emf*
ID	*index of the virtual beam or strip*
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*
RelativeToLeft	*relative deformations to the left endpoint (see the notes below)*
RelativeToRight	*relative deformations to the 427ight endpoint (see the notes below)*
Position	*position of the investigated cross section along the virtual beam or strip*
ColourMode	*colour mode*

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	*name of the file with extension emf*
ID	*index of the virtual beam or strip*
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*
RelativeToLeft	*relative deformations to the left endpoint (see the notes below)*
RelativeToRight	*relative deformations to the 427ight endpoint (see the notes below)*
Position	*position of the investigated cross section along the virtual beam or strip*
ColourMode	*colour mode*

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	*name of the file with extension emf*
ID	*index of the virtual beam or strip*
LoadCaseId	*load case 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 427ight 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 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] ELongBoolean **RelativeToLeft,** [in] ELongBoolean **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 ≤ LoadLevelCount)* *for timestep (0 < LoadLevelOrTimeStep ≤ 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 428ight 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 **EndReleasesDeformationsByLoadCaseId** ([in] long **LineId,**
[i/o] RDisplacementValues **StartDisplacement,** [i/o] RDisplacementValues **EndDisplacement,**
[out] BSTR **StartCombination,** [out] BSTR **EndCombination**)

LineId	*line index (0 < LineId ≤ AxisVMModel.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*

Retrieves deformations of end releases of a line element according to the LoadCaseId (and LoadLevelOrModeShapeOrTimeStep) property. Returns LineId or an error code (errDatabaseNotReady , deInvalidCombinationOfLoadCaseAndLoadLevel, deInvalidLineType, deLoadCaseIndexOutofBounds, deLineIndexOutOfBounds or deInvalidAnalysisType).

---

long **EndReleasesDeformationsByByLoadCombinationId** ([in] long **LineId,**
[i/o] RDisplacementValues **StartDisplacement,** [i/o] RDisplacementValues **EndDisplacement**,
[out] BSTR **StartCombination,** [out] BSTR **EndCombination**)

LineId	*line index (0 < LineId ≤ AxisVMModel.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*

Retrieves deformations of end releases according to the LoadCombinationId (and LoadLevelOrModeShapeOrTimeStep) property. Returns LineId or an error code (errDatabaseNotReady , deInvalidCombinationOfLoadCaseAndLoadLevel, deInvalidLineType, deLoadCombinationIndexOutofBounds, deLineIndexOutOfBounds or deInvalidAnalysisType).

---

long **EnvelopeEndReleasesDeformations** ([in] long **LineId,** [i/o] RDisplacementValues
StartDisplacement, [i/o] RDisplacementValues **EndDisplacement,**
[out] BSTR **StartCombination,** [out] BSTR **EndCombination**)

LineId	*line index (0 < LineId ≤ AxisVMModel.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*

Retrieves envelope deformations of end releases. Envelope is identified by Component,MinMaxType and EnvelopeUID properties . Returns LineId or an error code (errDatabaseNotReady , deInvalidLineType, deLineIndexOutOfBounds or deInvalidAnalysisType).

---

long **CriticalEndReleasesDeformations** ([in] long **LineId,** [i/o] RDisplacementValues
StartDisplacement, [i/o] RDisplacementValues **EndDisplacement,**
[out] BSTR **StartCombination,** [out] BSTR **EndCombination**)

LineId	*line index (0 < LineId ≤ AxisVMModel.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*

Retrieves critical deformations of end releases. Critical combination is identified by Component and MinMaxType properties (e.g. Ez min).Returns LineId or an error code (errDatabaseNotReady , deCombinationTypeNotValidForCurrentNationalDesignCode, deInvalidLineType, deLineIndexOutOfBounds or deInvalidAnalysisType).

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* AxisVMModel.Lines.Count)*

EndDisplacements *displacement results for all end releases at the end of the lines (Length of the array = 2* AxisVMModel.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* AxisVMModel.Lines.Count)*

EndDisplacements *displacement results for all end releases at the end of the lines (Length of the array = 2* AxisVMModel.Lines.Count)*

Retrieves all deformations of end releases according 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* AxisVMModel.Lines.Count)*

EndDisplacements *displacement results for all end releases at the end of the lines (Length of the array = 2* AxisVMModel.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* AxisVMModel.Lines.Count)*

EndDisplacements *displacement results for all end releases at the end of the lines (Length of the array = 2* AxisVMModel.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] ELongBoolean **WithReinforcement,** [out] SAFEARRAY(RDisplacementValues) **Displacements,** [out] SAFEARRAY(double) **PosX**)

MemberID	*member index (0 < MemberId ≤ AxisVMModel.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	*determines which displacements results should be read. Considering reinforcement (lbTrue) or not (lbFalse) (this setting is taken into account 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] ELongBoolean **WithReinforcement,** [out] SAFEARRAY(RDisplacementValues) **Displacements,** [out] SAFEARRAY(double) **PosX**)

MemberID	*member index (0 < MemberId ≤ AxisVMModel.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(RDisplacementValues) **Displacements,**
[out] SAFEARRAY(long) **LoadCaseOrCombinationIds,** [out] SAFEARRAY(long) **LoadLevels,**
[out] SAFEARRAY(double) **PosX**)

MemberID	*member index (0 < MemberId ≤ 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 > IAxisVMLoadcases.count then Load combination index = LoadCaseOrCombinationId – IAxisVMLoadcases.count*
LoadLevels	*array of load levels*
PosX	*array of cross-section positions in m according to the local x direction*

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).

---

long **GetCriticalMemberDisplacements** ([in] long **MemberID,** [in] EMinMaxType **MinMaxType,**
[in] ECombinationType **CombinationType,** [in] EAnalysisType **AnalysisType,**
[in] EDisplacement **Component,** [out] SAFEARRAY(RDisplacementValues) **Displacements,**
[out] SAFEARRAY(double) **PosX**)

MemberID	*member index (0 < MemberId ≤ 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*

Retrieves critical displacements in each cross-section of a member. Critical combination is identified by Component and MinMaxType properties (e.g. lsSmin). 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).

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	*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*
RelativeToLeft	*relative deformations to the left endpoint (see the notes below)*
RelativeToRight	*relative deformations to the 433ight endpoint (see the notes below)*
Position	*position of the investigated cross section along the member*
ColourMode	*colour mode*

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	*name of the file with extension emf*
MemberId	*member index*
EnvelopeUID	*unique envelope index*
AnalysisType	*analysis type*
Width	*picture's size in pixel (minimally acceptable value is 640)*
Height	*picture's size in pixel (minimally acceptable value is 580)*
EnvelopeOnly	*only Envelope*
RelativeToLeft	*relative deformations to the left endpoint (see the notes below)*
RelativeToRight	*relative deformations to the 433ight endpoint (see the notes below)*
Position	*position of the investigated cross section along the member*
ColourMode	*colour mode*

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**)

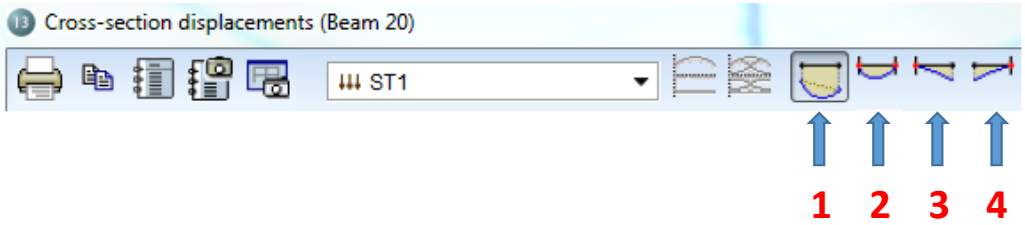
FileName	*name of the file with extension emf*
MemberId	*member index*
LoadCaseId	*load case index*
LoadLevelOrTimeStep	*for load level (0 < LoadLevelOrTimeStep ≤ LoadLevelCount) for timestep (0 < LoadLevelOrTimeStep ≤ TimeStepCount)*
AnalysisType	*analysis type*
Width	*picture's size in pixel (minimal 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 434ight endpoint (see the notes below)*
Position	*position of the investigated cross section along the member*
ColourMode	*colour mode*

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	*name of the file with extension emf*
MemberId	*member index*
LoadCombinationId	*load combination index*
LoadLevelOrTimeStep	*for load level (0 < LoadLevelOrTimeStep ≤ LoadLevelCount) for timestep (0 < LoadLevelOrTimeStep ≤ 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 434ight endpoint (see the notes below)*
Position	*position of the investigated cross section along the member*
ColourMode	*colour mode*

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] ELongBoolean **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)*
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 valuesof the component will be read*
ELongBoolean	**UserCreep** *Returns lbTrue if nonlinear analysis results consider concrete creep. More here…*
ELongBoolean	**WithReinforcement** • *Get or set this to determine which nodal displacements results should be read. Considering reinforcement (lbTrue) or not (lbFalse) (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.

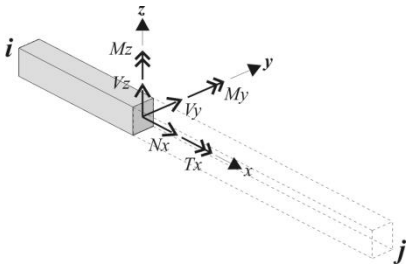
Error codes

enum **EForcesError** = {

feLineIndexOutOfBounds = –100001	*line index is out of bounds*
feLoadCaseIndexOutofBounds = –100002	*LoadCaseId is out of bounds*
feLoadCombinationIndexOutofBounds = –100003	*LoadCombinationId is out of bounds*
feNotValidLineType = –100004	*IAxisVMLine.LineType is not compatible with the reader function*
feSectionIndexOutOfBounds = –100005	*section index is out of bounds*
feCombinationTypeNotValidForCurrentNationalDesignCode = –100006	*CombinationType cannot be used for the selected design code or results not available for CombinationType. Use GetValidCombinationTypes to querry the valid Combination types.*
feCOMError = –100007	*internal COM error when calling GetRecordInfoFromGUIDs,SafeArrayCreateEx, SafeArrayAccesssData*
feLineHasNoSections = –100008	*line element has no cross-sections (e.g. link element)*
feNoValidLinesInTheModel = –100009	*no valid line elements in the model*
feSurfaceIndexOutOfBounds = –100010	*surface index is out of bounds*
feInvalidSurfaceVertexType = –100011	*surface vertex type is invalid*
feInvalidAnalysisType = –100012	*AnalysisType is incompatible with the function*
feInvalidCombinationOfLoadCaseAndLoadLevel = –100013	*no result is available for the given LoadCaseId and LoadLevelOrModeShape or this*
feInvalidCombinationOfLoadCombinationAndLoadLevel = –100014	*no result is available for the given LoadCombinationId and LoadLevelOrModeShape or this*
feNoResultBlocksInTheModel = –100015	*no result blocks in the model*
feNodeIndexOutOfBounds = –100016	*node index is out of bounds*
feNoSurfacesInTheModel = –100017	*no surface elements in the model*
feNodalSupportIndexOutOfBounds = –100018	*nodal support index is out of bounds*
feLineSupportIndexOutOfBounds = –100019	*line support index is out of bounds*
feNoNodalSupportsInTheModel = –100020	*no nodal supports in the model*
feNoLineSupportsInTheModel = –100021	*no line supports in the model*
feSurfaceSupportIndexOutOfBounds = –100022	*surface support index is out of bounds*
feNoSurfaceSupportsInTheModel = –100023	*no surface supports in the model*
feInvalidLineType = –100024	*line type is invalid*
feNoSpringsInTheModel = –100025	*no springs in the model*
feNoGapsInTheModel = –100026	*no gaps in the model*
feEdgeConnectionIndexOutOfBounds = –100027	*edge connection index is out of bounds*
feNoEdgeConnectionsInTheModel = –100028	*no edge connections in the model*
feLinkElementIndexOutOfBounds = –100029	*link element index is out of bounds*
feNoLinkElementsInTheModel = –100030	*no link elements in the model*
feMemberIndexOutOfBounds = –100031	*member index is out of bounds*
feInvalidEnvelopeUID = –100032	*EnvelopeUID is invalid*
feZeroValidLineNumber = -100033	*there is no selected line based on EConnectionToNodeType*
feVirtualBeamIndexOutOfBounds = -100034	*virtual beam index is out of bounds*
feVirtualBeamChainIndexOutOfBounds = -100035	*virtual beam's chain index is out of bounds*
feVirtualBeamSectionIndexOutOfBounds = -100036	*chain's section index is out of bounds*
feWindowIdNotValid = -100037	*the given WindowId is not valid*
feMembersSupportIndexOutOfBounds = -100037 }	*Member support index is out of range*

Enumerated types

enum **ELineForce** = {



lfNx = 0x00	*Nx normal force [kN]*
lfVy = 0x01	*Vy shear force [kN]* *(only for beams and ribs)*
lfVz = 0x02	*Vz shear force [kN]* *(only for beams and ribs)*
lfTx = 0x03	*Tx twisting moment [kNm]* *(only for beams and ribs)*
lfMy = 0x04	*My bending moment [kNm]* *(only for beams and ribs)*
lfMz = 0x05	*Mz bending moment [kNm]* *(only for beams and ribs)*
lfMyD = 0x06 }	*My bending momentconsidering rib eccentricity [kNm]* *(only for ribs)*

Line force component identifiers

enum **EVirtualBeamForce** = {

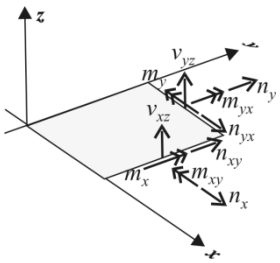
vbfNx = 0x00	*Nx normal force [kN]*
vbfVy = 0x01	*Vy shear force [kN]*
vbfVz = 0x02	*Vz shear force [kN]*
vbfTx = 0x03	*Tx twisting moment [kNm]*
vbfMy = 0x04	*My bending moment [kNm]*
vbfMz = 0x05}	*Mz bending moment [kNm]*

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)*

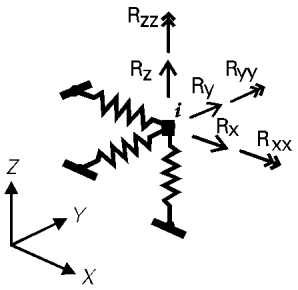
enum **ESurfaceForce** = {



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]*

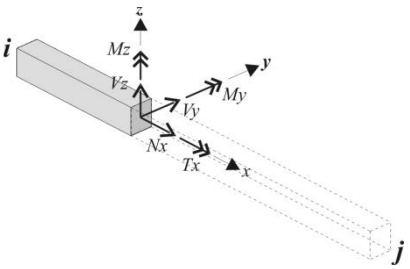
Surface support force component identifiers

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]*

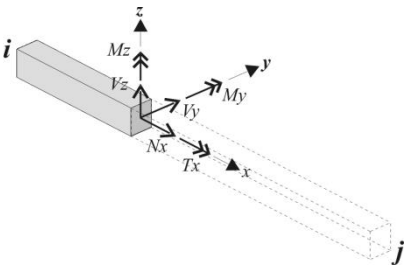
Spring force component identifiers

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 [kN/m] for LL links)
lefVy = 0x01	*Vy shear force* ([kN] for NN links and [kN/m] for LL links)
lefVz = 0x02	*Vz shear force* ([kN] for NN links and [kN/m] 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 } *Mz bending moment*
([kNm] for NN links and [kNm/m] for LL links)

enum **ECapacityCurveType** = {

cctImportantCharacteristics = 0x00 — *Instead of curve, typical characteristics are returned in array X in this order:*
Γ – transformation factor
m - mass of the equivalent SDOF system*
Fy - yield force of the equivalent SDOF system*
dm - maximal displacement of the equivalent bilinear system*
dy - yield displacement of the equivalent SDOF system*
T - natural period of vibration of the equivalent SDOF system*
det - target displacement of a perfectly elastic system*
dt - target displacement of the equivalent inelastic SDOF system*
dt - target displacement of the inelastic MDOF system ,
du - maximum considered displacement of the MDOF system (in accordance with the 85% capacity rule in Italy),
du - maximum considered displacement of the equivalent SDOF system (in accordance with the 85% capacity rule in Italy),*
dmax - maximum expected displacement of the MDOF system,
dmax - maximum expected displacement of the equivalent SDOF system*

cctMDOF = 0x01 — *MDOF capacity curve*
cctSDOF = 0x02 — *Equivalent SDOF capacity curve*
cctEquivalentBilinear = 0x03 — *Equivalent elastic bilinear capacity curve*
cctElasticADRS = 0x04 — *elastic ADRS spectrum*
cctInelasticADRS = 0x05 — *inelastic ADRS spectrum*
cctBilinearAD = 0x06 — *Plastic-elastic accelartion-displacement curve*
cctSDOFinADSpace = 0x07 — *Equivalent SDOF accelartion-displacement curve*
cctInitialPeriod = 0x08 — *Straight line corresponding to initial stiffness period*
}

enum **ESubsoilClass** = {

scA_Type1 = 0x01 — *Type 1 class A*
scB_Type1 = 0x02 — *Type 1 class B*
scC_Type1 = 0x03 — *Type 1 class C*
scD_Type1 = 0x04 — *Type 1 class D*
scE_Type1 = 0x05 — *Type 1 class E*
scA_Type2 = 0x06 — *Type 2 class A*
scB_Type2 = 0x07 — *Type 2 class B*
scC_Type2 = 0x08 — *Type 2 class C*
scD_Type2 = 0x09 — *Type 2 class D*
scE_Type2 = 0x10 — *Type 2 class E*
}

enum **ETopographicCategory** = {

tcT1 = 0x01 — *Category T1*
tcT2 = 0x02 — *Category T2*
tcT3 = 0x03 — *Category T3*
tcT4 = 0x04 — *Category T4*
}

enum **ESeismicComponentSumType** = {

scstUsual = 0x00	*simple sum considering +/- sign of seismic component as it is from the static analysis*
scstCritical = 0x01	*critical sum that results graeter value in absulate 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*
scstNAbsMyMz = 0x09	*it returns multiple results considering the absolute value of N and both +/- signs for My and Mz seismic components in summation*
scstNMy = 0x0A	*it returns multiple results considering both +/- signs for N and My seismic components in summation*
scstNMz = 0x0B	*it returns multiple results considering both +/- signs for N and Mz seismic components in summation*

}

Summation rule of seismic internal force components for line forces.

enum **ESupportSeismicComponentSumType**=

{

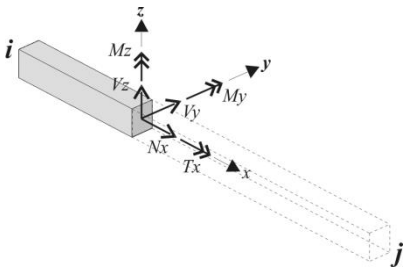
sscstCritical = 0x00	*simple sum considering +/- sign of seismic component as it is from the static analysis*
sscstFxyzWithLinkedMyz = 0x01**}|**	*it returns multiple results considering both +/- signs for Fx, Fy, Fz seismic components in summation. My and Mz will be taken into account with the same sign az Fy, Fz (hence the linked term)*

Summation rule of seismic internal force components for support forces.

Records / structures

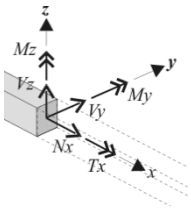
RLineForceValues = (

ELineType	**lfvLineType**	*line type*
double	**lfvNx**	*Nx normal force [kN]*
double	**lfvVy**	*Vy shear force [kN] (only for beams and ribs)*
double	**lfvVz**	*Vz shear force [kN] (only for beams and ribs)*
double	**lfvTx**	*Tx twisting moment [kNm] (only for beams and ribs)*
double	**lfvMy**	*My bending moment [kNm] (only for beams and ribs)*
double	**lfvMz**	*Mz bending moment [kNm] (only for beams and ribs)*
double	**lfvMyD**	*MyD design moment including the effect of eccentricity [kNm] (only for ribs)*
)		



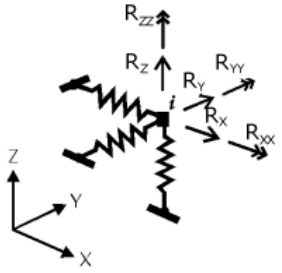
RVirtualBeamForceValues = (

double	**vbfvNx**	*Nx normal force [kN]*
double	**vbfvVy**	*Vy shear force [kN]*
double	**vbfvVz**	*Vz shear force [kN]*
double	**vbfvTx**	*Tx twisting moment [kNm]*
double	**vbfvMy**	*My bending moment [kNm]*
double	**vbfvMz**	*Mz bending moment [kNm]*
)		



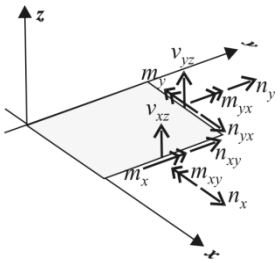
RNodalSupportForceValues = (

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**	*nx cross-section force [kN/m]*
double	**sfvNy**	*ny cross-section force [kN/m]*
double	**sfvNxy**	*nxy cross-section torsional force [kN/m]*
double	**sfvMx**	*mx bending moment [kNm/m]*
double	**sfvMy**	*my bending moment [kNm/m]*
double	**sfvMxy**	*mxy torsional moment [kNm/m]*
double	**sfvVxz**	*vxz shear force [kN/m]*
double	**sfvVyz**	*vyz shear force [kN/m]*
double	**sfvVSz**	*vSz resultant shear force [kN/m]*
double	**sfvN1**	*n1 1st principal force [kN/m]*
double	**sfvN2**	*n2 2nd principal force [kN/m]*
double	**sfvAn**	*an principal force direction [°]*
double	**sfvM1**	*m1 1st principal moment [kNm/m]*
double	**sfvM2**	*m2 2nd principal moment [kNm/m]*
double	**sfvAm**	*am principal moment direction [°]*
double	**sfvNxD**	*nxD design force [kN/m]*
double	**sfvNyD**	*nyD design force [kN/m]*
double	**sfvMxDp**	*mxD design moment (plus) [kNm/m]*
double	**sfvMxDm**	*mxD design moment (minus) [kNm/m]*
double	**sfvMyDp**	*myD design moment (plus) [kNm/m]*
double	**sfvMyDm**	*myD design moment (minus) [kNm/m]*
	)	



RSurfaceForces = (

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*
RSurfaceForceValues	**sfvCenterPoint**	*forces at the surface centerpoint*
RSurfaceForceValues	**sfvContourPoint1**	*forces at the 1st contour node*
RSurfaceForceValues	**sfvContourPoint2**	*forces at the 2nd contour node*
RSurfaceForceValues	**sfvContourPoint3**	*forces at the 3rd contour node*
RSurfaceForceValues	**sfvContourPoint4**	*forces at the 4th contour node*
RSurfaceForceValues	**sfvContourLineMidPoint1**	*forces at the 1st contour line midpoint*
RSurfaceForceValues	**sfvContourLineMidPoint2**	*forces at the 2nd contour line midpoint*
RSurfaceForceValues	**sfvContourLineMidPoint3**	*forces at the 3rd contour line midpoint*
RSurfaceForceValues	**sfvContourLineMidPoint4**	*forces at the 4th contour line midpoint*
	)	

RLineSupportForceValues = (

double	**Rx**	*support force in local x direction [kN/m]*
double	**Ry**	*support force in local y direction [kN/m]*
double	**Rz**	*support force in local z direction [kN/m]*
double	**Rxx**	*support moment about the local x axis [kNm/m]*
double	**Ryy**	*support moment about the local y axis [kNm/m]*
double	**Rzz**	*support moment about the local z axis [kNm/m]*
double	**Rr**	*resultant force [kN/m]*
double	**Rrr**	*resultant moment [kNm/m]*
	)	

RSurfaceSupportForceValues = (

double	**Rx**	*support force in local x direction [kN/m²]*
double	**Ry**	*support force in local y direction [kN/m²]*
double	**Rz**	*support force in local z direction [kN/m²]*

)

RSurfaceSupportForces = (

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*
RSurfaceSupportForceValues	**ssfvCenterPoint**	*support forces at the surface centerpoint*
RSurfaceSupportForceValues	**ssfvContourPoint1**	*support forces at the 1st contour node*
RSurfaceSupportForceValues	**ssfvContourPoint2**	*support forces at the 2nd contour node*
RSurfaceSupportForceValues	**ssfvContourPoint3**	*support forces at the 3rd contour node*
RSurfaceSupportForceValues	**ssfvContourPoint4**	*support forces at the 4th contour node*
RSurfaceSupportForceValues	**ssfvContourLineMidPoint1**	*support forces at the 1st contour line midpoint*
RSurfaceSupportForceValues	**ssfvContourLineMidPoint2**	*support forces at the 2nd contour line midpoint*
RSurfaceSupportForceValues	**ssfvContourLineMidPoint3**	*support forces at the 3rd contour line midpoint*
RSurfaceSupportForceValues	**ssfvContourLineMidPoint4**	*support forces at the 4th contour line midpoint*

)

RSpringForceValues = (

double	**Rx**	*force in local x direction [kN]*
double	**Ry**	*force in local y direction [kN]*
double	**Rz**	*force in local z direction [kN]*
double	**Rxx**	*moment about the local x axis [kNm]*
double	**Ryy**	*moment about the local y axis [kNm]*
double	**Rzz**	*moment about the local z axis [kNm]*

)

REdgeConnectionForceValues = (

double	**ecfvNx**	*force in local x direction [kN]*
double	**ecfvVy**	*force in local y direction [kN]*
double	**ecfvVz**	*force in local z direction [kN]*
double	**ecfvTx**	*moment about the local x axis [kNm]*
double	**ecfvMy**	*moment about the local y axis [kNm]*
double	**ecfvMz**	*moment about the local z axis [kNm]*

)

REdgeConnectionForces = (

REdgeConnectionForceValues	**ecfSection1**	*edge connection forces at the beginning of edge*
REdgeConnectionForceValues	**ecfSection2**	*edge connection forces at middle of edge*
REdgeConnectionForceValues	**ecfSection3**	*edge connection forces at the end of edge*

)

RLinkElementForceValues = (

double	**lefvNx**	*force in local x direction [kN]*
double	**lefvVy**	*force in local y direction [kN]*
double	**lefvVz**	*force in local z direction [kN]*
double	**lefvTx**	*moment about the local x axis [kNm]*
double	**lefvMy**	*moment about the local y axis [kNm]*
double	**lefvMz**	*moment about the local z axis [kNm]*

)

RLinkElementForces = (

ELinkElementType	**lefLinkElementType**	*type of the link element*
RLinkElementForceValues	**lefSection1**	*link element forces at the beginning of link*
RLinkElementForceValues	**lefSection2**	*link element forces at middle of link*
RLinkElementForceValues	**lefSection3**	*link element forces at the end of link*

)

Gap Forces

Single EleMENT reader functions

long **GapForceByLoadCaseId** ([in] long **LineId,** [i/o] double **Force,** [out] BSTR **Combination**)

LineId	*line index (0 < LineId ≤ AxisVMModel.Lines.Count)*
Force	*gap force*
Combination	*name of the load case*

Retrieves gap force according to the LoadCaseId (and LoadLevelOrTimeStep) property. Returns LineId or an error code (errDatabaseNotReady or see EForcesError).

---

long **GapForceByLoadCombinationId** ([in] long **LineId,** [i/o] double **Force,**
[out] BSTR **Combination**)

LineId	*line index (0 < LineId ≤ AxisVMModel.Lines.Count)*
Force	*gap force*
Combination	*name of the load case*

Retrieves gap force according to the LoadCombinationId (and LoadLevelOrTimeStep) property. Returns LineId or an error code (errDatabaseNotReady or see EForcesError).

---

long **EnvelopeGapForce** ([in] long **LineId,** [i/o] double **Force,** [out] BSTR **Combination**)

LineId	*line index (0 < LineId ≤ 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 LineId or an error code (errDatabaseNotReady or see EForcesError).

---

long **EnvelopeGapForce2** ([in] long **LineId,** [i/o] double **Force,** [out] long **LoadCaseOrCombinationId,**
[out] long **LoadLevel**)

LineId	*line index (0 < LineId ≤ AxisVMModel.Lines.Count)*
Force	*gap force*
LoadCaseOrCombinationId	*load case or load combination index, if index is > 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 LineId or an error code (errDatabaseNotReady or see EForcesError).

---

long **CriticalGapForce** ([in] long **LineId,** [i/o] double **Force,** [out] BSTR **Combination**)

LineId	*line index (0 < LineId ≤ 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 LineId or an error code (errDatabaseNotReady or see EForcesError).

---

long **CriticalGapForce2** ([in] long **LineId,** [i/o] double **Force,** [out] ECombinationType **CriticalCombinationType,** [out] SAFEARRAY(double) **Factors,** [out] SAFEARRAY(long) **LoadCaseIds**)

CriticalCombinationType	*combination type corresponding to critical load combination*
Factors	*load factors of the critical load combination*
LoadCaseIds	*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 **LineId,**
[i/o] SAFEARRAY(RResultBlockInfo) **ResultBlockInfo,** [i/o] SAFEARRAY(double) **Forces**)

LineId *line index (0 < LineId ≤ 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 **LineId** 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 **LineForceByLoadCaseId** ([in] long **LineId,** [in] long **SectionId,**
[i/o] RLineForceValues **Force,** [out] double **PosX,** [out] BSTR **Combination**)

LineId	*line index (0 < LineId ≤ AxisVMModel.Lines.Count)*
SectionId	*section index (0 < SectionId ≤ AxisVMModel.Lines[LineId].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 LoadCaseId (and LoadLevelOrTimeStep) property. Returns LineId or an error code (errDatabaseNotReady or see EForcesError).

---

long **LineForceByLoadCombinationId** ([in] long **LineId,** [in] long **SectionId,**
[i/o] RLineForceValues **Force,** [out] double **PosX,** [out] BSTR **Combination**)

LineId	*line index (0 < LineId ≤ AxisVMModel.Lines.Count)*
SectionId	*section index (0 < SectionId ≤ AxisVMModel.Lines[LineId].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 LineId or an error code (errDatabaseNotReady or see EForcesError).

---

long **EnvelopeLineForce** ([in] long **LineId,** [in] long **SectionId,**
[i/o] RLineForceValues **Force,** [out] double **PosX,** [out] BSTR **Combination**)

LineId	*line index (0 < LineId ≤ AxisVMModel.Lines.Count)*
SectionId	*section index (0 < SectionId ≤ AxisVMModel.Lines[LineId].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 LineId or an error code (errDatabaseNotReady or see EForcesError).

---

long **EnvelopeLineForce2** ([in] long **LineId,** [in] long **SectionId,**
[i/o] RLineForceValues **Force,** [out] double **PosX,** [out] long **LoadCaseOrCombinationId,**
[out] long **LoadLevel**)

LineId	*line index (0 < LineId ≤ AxisVMModel.Lines.Count)*
SectionId	*section index (0 < SectionId ≤ AxisVMModel.Lines[LineId].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 LineId or an error code (errDatabaseNotReady or see EForcesError).

---

long **CriticalLineForce** ([in] long **LineId,** [in] long **SectionId,**
[i/o] RLineForceValues **Force,** [out] double **PosX,** [out] BSTR **Combination**)

LineId	*line index (0 < LineId ≤ AxisVMModel.Lines.Count)*
SectionId	*section index (0 < SectionId ≤ AxisVMModel.Lines[LineId].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. Returns LineId or an error code (errDatabaseNotReady or see EForcesError).

---

long **CriticalLineForce2** ([in] long **LineId,** [in] long **SectionId,** [i/o] RLineForceValues **Force,** [out] double **PosX,** [out] ECombinationType **CriticalCombinationType,** [out] SAFEARRAY(double) **Factors,**
[out] SAFEARRAY(long) **LoadCaseIds**)

CriticalCombinationType	*combination type corresponding to critical load combination*
Factors	*load factors of the critical load combination*
LoadCaseIds	*load case indexes of the critical load combination*

Similar to CriticalLineForce with more parameters described above.

---

long **LineForceByLoadCombinationIdEQ** ([in] long **LineId,** [in] long **SectionId,**
[i/o] SAFEARRAY(RLineForceValues) **Force,** [out] double **PosX,** [out] BSTR **Combination**)

LineId	*line index (0 < LineId ≤ AxisVMModel.Lines.Count)*
SectionId	*section index (0 < SectionId ≤ AxisVMModel.Lines[LineId].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. 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).

---

long **EnvelopeLineForceEQ** ([in] long **LineId,** [in] long **SectionId,**
[i/o] SAFEARRAY(RLineForceValues) **Force,** [out] double **PosX,** [out] BSTR **Combination**)

LineId	*line index (0 < LineId ≤ AxisVMModel.Lines.Count)*
SectionId	*section index (0 < SectionId ≤ AxisVMModel.Lines[LineId].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. 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).

---

long **CriticalLineForceEQ** ([in] long **LineId,** [in] long **SectionId,**
[i/o] SAFEARRAY(RLineForceValues) **Force,** [out] double **PosX,** [out] BSTR **Combination**)

LineId	*line index (0 < LineId ≤ AxisVMModel.Lines.Count)*
SectionId	*section index (0 < SectionId ≤ AxisVMModel.Lines[LineId].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 **LineForcesByLoadCaseId** ([in] long **LineId,**
[i/o] SAFEARRAY(RLineForceValues) **Forces,** [out] SAFEARRAY(double) **PosX**)

LineId	*line index (0 < LineId ≤ AxisVMModel.Lines.Count)*
Forces	*array of line forces for the line element.* *Length of the array is AxisVMModel.Lines[LineId].SectionCount*
PosX	*array of cross-section positions in m according to the local x direction* *Length of the array is AxisVMModel.Lines[LineId].SectionCount*

Retrieves forces 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 EForcesError).

---

long **LineForcesByLoadCombinationId** ([in] long **LineId,**
[i/o] SAFEARRAY(RLineForceValues) **Forces,** [out] SAFEARRAY(double) **PosX**)

LineId	*line index (0 < LineId ≤ AxisVMModel.Lines.Count)*
Forces	*array of line forces for the line element.* *Length of the array is AxisVMModel.Lines[LineId].SectionCount*
PosX	*array of cross-section positions in m according to the local x direction* *Length of the array is AxisVMModel.Lines[LineId].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 **LineId,**
[i/o] SAFEARRAY(RLineForceValues) **Forces,** [out] SAFEARRAY(double) **PosX**)

LineId	*line index (0 < LineId ≤ AxisVMModel.Lines.Count)*
Forces	*array of envelope line forces for the line element.* *Length of the array is AxisVMModel.Lines[LineId].SectionCount*
PosX	*array of cross-section positions in m according to the local x direction* *Length of the array is AxisVMModel.Lines[LineId].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 **LineId,**
[i/o] SAFEARRAY(RLineForceValues) **Forces,** [out] SAFEARRAY(double) **PosX**)

LineId	*line index (0 < LineId ≤ AxisVMModel.Lines.Count)*
Forces	*array of critical line forces for the line element.* *Length of the array is AxisVMModel.Lines[LineId].SectionCount*
PosX	*array of cross-section positions in m according to the local x direction* *Length of the array is AxisVMModel.Lines[LineId].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 **LineForcesByLoadCaseIdConnectedToNode** ([in] long **NodeId,** [in] long **WindowId,**
[in] EConnectedToNodeType **ConnectedToNodeType,**
[out] SAFEARRAY(RLineForceValues) **Forces,** [out] SAFEARRAY(long) **ConnectedLineIds,**
[out] SAFEARRAY(BSTR) **Combinations**)

NodeId	*node index the lines connected to (0 < NodeId ≤ AxisVMModel.Nodes.Count)*
WindowId	*active window index (0 < WindowId ≤ AxisVMModel.Windows.Count)* *(Note: used only if EConnectedToNodeType = ctntVisible)*
ConnectedToNodeType	*selection type*
Forces	*array of line force results*
ConnectedLineIds	*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 NodeId index according to the LoadCaseId (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 **NodeId,**
[in] long **WindowId,** [in] EConnectedToNodeType **ConnectedToNodeType,**
[out] SAFEARRAY(RLineForceValues) **Forces,**
[out] SAFEARRAY(long) **ConnectedLineIds,** [out] SAFEARRAY(BSTR) **Combinations**)

NodeId	*node index the lines connected to (0 < NodeId ≤ AxisVMModel.Nodes.Count)*
WindowId	*active window index (0 < WindowId ≤ AxisVMModel.Windows.Count)* *(Note: used only if EConnectedToNodeType = ctntVisible)*
ConnectedToNodeType	*selection type*
Forces	*array of line force results*
ConnectedLineIds	*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 NodeId 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 **NodeId,** [in] long **WindowId,**
[in] EConnectedToNodeType **ConnectedToNodeType,** [in] long **LineId,**
[out] SAFEARRAY(RLineForceValues) **Forces,**
[out] SAFEARRAY(long) **ConnectedLineIds,** [out] SAFEARRAY(BSTR) **Combinations**)

NodeId	*node index the lines connected to (0 < NodeId ≤ AxisVMModel.Nodes.Count)*
WindowId	*active window index (0 < WindowId ≤ AxisVMModel.Windows.Count) (Note: used only if EConnectedToNodeType = ctntVisible)*
ConnectedToNodeType	*selection type*
Component	*reference component (Nx, Vy, etc.)*
LineId	*reference line connected to the node*
Forces	*array of envelope line force results*
ConnectedLineIds	*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 NodeId 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. Nx max) related to line with LineId 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 **NodeId,** [in] long **WindowId,**
[in] EConnectedToNodeType **ConnectedToNodeType,** [in] long **LineId,**
[out] SAFEARRAY(RLineForceValues) **Forces,**
[out] SAFEARRAY(long) **ConnectedLineIds,** [out] SAFEARRAY(BSTR) **Combinations**)

NodeId	*node index the lines connected to (0 < NodeId ≤ AxisVMModel.Nodes.Count)*
WindowId	*active window index (0 < WindowId ≤ AxisVMModel.Windows.Count) (Note: used only if EConnectedToNodeType = ctntVisible)*
ConnectedToNodeType	*selection type*
LineId	*reference line connected to the node*
Forces	*array of critical line force results*
ConnectedLineIds	*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 NodeId 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. Nx max) related to line with LineId 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 = AxisVMModel.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 = AxisVMModel.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 = AxisVMModel.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.*
Length of the array = AxisVMModel.Lines.Count

Forces *array of critical 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 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.
Returns the total number of cross-sections or an error code (errDatabaseNotReady or see EForcesError).

Multiple BLOCK reader functions

long **LineForcesForResultBlocks** ([in] long **LineId,** [in] long **SectionId,**
[i/o] SAFEARRAY(RResultBlockInfo) **ResultBlockInfo,**
[i/o] SAFEARRAY(RLineForceValues) **Forces,** [out] SAFEARRAY(double) **PosX**)

LineId *line index (0 < LineId ≤ AxisVMModel.Lines.Count)*

SectionId *section index (0 < SectionId ≤ AxisVMModel.Lines[LineId].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*

Retrieves forces 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, Forces and PosX arrays or an error code (errDatabaseNotReady or see EForcesError).

Single ELEMENT reader functions

long **VirtualBeamOrStripForcesByLoadCaseId** ([in] long **Index,** [in] long **ChainId,**
[i/o] SAFEARRAY(RVirtualBeamForceValues)* **Forces,** [i/o] SAFEARRAY(RPoint3d)* **Pos**)

Index *virtual beam/strip index (0 < Index ≤ AxisVMModel.VirtualBeams.Count)*
ChainId *virtual beam's/strip's chain index*
(0 < ChainId ≤ AxisVMModel.VirtualBeams.ChainCount[Index])
Forces *array of virtual beam/strip forces*
Length of the array is AxisVMModel.VirtualBeams.SectionCount[Index, ChainId]
Pos *array of section positions in m according to the global coordinate system*
Length of the array is AxisVMModel.VirtualBeams.SectionCount[Index, ChainId]

Retrieves forces for each sections of a given chain of a virtual beam/strip according to the LoadCaseId (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 ≤ AxisVMModel.VirtualBeams.Count)*
ChainId *virtual beam's/strip's chain index*
(0 < ChainId ≤ AxisVMModel.VirtualBeams.ChainCount[Index])
Forces *array of virtual beam/strip forces*
Length of the array is AxisVMModel.VirtualBeams.SectionCount[Index, ChainId]
Pos *array of section positions in m according to the global coordinate system*
Length of the array is 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 ≤ AxisVMModel.VirtualBeams.Count)*
ChainId *virtual beam's/strip's chain index*
(0 < ChainId ≤ AxisVMModel.VirtualBeams.ChainCount[Index])
Forces *array of virtual beam/strip forces*
Length of the array is AxisVMModel.VirtualBeams.SectionCount[Index, ChainId]
Pos *array of section positions in m according to the global coordinate system*
Length of the array is 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 < Index ≤ AxisVMModel.VirtualBeams.Count)*
ChainId	*virtual beam's/strip's chain index (0 < ChainId ≤ AxisVMModel.VirtualBeams.ChainCount[Index])*
Forces	*array of virtual beam/strip forces Length of the array is AxisVMModel.VirtualBeams.SectionCount[Index, ChainId]*
Pos	*array of section positions in m according to the global coordinate system 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*

Retrieves envelope forces for each sections of a given chain of a virtual beam/strip. Envelope is identified by VirtualBeamForceComponent, 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 EForcesError).

---

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 < Index ≤ AxisVMModel.VirtualBeams.Count)*
ChainId	*virtual beam's/strip's chain index (0 < ChainId ≤ AxisVMModel.VirtualBeams.ChainCount[Index])*
SectionId	*chain's section index (0 < ChainId ≤ 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*

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). Returns the section index or an error code (errDatabaseNotReady or see EForcesError).

---

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)* **LoadCaseIds**)

CriticalCombinationType	*combnation type corresponding to critical load combination*
Factors	*load factors of the critical load combination*
LoadCaseIds	*load case indexes of the critical load combination*

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).

---

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 < Index ≤ AxisVMModel.VirtualBeams.Count)*
ChainId	*virtual beam's/strip's chain index (0 < ChainId ≤ AxisVMModel.VirtualBeams.ChainCount[Index])*
Forces	*array of virtual beam/strip forces Length of the array is AxisVMModel.VirtualBeams.SectionCount[Index, ChainId]*
Pos	*array of section positions in m according to the global coordinate system Length of the array is AxisVMModel.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	*name of the file with extension emf*
ID	*index of the virtual beam or strip*
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 virtual beam or strip*
ColourMode	*colour mode*

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	*name of the file with extension emf*
ID	*index of the virtual beam or strip*
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 virtual beam or strip*
ColourMode	*colour mode*

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	*name of the file with extension emf*
ID	*index of the virtual beam or strip*
LoadCaseId	*load case 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 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 ≤ LoadLevelCount)* *for timestep (0 < LoadLevelOrTimeStep ≤ 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 < LineSupportId ≤ AxisVMModel.LineSupports.Count)*
LineSectionId	*section index (0 < SectionId ≤ AxisVMModel.LineSupports.SectionCount[LineSupportId,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 < LineSupportId ≤ AxisVMModel.LineSupports.Count)*
LineSectionId	*section index (0 < SectionId ≤ AxisVMModel.LineSupports.SectionCount[LineSupportId,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 **LineSupportForceByLoadCombinationIdEQ** ([in] long **LineSupportId,** [in] long **LineSectionId,** [i/o] SAFEARRAY(RLineSupportForceValues) **Forces,** [out] double **PosX,** [out] BSTR **Combination**)

LineSupportId	*line support index (0 < LineSupportId ≤ AxisVMModel.LineSupports.Count)*
LineSectionId	*section index (0 < SectionId ≤ AxisVMModel.LineSupports.SectionCount[LineSupportId,AnalysisType])*
Forces	*array of force results*
PosX	*position of SectionId in m according to the local x direction*
Combination	*name of the load case*

Retrieves the forces of a line support for a force component according to the LoadCombinationId. The generation of Forces is governed by the SupportSeismicSumType property. The full list of the hidden parameters : AnalysisType, LoadCombinationId , LineSupportForceComponent, SupportSeismicSumType Returns the length of Forces 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 < LineSupportId ≤ AxisVMModel.LineSupports.Count)*
LineSectionId	*section index (0 < SectionId ≤ AxisVMModel.LineSupports.SectionCount[LineSupportId,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 < LineSupportId ≤ AxisVMModel.LineSupports.Count)*

LineSectionId *section index (0 < SectionId ≤ AxisVMModel.LineSupports.SectionCount[LineSupportId,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 **EnvelopeLineSupportForceEQ** ([in] long **LineSupportId,** [in] long **LineSectionId,** [i/o] SAFEARRAY(RLineSupportForceValues) **Forces,** [out] double **PosX,** [out] BSTR **Combination**)

LineSupportId *line support index (0 < LineSupportId ≤ AxisVMModel.LineSupports.Count)*
LineSectionId *section index (0 < SectionId ≤ AxisVMModel.LineSupports.SectionCount[LineSupportId,AnalysisType])*
Forces *array of 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 the forces of a line support for a force component from an envelope. Envelope is identified by MinMaxType and EnvelopeUID properties. The generation of Forces is governed by the SupportSeismicSumType property. The full list of the hidden parameters : AnalysisType, MinMaxType, EnvelopeUID, LineSupportForceComponent, SupportSeismicSumType. Returns the length of Forces 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 *line support index (0 < LineSupportId ≤ AxisVMModel.LineSupports.Count)*
LineSectionId *section index (0 < SectionId ≤ AxisVMModel.LineSupports.SectionCount[LineSupportId,AnalysisType])*
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 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) **LoadCaseIds**)

CriticalCombinationType *combination typec orresponding to critical load combination*
Factors *load factors of the critical load combination*
LoadCaseIds *load case indexes of the critical load combination*

Similar to CriticalLineSupportForce with more parameters described above.

---

long **CriticalLineSupportForceEQ** ([in] long **LineSupportId,** [in] long **LineSectionId,** [i/o] SAFEARRAY(RLineSupportForceValues) **Forces,** [out] double **PosX,** [out] BSTR **Combination**)

LineSupportId *line support index (0 < LineSupportId ≤ AxisVMModel.LineSupports.Count)*
LineSectionId *section index (0 < SectionId ≤ AxisVMModel.LineSupports.SectionCount[LineSupportId,AnalysisType])*
Forces *array of 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 the forces of a line support for a force component for a given critical combination type. The critical combination is determined by MinMaxType and CombinationType properties. The generation of Forces is governed by the SupportSeismicSumType property. The full list of the hidden parameters : AnalysisType, MinMaxType, CombinationType, LineSupportForceComponent, SupportSeismicSumType. Returns the length of Forces or an error code (errDatabaseNotReady or see EForcesError).

Single ELEMENT reader functions

long **LineSupportForcesByLoadCaseId** ([in] long **LineSupportId,**
[i/o] SAFEARRAY(RLineSupportForceValues) **Forces,** [out] SAFEARRAY(double) **PosX**)

LineSupportId *line support index (0 < LineSupportId ≤ AxisVMModel.LineSupports.Count)*
Forces *array of line support forces*
Length of the array is AxisVMModel.LineSupports.SectionCount[LineSupportId,AnalysisType])
PosX *array of cross-section positions in m according to the local x direction - Length of the array is AxisVMModel.LineSupports.SectionCount[LineSupportId,AnalysisType])*

Retrieves line support forces 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 EForcesError).

---

long **LineSupportForcesByLoadCombinationId** ([in] long **LineSupportId,**
[i/o] SAFEARRAY(RLineSupportForceValues) **Forces,** [out] SAFEARRAY(double) **PosX**)

LineSupportId *line support index (0 < LineSupportId ≤ AxisVMModel.LineSupports.Count)*
Forces *array of line support forces*
Length of the array is AxisVMModel.LineSupports.SectionCount[LineSupportId,AnalysisType])
PosX *array of cross-section positions in m according to the local x direction*
Length of the array is SectionCount[LineId]

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 ≤ AxisVMModel.LineSupports.Count)*
Forces *array of line support forces*
Length of the array is AxisVMModel.LineSupports.SectionCount[LineSupportId,AnalysisType])
PosX *array of cross-section positions in m according to the local x direction*
Length of the array is SectionCount[LineId]

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.
Returns the total number of cross-sections or an error code (errDatabaseNotReady or see EForcesError).

---

long **CriticalLineSupportForces** ([in] long **LineSupportId,**
[i/o] SAFEARRAY(RLineSupportForceValues) **Forces,** [out] SAFEARRAY(double) **PosX**)

LineSupportId *line support index (0 < LineSupportId ≤ AxisVMModel.LineSupports.Count)*
Forces *array of line support forces*
Length of the array is AxisVMModel.LineSupports.SectionCount[LineSupportId,AnalysisType])
PosX *array of cross-section positions in m according to the local x direction*
Length of the array is SectionCount[LineId]

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.
Returns the total number of cross-sections or an error code (errDatabaseNotReady or see EForcesError).

Multiple element reader functions

long **AllLineSupportForcesByLoadCaseId** ([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 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 forces of all line supports 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 **AllLineSupportForcesByLoadCombinationId** ([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 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 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).

---

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 < LineSupportId ≤ AxisVMLineSupports.Count)*

LineSectionId *section index (0 < SectionId ≤ AxisVMLineSupports.SectionCount[LineSupportId,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 < MembersSupportId ≤ 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 < MembersSupportId ≤ 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 < MembersSupportId ≤ 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	*member support index (0 < MembersSupportId ≤ AxisVMModel.MembersSupports.Count)*
MinMaxType	*only min or max allowed*
CombinationType	*load combination type allowed by national design code*
AnalysisType	*analysis type*
Component	*support force component*
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, 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	*nodal support index (0 < NodalSupportId ≤ AxisVMNodalSupports.Count)*
Force	*force results*
Combination	*name of the load case*

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	*nodal support index (0 < NodalSupportId ≤ AxisVMNodalSupports.Count)*
Force	*force results*
Combination	*name of the load combination*

Retrieves forces of a nodal support according to the LoadCombinationId (and LoadLevelOrTimeStep) property. Returns NodalSupportId or an error code (errDatabaseNotReady or see EForcesError).

---

long **NodalSupportForceByLoadCombinationIdEQ** ([in] long **NodalSupportId,**
[i/o] SAFEARRAY(RNodalSupportForceValues) **Forces,** [out] BSTR **Combination**)

NodalSupportId	*nodal support index (0 < NodalSupportId ≤ AxisVMNodalSupports.Count)*
Forces	*array of nodal support forces*
Combination	*name of the load combination*

Retrieves forces of a nodal support from the load combination in the LoadCombinationId property. The generation of Forces is governed by the SupportSeismicSumType property. The full list of the hidden parameters : AnalysisType, LoadCombinationId , NodalSupportForceComponent, SupportSeismicSumType. Returns the length of Forces or an error code (errDatabaseNotReady or see EForcesError).

---

long **EnvelopeNodalSupportForce** ([in] long **NodalSupportId,**
[i/o] RNodalSupportForceValues **Force,** [out] BSTR **Combination**)

NodalSupportId	*nodal support index (0 < NodalSupportId ≤ AxisVMNodalSupports.Count)*
Force	*force results*
Combination	*load case or combination in which Component has its minimum or maximum*

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 **EnvelopeNodalSupportForceEQ**([in] long **NodalSupportId**,

[i/o] SAFEARRAY(*RNodalSupportForceValues*) **Forces**, [out] BSTR **Combination**)

NodalSupportId	*nodal support index (0 < NodalSupportId ≤ AxisVMNodalSupports.Count)*
Forces	*array of nodal support forces*
Combination	*load case or combination in which Component has its minimum or maximum*

Retrieves the forces of a nodal support for a force component from an envelope. Envelope is identified by MinMaxType and EnvelopeUID properties. The generation of Forces is governed by the SupportSeismicSumType property. The full list of the hidden parameters : AnalysisType, MinMaxType, EnvelopeUID, NodalSupportForceComponent, SupportSeismicSumType. Returns the length of Forces or an error code (errDatabaseNotReady or see EForcesError).

---

long **EnvelopeNodalSupportForce2** ([in] long **NodalSupportId,**
[i/o] RNodalSupportForceValues **Force,** [out] long **LoadCaseOrCombinationId,**
[out] long **LoadLevel**)

NodalSupportId	*nodal support index (0 < NodalSupportId ≤ AxisVMNodalSupports.Count)*
Force	*force results*
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 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 **CriticalNodalSupportForce** ([in] long **NodalSupportId,** [i/o] RNodalSupportForceValues **Force,**
[out] BSTR **Combination**)

NodalSupportId	*nodal support index (0 < NodalSupportId ≤ AxisVMNodalSupports.Count)*
Force	*force results*
Combination	*critical combination in which Component has its minimum or maximum*

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).

---

long **CriticalNodalSupportForce2** ([in] long **NodalSupportId,**
[i/o] RNodalSupportForceValues **Force,** [out] ECombinationType **CriticalCombinationType,**
[out] SAFEARRAY(double) **Factors,** [out] SAFEARRAY(long) **LoadCaseIds**)

CriticalCombinationType	*combination type corresponding to critical load combination*
Factors	*load factors of the critical load combination*
LoadCaseIds	*load case indexes of the critical load combination*

Similar to CriticalNodalSupportForce with more parameters described above.

---

long **CriticalNodalSupportForceEQ** ([in] long **NodalSupportId,** [i/o] SAFEARRAY(RNodalSupportForceValues) **Forces,**
[out] BSTR **Combination**)

NodalSupportId	*nodal support index* *(0 < NodalSupportId ≤ AxisVMNodalSupports.Count)*
Forces	*array of nodal support forces*
Combination	*critical combination in which Component has its minimum or maximum*

Retrieves the forces of a nodal support for a force component for a given critical combination type. The critical combination is determined by MinMaxType and CombinationType properties. The generation of Forces is governed by the SupportSeismicSumType property. The full list of the hidden parameters : AnalysisType, MinMaxType, CombinationType, NodalSupportForceComponent, SupportSeismicSumType. Returns the length of Forces or an error code (errDatabaseNotReady or see EForcesError).

Multiple element reader functions

long **AllNodalSupportForcesByLoadCaseId** (
[i/o] SAFEARRAY(RNodalSupportForceValues) **Forces**)

Forces	*array of nodal support forces.* *Length of the array is AxisVMNodalSupports.Count.*

Retrieves forces of all nodal supports 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 **AllNodalSupportForcesByLoadCombinationId** (
[i/o] SAFEARRAY(RNodalSupportForceValues) **Forces**)

Forces	*array of nodal support forces.* *Length of the array is AxisVMNodalSupports.Count.*

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).

---

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 < NodalSupportId ≤ 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 < LineId ≤ AxisVMModel.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 < LineId ≤ AxisVMModel.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**)

LineId	*line index (0 < LineId ≤ AxisVMModel.Lines.Count)*
Force	*force results*
Combination	*load case or combination in which Component has its minimum or maximum*

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).

---

long **EnvelopeSpringForce2** ([in] long **LineId,** [i/o] RSpringForceValues **Force,** [out] long **LoadCaseOrCombinationId,** [out] long **LoadLevel**)

LineId	*line index (0 < LineId ≤ AxisVMModel.Lines.Count)*
Force	*force results*
LoadCaseOrCombinationId	*load case or load combination index, if index is > IAxisVMLoadcases.count then Load combination index = LoadCaseOrCombinationId – IAxisVMLoadcases.count*
LoadLevel	*load level*

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).

---

long **CriticalSpringForce** ([in] long **LineId,** [i/o] RSpringForceValues **Force,** [out] BSTR **Combination**)

LineId	*line index (0 < LineId ≤ AxisVMModel.Lines.Count)*
Force	*force results*
Combination	*critical combination in which Component has its minimum or maximum*

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).

---

long **CriticalSpringForce2** ([in] long **LineId,** [i/o] RSpringForceValues **Force,**
[out] ECombinationType **CriticalCombinationType,** [out] SAFEARRAY(double) **Factors,**
[out] SAFEARRAY(long) **LoadCaseIds**)

CriticalCombinationType	*combination type corresponding to critical load combination*
Factors	*load factors of the critical load combination*
LoadCaseIds	*load case indexes of the critical load combination*

Similar to CriticalSpringForce with more parameters described above.

Multiple element reader functions

long **AllSpringForcesByLoadCaseId** ([i/o] SAFEARRAY(RSpringForceValues) **Forces**)

Forces	*array of spring forces.* *Length of the array is the number of spring elements in the model.*

Retrieves all spring 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 **AllSpringForcesByLoadCombinationId** (
[i/o] SAFEARRAY(RSpringForceValues) **Forces**)

Forces *array of spring forces.*
Length of the array is the number of spring elements in the model.

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).

---

long **AllEnvelopeSpringForces** ([i/o] SAFEARRAY(RSpringForceValues) **Forces**)

Forces *array of spring forces.*
Length of the array is the number of spring elements in the model.

Retrieves all envelope spring forces. Envelope is identified by SpringtForceComponent, 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 **AllCriticalSpringForces** ([i/o] SAFEARRAY(RSpringForceValues) **Forces**)

Forces *array of spring forces.*
Length of the array is the number of spring elements in the model.

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.
Returns the array length or an error code (errDatabaseNotReady or see EForcesError).

Multiple BLOCK reader functions

long **SpringForcesForResultBlocks** ([in] long **LineId,**[i/o] SAFEARRAY(RResultBlockInfo) **ResultBlockInfo,** [i/o] SAFEARRAY(RSpringForceValues) **Forces**)

LineId *line index (0 < LineId ≤ AxisVMModel.Lines.Count)*
ResultBlockInfo *array of description records for result blocks*
Forces *force results for all result blocks*

Retrieves spring 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).

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 < SurfaceVertexId ≤AxisVMNodes.Count)*
svtContourLinePoint	*line index (0 < SurfaceVertexId ≤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 **SurfaceId,** [in] ESurfaceVertexType **SurfaceVertexType,** [in] long **SurfaceVertexId,** [i/o] RSurfaceForceValues **Force,** [out] BSTR **Combination**)

SurfaceId *surface index (0 < SurfaceId ≤AxisVMSurfaces.Count)*
SurfaceVertexType *vertex type*
SurfaceVertexId *vertex identifier (node or line index)*
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 SurfaceId or an error code (errDatabaseNotReady or see EForcesError).

---

long **SurfaceForceByLoadCombinationId** ([in] long **SurfaceId,** [in] ESurfaceVertexType **SurfaceVertexType,** [in] long **SurfaceVertexId,** [i/o] RSurfaceForceValues **Force,** [out] BSTR **Combination**)

SurfaceId *surface index (0 < SurfaceId ≤AxisVMSurfaces.Count)*
SurfaceVertexType *vertex type*
SurfaceVertexId *vertex identifier (node or line index)*
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 SurfaceId or an error code (errDatabaseNotReady or see EForcesError).

---

long **SurfaceForceByLoadCombinationIdEQ** ([in] long **SurfaceId,** [in] ESurfaceVertexType **SurfaceVertexType,** [in] long **SurfaceVertexId,** [i/o] SAFEARRAY(RSurfaceForceValues) **Forces,** [out] BSTR **Combination**)

SurfaceId *surface index (0 < SurfaceId ≤AxisVMSurfaces.Count)*
SurfaceVertexType *vertex type*
SurfaceVertexId *vertex identifier (node or line index)*
Forces *array of force results*
Combination *name of the load combination*

Retrieves forces at a point of a surface according to the LoadCombinationId (and LoadLevelOrTimeStep) property. The full list of the hidden parameters : AnalysisType, LoadCombinationId, LoadLevelOrTimeStep (but since only linear analysis is valid, it is not used), SurfaceForceComponent. Returns the length of Forces or an error code (errDatabaseNotReady or see EForcesError).

---

long **EnvelopeSurfaceForce** ([in] long **SurfaceId,** [in] ESurfaceVertexType **SurfaceVertexType,** [in] long **SurfaceVertexId,** [i/o] RSurfaceForceValues **Force,** [out] BSTR **Combination**)

SurfaceId *surface index (0 < SurfaceId ≤AxisVMSurfaces.Count)*
SurfaceVertexType *vertex type*
SurfaceVertexId *vertex identifier (node or line index)*
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 SurfaceId or an error code (errDatabaseNotReady or see EForcesError).

---

long **EnvelopeSurfaceForceEQ** ([in] long **SurfaceId,** [in] ESurfaceVertexType **SurfaceVertexType,** [in] long **SurfaceVertexId,** [i/o] SAFEARRAY(RSurfaceForceValues) **Forces,** [out] BSTR **Combination**)

SurfaceId *surface index (0 < SurfaceId ≤ AxisVMSurfaces.Count)*
SurfaceVertexType *vertex type*
SurfaceVertexId *vertex identifier (node or line index)*
Forces *array of 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 MinMaxType and EnvelopeUID properties. Force contains the result of the load case or combination in which Component is maximal or minimal. . The full list of the hidden parameters : AnalysisType, MinMaxType, EnvelopeUID, SurfaceForceComponent. Returns the length of Forces or an error code (errDatabaseNotReady or see EForcesError).

---

long **CriticalSurfaceForce** ([in] long **SurfaceId,** [in] ESurfaceVertexType **SurfaceVertexType,** [in] long **SurfaceVertexId,** [i/o] RSurfaceForceValues **Force,** [out] BSTR **Combination**)

SurfaceId *surface index (0 < SurfaceId ≤ AxisVMSurfaces.Count)*
SurfaceVertexType *vertex type*
SurfaceVertexId *vertex identifier (node or line index)*
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 SurfaceId or an error code (errDatabaseNotReady or see EForcesError).

---

long **CriticalSurfaceForce2** ([in] long **SurfaceId,** [in] ESurfaceVertexType **SurfaceVertexType,** [in] long **SurfaceVertexId,** [i/o] RSurfaceForceValues **Force,** [out] ECombinationType **CriticalCombinationType,** [out] SAFEARRAY(double) **Factors,** [out] SAFEARRAY(long) **LoadCaseIds**)

SurfaceId *surface index (0 < SurfaceId ≤ AxisVMSurfaces.Count)*
SurfaceVertexType *vertex type*
SurfaceVertexId *vertex identifier (node or line index)*
Force *force results*
CriticalCombinationType *combination type corresponding to critical load combination*
Factors *load factors of the critical load combination*
LoadCaseIds *load case indexes of the critical load combination*

Same as CriticalSurfaceForce, but retrieving parseable combination data.

---

long **CriticalSurfaceForceEQ** ([in] long **SurfaceId,** [in] ESurfaceVertexType **SurfaceVertexType,** [in] long **SurfaceVertexId,** [i/o] SAFEARRAY(RSurfaceForceValues) **Forces,** [out] BSTR **Combination**)

SurfaceId *surface index (0 < SurfaceId ≤ AxisVMSurfaces.Count)*
SurfaceVertexType *vertex type*
SurfaceVertexId *vertex identifier (node or line index)*
Forces *array of force results*

Combination *critical combination in which Component has its minimum or maximum*

Retrieves critical forces at one point of a surface element. The returned critical combination depends on SurfaceForceComponent and MinMaxType properties (e.g. Nxy max). Force contains the result of the critical combination in which Component is maximal or minimal. The full list of the hidden parameters : AnalysisType, MinMaxType, CombinationType, SurfaceForceComponent. Returns the length of Forces or an error code (errDatabaseNotReady or see EForcesError).

Single ELEMENT reader functions

long **SurfaceForcesByLoadCaseId** ([in] long **SurfaceId,** [i/o] RSurfaceForces **Forces**)

SurfaceId *surface index (0 < SurfaceId ≤AxisVMModel.Surfaces.Count)*
Forces *surface forces on the element*

Retrieves forces in all available points of a surface element according to the LoadCaseId (and LoadLevelOrTimeStep) property. Returns SurfaceId or an error code (errDatabaseNotReady or see EForcesError).

---

long **SurfaceForcesByLoadCombinationId** ([in] long **SurfaceId,** [i/o] RSurfaceForces **Forces**)

SurfaceId *surface index (0 < SurfaceId ≤AxisVMModel.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 SurfaceId or an error code (errDatabaseNotReady or see EForcesError).

---

long **EnvelopeSurfaceForces** ([in] long **SurfaceId,** [i/o] RSurfaceForces **Forces**)

SurfaceId *surface index (0 < SurfaceId ≤AxisVMModel.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 SurfaceId or an error code (errDatabaseNotReady or see EForcesError).

---

long **CriticalSurfaceForces** ([in] long **SurfaceId,** [i/o] RSurfaceForces **Forces**)

SurfaceId *surface index (0 < SurfaceId ≤AxisVMModel.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 SurfaceId 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 AxisVMModel.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 AxisVMModel.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 AxisVMModel.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 AxisVMModel.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 **SurfaceId,** [i/o] SAFEARRAY(RResultBlockInfo) **ResultBlockInfo,**
[i/o] SAFEARRAY(RSurfaceForces) **Forces**)

SurfaceId *surface index (0 < SurfaceId ≤ AxisVMModel.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 **SurfaceId,** [in] ESurfaceVertexType **SurfaceVertexType,** [in] long **SurfaceVertexId,** [i/o] SAFEARRAY(RResultBlockInfo) **ResultBlockInfo,**
[i/o] SAFEARRAY(RSurfaceForceValues) **Forces**)

SurfaceId	*surface index (0 < SurfaceId ≤ 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 **SurfaceSupportForceByLoadCaseId** ([in] long **SurfaceSupportId,**
[in] ESurfaceVertexType **SurfaceVertexType,** [in] long **SurfaceVertexId,**
[i/o] RSurfaceSupportForceValues **Force,** [out] BSTR **Combination**)

SurfaceSupportId	*surface support index (0 < SurfaceSupportId ≤ 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 < SurfaceSupportId ≤ 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 < SurfaceSupportId ≤ 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 < SurfaceSupportId ≤ AxisVMSurfaceSupports.Count)*
SurfaceVertexType	*vertex type*
SurfaceVertexId	*vertex identifier*
Force	*force results*
LoadCaseOrCombinationId	*load case or load combination index, if index is > IAxisVMLoadcases.count then Load combination index = LoadCaseOrCombinationId – IAxisVMLoadcases.count*
LoadLevel	*load level*

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 **CriticalSurfaceSupportForce** ([in] long **SurfaceSupportId,**
[in] ESurfaceVertexType **SurfaceVertexType,** [in] long **SurfaceVertexId,**
[i/o] RSurfaceSupportForceValues **Force,** [out] BSTR **Combination**)

SurfaceSupportId	*surface support index (0 < SurfaceSupportId ≤ AxisVMSurfaceSupports.Count)*
SurfaceVertexType	*vertex type*
SurfaceVertexId	*vertex identifier*
Force	*force results*
Combination	*critical combination in which Component has its minimum or maximum*

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).

---

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) **LoadCaseIds**)

CriticalCombinationType	*combination type corresponding to critical load combination*
Factors	*load factors of the critical load combination*
LoadCaseIds	*load case indexes of the critical load combination*

Similar to CriticalSurfaceSupportForce with more parameters described above.

Single ELEMENT reader functions

long **SurfaceSupportForcesByLoadCaseId** ([in] long **SurfaceSupportId,**
[i/o] RSurfaceSupportForces **Forces**)

SurfaceSupportId	*surface support index (0 < SurfaceSupportId ≤ AxisVMSurfaceSupports.Count)*
Forces	*surface support forces on the element*

Retrieves surface support forces in all available points of a surface element according to the LoadCaseId (and LoadLevelOrTimeStep) property. Returns SurfaceSupportId or an error code (errDatabaseNotReady or see EForcesError).

---

long **SurfaceSupportForcesByLoadCombinationId** ([in] long **SurfaceSupportId,**
[i/o] RSurfaceSupportForces **Forces**)

SurfaceSupportId *surface support index (0 < SurfaceSupportId ≤ AxisVMSurfaceSupports.Count)*
Forces *surface support forces on the element*

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).

---

long **EnvelopeSupportSurfaceForces** ([in] long **SurfaceSupportId,** [i/o] RSurfaceSupportForces **Forces**)

SurfaceSupportId *surface support index (0 < SurfaceSupportId ≤ AxisVMSurfaceSupports.Count)*
Forces *surface support forces on the element*

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).

---

long **CriticalSurfaceSupportForces** ([in] long **SurfaceSupportId,** [i/o] RSurfaceSupportForces **Forces**)

SurfaceSupportId *surface support index (0 < SurfaceSupportId ≤ AxisVMSurfaceSupports.Count)*
Forces *surface support forces on the element*

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).

Multiple element reader functions

long **AllSurfaceSupportForcesByLoadCaseId** ([i/o] SAFEARRAY(RSurfaceSupportForces) **Forces**)

Forces *array of surface support forces.*
Length of the array is AxisVMModel.SurfaceSupports.Count

Retrieves surface support 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 **AllSurfaceSupportForcesByLoadCombinationId** (
[i/o] SAFEARRAY(RSurfaceSupportForces) **Forces**)

Forces *array of surface support forces.*
Length of the array is AxisVMModel.SurfaceSupports.Count

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).

---

long **AllEnvelopeSurfaceSupportForces** ([i/o] SAFEARRAY(RSurfaceSupportForces) **Forces**)

Forces *array of surface support forces.*
Length of the array is AxisVMModel.SurfaceSupports.Count

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.
Returns the total number of records in the array or an error code (errDatabaseNotReady or see EForcesError).

---

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	*edge connection index (0 < EdgeConnectionId ≤ AxisVMEdgeConnections.Count)*
LoadCombinationId	*load combination index (0 < LoadCombinationId ≤ AxisVMLoadCombinations.Count)*
LoadLevel	*load level (increment) index*
AnalysisType	*type of analysis*
Forces	*forces in edge connection*
Combination	*name of the combination*

If successful returns edge connection index, otherwise returns an error code (errDatabaseNotReady, feInvalidCombinationOfLoadCombinationAndLoadLevel, feInvalidAnalysisType, feEdgeConnectionIndexOutOfBounds).

---

long **GetEnvelopeEdgeConnectionForces** ([in] long **EdgeConnectionId,** [in] long **SectionId**, [in] EMinMaxType **MinMaxType**, [in] EAnalysisType **AnalysisType**, [in] EEdgeConnectionForce **Component,** [i/o] REdgeConnectionForces* **Forces,** [out] BSTR* **Combination**)

EdgeConnectionId	*edge connection index (0 < EdgeConnectionId ≤ AxisVMEdgeConnections.Count)*
SectionId	*section index (1,2 or 3) along edge*
MinMaxType	*minimum or maximum value*
AnalysisType	*type of analysis*
Component	*edge connection force component*
Forces	*forces of edge connection*
Combination	*name of the combination*

Envelope is identified by EdgeConnectionForceComponent, MinMaxType and EnvelopeUID properties. If successful returns edge connection index, otherwise returns an error code (errDatabaseNotReady, feInvalidAnalysisType, feEdgeConnectionIndexOutOfBounds, feSectionIndexOutOfBounds).

---

long **GetEnvelopeEdgeConnectionForces2** ([in] long **EdgeConnectionId,** [in] long **SectionId**, [in] EMinMaxType **MinMaxType**, [in] EAnalysisType **AnalysisType**, [in] EEdgeConnectionForce **Component,** [i/o] REdgeConnectionForces* **Forces,** [out] long **LoadCaseOrCombinationId,** [out] long **LoadLevel**)

EdgeConnectionId	*edge connection index (0 < EdgeConnectionId ≤ AxisVMEdgeConnections.Count)*
SectionId	*section index (1,2 or 3) along edge*
MinMaxType	*minimum or maximum value*
AnalysisType	*type of analysis*
Component	*edge connection force component*
Forces	*forces of edge connection*
LoadCaseOrCombinationId	*load case or load combination index, if index is > IAxisVMLoadcases.count then Load combination index = LoadCaseOrCombinationId – IAxisVMLoadcases.count*
LoadLevel	*load level*

Envelope is identified by EdgeConnectionForceComponent, MinMaxType and EnvelopeUID properties. If successful returns edge connection index, otherwise returns an error code (errDatabaseNotReady, feInvalidAnalysisType, feEdgeConnectionIndexOutOfBounds, feSectionIndexOutOfBounds).

---

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 ≤ AxisVMEdgeConnections.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] ECombinationType **CriticalCombinationType,** [out] SAFEARRAY(double) **Factors,** [out] SAFEARRAY(long) **LoadCaseIds**)

EdgeConnectionId	*edge connection index (0 < EdgeConnectionId ≤ AxisVMEdgeConnections.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*
LoadCaseIds	*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 **EdgeConnectionForcesByLoadCaseId** ([in] long **EdgeConnectionId,** [i/o] REdgeConnectionForces* **Forces,** [out] BSTR***** **Combination**)

EdgeConnectionId	*edge connection index (0 < EdgeConnectionId ≤ AxisVMEdgeConnections.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**)

EdgeConnectionId	*edge connection index (0 < EdgeConnectionId ≤ AxisVMEdgeConnections.Count)*
Forces	*forces*
Combination	*name of the combination*

Envelope is identified by EdgeConnectionForceComponent, MinMaxType and EnvelopeUID properties. If successful returns edge connection index, otherwise returns an error code (errDatabaseNotReady, feInvalidCombinationOfLoadCombinationAndLoadLevel, feInvalidAnalysisType, feEdgeConnectionIndexOutOfBounds).

---

long **EnvelopeEdgeConnectionForces** ([in] long **EdgeConnectionId,** [in] long **SectionId,**
[i/o] REdgeConnectionForces* **Forces,** [out] BSTR* **Combination**)

EdgeConnectionId	*edge connection index (0 < EdgeConnectionId ≤ AxisVMEdgeConnections.Count)*
SectionId	*section index (1,2 or 3) along edge*
Forces	*forces of edge connection*
Combination	*name of the combination*

If successful returns edge connection index, otherwise returns an error code (errDatabaseNotReady, feInvalidAnalysisType, feEdgeConnectionIndexOutOfBounds, feSectionIndexOutOfBounds).

---

long **EnvelopeEdgeConnectionForces2** ([in] long **EdgeConnectionId,** [in] long **SectionId,**
[i/o] REdgeConnectionForces* **Forces,** [out] long **LoadCaseOrCombinationId,**
[out] long **LoadLevel**)

EdgeConnectionId	*edge connection index (0 < EdgeConnectionId ≤ AxisVMEdgeConnections.Count)*
SectionId	*section index (1,2 or 3) along edge*
Forces	*forces of edge connection*
LoadCaseOrCombinationId	*load case or load combination index, if index is > IAxisVMLoadcases.count then Load combination index = LoadCaseOrCombinationId – IAxisVMLoadcases.count*
LoadLevel	*load level*

Envelope is identified by EdgeConnectionForceComponent, MinMaxType and EnvelopeUID properties. If successful returns edge connection index, otherwise returns an error code (errDatabaseNotReady, feInvalidAnalysisType, feEdgeConnectionIndexOutOfBounds, feSectionIndexOutOfBounds).

---

long **CriticalEdgeConnectionForces** ([in] long **EdgeConnectionId,** [in] long **SectionId,**
[i/o] REdgeConnectionForces* **Forces,** [out] BSTR* **Combination**)

EdgeConnectionId	*edge connection index (0 < EdgeConnectionId ≤ AxisVMEdgeConnections.Count)*
SectionId	*section index (1 for NN or 1,2 or 3 fol LL)*
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 **CriticalEdgeConnectionForces2** ([in] long **EdgeConnectionId,** [in] long **SectionId,**
[i/o] REdgeConnectionForces* **Forces,** [out] ECombinationType **CriticalCombinationType,**
[out] SAFEARRAY(double) **Factors,** [out] SAFEARRAY(long) **LoadCaseIds**)

CriticalCombinationType	*combination type corresponding to critical load combination*
Factors	*load factors of the critical load combination*
LoadCaseIds	*load case indexes of the critical load combination*

Similar to CriticalEdgeConnectionForces with more parameters described above.

Multiple element reader functions

long **GetAllEdgeConnectionForcesByLoadCaseId** ([in] long **LoadCaseId,** [in] long **LoadLevel,** [in] EAnalysisType **AnalysisType,** [out] SAFEARRAY(REdgeConnectionForces)* **Forces**)

LoadCaseId	*load case index (0 < LoadCaseId ≤ AxisVMLoadCases.Count)*
LoadLevel	*load level or time step (increment) index*
AnalysisType	*type of Analysis*
Forces	*array with forces of all edge connections*

If successful returns number of edge connections, otherwise returns an error code (errDatabaseNotReady, feInvalidCombinationOfLoadCaseAndLoadLevel, feInvalidAnalysisType, feNoEdgeConnectionsInTheModel).

---

long **GetAllEdgeConnectionForcesByLoadCombinationId** ([in] long **LoadCombinationId,** [in] long **LoadLevel,** [in] EAnalysisType **AnalysisType,** [out] SAFEARRAY(REdgeConnectionForces)* **Forces**)

LoadCombinationId	*load combination index (0 < LoadCombinationId ≤ AxisVMLoadCombinations.Count)*
LoadLevel	*load level or time step (increment) index*
AnalysisType	*type of Analysis*
Forces	*array with forces of all edge connections*

If successful returns number of edge connections, otherwise returns an error code (errDatabaseNotReady, feInvalidCombinationOfLoadCombinationAndLoadLevel, feInvalidAnalysisType, feNoEdgeConnectionsInTheModel).

---

long **GetAllEnvelopeEdgeConnectionForces** ([in] EMinMaxType **MinMaxType**, [in] EAnalysisType **AnalysisType**, [in] EEdgeConnectionForce **Component,** [out] SAFEARRAY(REdgeConnectionForces)* **Forces**)

MinMaxType	*minimum or maximum value*
AnalysisType	*type of analysis*
Component	*edge connection force component*
Forces	*array with forces of all edge connections (each edge connection returns three REdgeConnectionForces records)*

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).

---

long **GetAllCriticalEdgeConnectionForces** ([in] EMinMaxType **MinMaxType**, [in] ECombinationType **CombinationType**, [in] EAnalysisType **AnalysisType**, [in] EEdgeConnectionForce **Component,** [out] SAFEARRAY(REdgeConnectionForces)* **Forces**)

MinMaxType	*minimum or maximum value*
CombinationType	*combination type*
AnalysisType	*type of analysis*
Component	*edge connection force component*
Forces	*array with forces of all edge connections (each edge connection returns three REdgeConnectionForces records)*

If successful returns 3 x number of edge connections, otherwise returns an error code (errDatabaseNotReady, feInvalidAnalysisType, feNoEdgeConnectionsInTheModel, feCombinationTypeNotValidForCurrentNationalDesignCode).

---

long **AllEdgeConnectionForcesByLoadCaseId** ([out] SAFEARRAY(REdgeConnectionForces)* **Forces**)

Forces	*array with forces of all edge connections*

If successful returns number of edge connections, otherwise returns an error code (errDatabaseNotReady, feInvalidCombinationOfLoadCaseAndLoadLevel, feInvalidAnalysisType, feNoEdgeConnectionsInTheModel).

---

long **AllEdgeConnectionForcesByLoadCombinationId** ([out] SAFEARRAY(REdgeConnectionForces)* **Forces**)

Forces *array with forces of all edge connections*

If successful returns number of edge connections, otherwise returns an error code (errDatabaseNotReady, feInvalidCombinationOfLoadCombinationAndLoadLevel, feInvalidAnalysisType, feNoEdgeConnectionsInTheModel).

---

long **AllEnvelopeEdgeConnectionForces** ([out] SAFEARRAY(REdgeConnectionForces)* **Forces**)

Forces *array with forces of all edge connections*

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).

---

long **AllCriticalEdgeConnectionForces** ([out] SAFEARRAY(REdgeConnectionForces)* **Forces**)

Forces *array with forces of all edge connections*

If successful returns 3 x number of edge connections, otherwise returns an error code (errDatabaseNotReady, feInvalidAnalysisType, feNoEdgeConnectionsInTheModel, feCombinationTypeNotValidForCurrentNationalDesignCode).

Multiple BLOCK reader functions

long **GetEdgeConnectionForcesForResultBlocks** ([in] long **EdgeConnectionId,** [in] EAnalysisType **AnalysisType,** [out] SAFEARRAY(RResultBlockInfo)* **ResultBlockInfo,** [out] SAFEARRAY(REdgeConnectionForces)* **Forces**)

EdgeConnectionId *edge connection index*
(0 < EdgeConnectionId ≤ AxisVMEdgeConnections.Count)
AnalysisType *type of analysis*
ResultBlockInfo *array of result block info*
Forces *array with forces of all edge connections*

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).

---

long **EdgeConnectionForcesForResultBlocks** ([in] long **EdgeConnectionId,** [out] SAFEARRAY(RResultBlockInfo)* **ResultBlockInfo,** [out] SAFEARRAY(REdgeConnectionForces)* **Forces**)

EdgeConnectionId *edge connection index*
(0 < EdgeConnectionId ≤ AxisVMEdgeConnections.Count)
ResultBlockInfo *array of result block info*
Forces *array with forces of all edge connections*

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).

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**)

LinkElementId	*link element index (0 < LinkElementId ≤ AxisVMLinkElements.Count)*
LoadCaseId	*load case index (0 < LoadCaseId ≤ AxisVMLoadCases.Count)*
LoadLevel	*load level or time step (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, feLinkElementIndexOutOfBounds).

---

long **GetLinkElementForcesByLoadCombinationId** ([in] long **LinkElementId,** [in] long **LoadCombinationId,** [in] long **LoadLevel,** [in] EAnalysisType **AnalysisType,** [i/o] RLinkElementForces* **Forces,** [out] BSTR*** Combination**)

LinkElementId	*link element index (0 < LinkElementId ≤ AxisVMLinkElements.Count)*
LoadCombinationId	*load combination index (0 < LoadCombinationId ≤ AxisVMLoadCombinations.Count)*
LoadLevel	*load level or time step (increment) index*
AnalysisType	*type of analysis*
Forces	*forces in edge connection*
Combination	*name of the combination*

If successful returns edge connection index, otherwise returns an error code (errDatabaseNotReady, feInvalidCombinationOfLoadCombinationAndLoadLevel, feInvalidAnalysisType, feLinkElementIndexOutOfBounds).

---

long **GetEnvelopeLinkElementForces** ([in] long **LinkElementId,** [in] long **SectionId**, [in] EMinMaxType **MinMaxType**, [in] EAnalysisType **AnalysisType**, [in] ELinkElementForce **Component,** [i/o] RLinkElementForces* **Forces,** [out] BSTR* **Combination**)

LinkElementId	*link element index (0 < LinkElementId ≤ AxisVMLinkElements.Count)*
SectionId	*section index (1 for NN or 1,2 or 3 fol LL)*
MinMaxType	*minimum or maximum value*
AnalysisType	*type of analysis*
Component	*link element force component*
Forces	*forces of edge connection*
Combination	*name of the combination*

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)).

---

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**)

LinkElementId	*link element index (0 < LinkElementId ≤ AxisVMLinkElements.Count)*
SectionId	*section index (1 for NN or 1,2 or 3 fol LL)*
MinMaxType	*minimum or maximum value*
CombinationType	*combination type*
AnalysisType	*type of analysis*
Component	*link element 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, feLinkElementIndexOutOfBounds, feSectionIndexOutOfBounds (for LL elements), feCombinationTypeNotValidForCurrentNationalDesignCode.).

---

long **LinkElementForcesByLoadCaseId** ([in] long **LinkElementId,** [i/o] RLinkElementForces* **Forces,** [out] BSTR*** Combination**)

LinkElementId	*link element index (0 < LinkElementId ≤ AxisVMLinkElements.Count)*
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, feLinkElementIndexOutOfBounds).

---

long **LinkElementForcesByLoadCombinationId** ([in] long **LinkElementId,** [i/o] RLinkElementForces* **Forces,** [out] BSTR*** Combination**)

LinkElementId	*link element index (0 < LinkElementId ≤ AxisVMLinkElements.Count)*
Forces	*forces in edge connection*
Combination	*name of the combination*

If successful returns edge connection index, otherwise returns an error code (errDatabaseNotReady, feInvalidCombinationOfLoadCombinationAndLoadLevel, feInvalidAnalysisType, feLinkElementIndexOutOfBounds).

---

long **EnvelopeLinkElementForces** ([in] long **LinkElementId,** [in] long **SectionId,** [i/o] RLinkElementForces* **Forces,** [out] BSTR* **Combination**)

LinkElementId	*link element index (0 < LinkElementId ≤ AxisVMLinkElements.Count)*
SectionId	*section index (1 for NN or 1,2 or 3 fol LL)*
Forces	*forces of edge connection*
Combination	*name of the combination*

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)).

---

long **EnvelopeLinkElementForces2** ([in] long **LinkElementId,** [in] long **SectionId,**
[i/o] RLinkElementForces* **Forces,** [out] long **LoadCaseOrCombinationId,** [out] long **LoadLevel**)

LinkElementId	*link element index (0 < LinkElementId ≤ AxisVMLinkElements.Count)*
SectionId	*section index (1 for NN or 1,2 or 3 fol LL)*
Forces	*forces of edge connection*
LoadCaseOrCombinationId	*load case or load combination index, if index is > IAxisVMLoadcases.count then Load combination index = LoadCaseOrCombinationId – IAxisVMLoadcases.count*
LoadLevel	*load level*

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)).

---

long **CriticalLinkElementForces** ([in] long **LinkElementId,** [in] long **SectionId,**
[i/o] RLinkElementForces* **Forces,** [out] BSTR* **Combination**)

LinkElementId	*link element index (0 < LinkElementId ≤ AxisVMLinkElements.Count)*
SectionId	*section index (1 for NN or 1,2 or 3 fol LL)*
Forces	*forces of edge connection*
Combination	*name of critical combination*

If successful returns edge connection index, otherwise returns an error code (errDatabaseNotReady, feInvalidAnalysisType, feLinkElementIndexOutOfBounds, feSectionIndexOutOfBounds (for LL elements), feCombinationTypeNotValidForCurrentNationalDesignCode.).

---

long **CriticalLinkElementForces2** ([in] long **LinkElementId,** [in] long **SectionId,**
[i/o] RLinkElementForces* **Forces,** [out] ECombinationType **CriticalCombinationType,**
[out] SAFEARRAY(double) **Factors,** [out] SAFEARRAY(long) **LoadCaseIds**)

CriticalCombinationType	*combination type corresponding to critical load combination*
Factors	*load factors of the critical load combination*
LoadCaseIds	*load case indexes of the critical load combination*

Similar to CriticalLinkElementForces with more parameters described above.

Multiple element reader functions

long **GetAllLinkElementForcesByLoadCaseId** ([in] long **LoadCaseId,** [in] long **LoadLevel,** [in] EAnalysisType **AnalysisType,** [out] SAFEARRAY(RLinkElementForces)* **Forces**)

LoadCaseId	*load case index (0 < LoadCaseId ≤ AxisVMLoadCases.Count)*
LoadLevel	*load level or time step (increment) index*
AnalysisType	*type of analysis*
Forces	*Array with forces from all link elements*

If successful returns number of link elements, otherwise returns an error code (errDatabaseNotReady, feInvalidCombinationOfLoadCaseAndLoadLevel, feInvalidAnalysisType).

long **GetAllLinkElementForcesByLoadCombinationId** ([in] long **LoadCombinationId,** [in] long **LoadLevel,** [in] EAnalysisType **AnalysisType,** [out] SAFEARRAY(RLinkElementForces)* **Forces**)

LoadCombinationId *load combination index (0 < LoadCombinationId ≤ AxisVMLoadCombinations.Count)*
LoadLevel *load level or time step (increment) index*
AnalysisType *type of analysis*
Forces *Array with forces from all link elements*

If successful returns number of link elements, otherwise returns an error code (errDatabaseNotReady, feInvalidCombinationOfLoadCombinationAndLoadLevel, feInvalidAnalysisType).

---

long **GetAllEnvelopeLinkElementForces** ([in] EMinMaxType **MinMaxType**, [in] EAnalysisType **AnalysisType**, [in] ELinkElementForce **Component,** [out] SAFEARRAY(RLinkElementForces)* **Forces**)

MinMaxType *minimum or maximum value*
AnalysisType *type of analysis*
Component *link element force component*
Forces *Array with forces from all link elements*

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, feInvalidAnalysisType, feSectionIndexOutOfBounds(for LL elements))

---

long **GetAllCriticalLinkElementForces** ([in] EMinMaxType **MinMaxType**, [in] ECombinationType **CombinationType**, [in] EAnalysisType **AnalysisType**, [in] ELinkElementForce **Component,** [out] SAFEARRAY(RLinkElementForces)* **Forces**)

MinMaxType *minimum or maximum value*
CombinationType *combination type*
AnalysisType *type of analysis*
Component *link element force component*
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).

---

long **AllLinkElementForcesByLoadCaseId** ([out] SAFEARRAY(RLinkElementForces)* **Forces**)

Forces *Array with forces from all link elements*

If successful returns number of link elements, otherwise returns an error code (errDatabaseNotReady, feInvalidCombinationOfLoadCaseAndLoadLevel, feInvalidAnalysisType).

---

long **AllLinkElementForcesByLoadCombinationId** ([out] SAFEARRAY(RLinkElementForces)* **Forces**)

Forces *Array with forces from all link elements*

If successful returns number of link elements, otherwise returns an error code (errDatabaseNotReady, feInvalidCombinationOfLoadCombinationAndLoadLevel, feInvalidAnalysisType).

---

long **AllEnvelopeLinkElementForces** ([out] SAFEARRAY(RLinkElementForces)* **Forces**)

Forces *Array with forces from all link elements*

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, feInvalidAnalysisType, feSectionIndexOutOfBounds(for LL elements))

---

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 < LinkElementId ≤ 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 < LinkElementId ≤ 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 **GetMemberForcesByLoadCaseId** ([in] long **MemberID,** [in] long **LoadCaseId,** [in] long **LoadLevelOrTimeStep,** [in] EAnalysisType **AnalysisType,** [out] SAFEARRAY(RLineForceValues) **Forces,** [out] SAFEARRAY(double) **PosX**)

MemberID *member index (0 < MemberId ≤ AxisVMModel.Members.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 forces for each cross-section of a given element according to the LoadCaseId and LoadLevelOrTimeStep. Returns the total number of cross-sections or an error code (errDatabaseNotReady or see EForcesError).

long **GetMemberForcesByLoadCombinationId** ([in] long **MemberID,** [in] long **LoadCombinationId,**
[in] long **LoadLevelOrTimeStep,** [in] EAnalysisType **AnalysisType,**
[out] SAFEARRAY(RLineForceValues) **Forces,** [out] SAFEARRAY(double) **PosX**)

MemberID	*member index (0 < MemberId ≤ AxisVMModel.Members.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 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 LoadCombinationId and LoadLevelOrTimeStep. Returns the total number of cross-sections or an error code (errDatabaseNotReady or see EForcesError).

---

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**)

MemberID	*member index (0 < MemberId ≤ AxisVMModel.Members.Count)*
EnvelopeUID	*unique envelope index*
MinMaxType	*minimum or maximum value*
AnalysisType	*analysis type*
Component	*force component*
Forces	*array of member forces for the member*
PosX	*array of cross-section positions in m according to the local x direction*
LoadCaseOrCombinationIds	*array of load case or load combination indexes, if index is > IAxisVMLoadcases.count then Load combination index = LoadCaseOrCombinationId – IAxisVMLoadcases.count*
LoadLevels	*array of load levels*

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).

---

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**)

MemberID	*member index (0 < MemberId ≤ AxisVMModel.Members.Count)*
MinMaxType	*Minimum or maximum value*
CombinationType	*combination type*
AnalysisType	*analysis type*
Component	*force component*
Forces	*array of member forces for the member*
PosX	*array of cross-section positions in m according to the local x direction*

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).

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] ELongBoolean **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 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] ELongBoolean **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 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	*name of the file with extension emf*
MemberId	*member index*
LoadCaseId	*load case 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 member*
ColourMode	*colour mode*

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	*name of the file with extension emf*
MemberId	*member index*
LoadCombinationId	*load combination index*
LoadLevelOrTimeStep	*for load level (0 < LoadLevelOrTimeStep ≤ LoadLevelCount)* *for timestep (0 < LoadLevelOrTimeStep ≤ 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 member*
ColourMode	*colour mode*

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] ELongBoolean **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 < LoadCaseId ≤AxisVMModel.LoadCases.Count)*
long	**LoadCombinationId** • *Get or set load combination index (0 < LoadCombinationId ≤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 for line forces*
ESupportSeismicComponentSumType	**SupportSeismicSumType**• *Get or set the type of summation of seismic internal forces for support 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*
ESurfaceSupportForce	**SurfaceSupportForceComponent** • *Get or set the surface support force component for envelope or critical results*
ELongBoolean	**UserCreep** *Returns lbTrue if nonlinear analysis results consider concrete creep. More here…*
EVirtualBeamForce	**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	*COM server error*
rceLoadCaseIdIndexOutOfBounds = -100002	*Invalid LoadCaseID while reading results*
rceLoadCombinationIdIndexOutOfBounds = -100003	*Invalid CombinationID while reading results*
rceInvalidAnalysisType = -100004	*Invalid type of results for used analysis*
rceCombinationTypeNotValidForCurrentNationalDesignCode = -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*
rceInvalidCombinationOfLoadCaseAndLoadLevel = -100006	*Invalid combination LoadCaseID and load level*
rceInvalidCombinationOfLoadCombinationAndLoadLevel = -100007}	*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	**rmxumin_ULS**	*Minimum ultimate(residual) moment for ULS in x direction [kNm/m]*
double	**rmxumax_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	**rmxumin_SLS**	*Minimum ultimate(residual) moment for SLS in x direction [kNm/m]*
double	**rmxumax_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 **GetReinforcementChecksByLoadCaseId** ([in] long **SurfaceId,** [in] long **LoadCaseId,** [in] long **LoadLevel**, [in] EAnalysisType **AnalysisType,**
[i/o] RReinforcementCheck* **ReinforcementCheck,** [out] BSTR* **Combination**)

SurfaceId	*index of the surface element*
LoadCaseId	*load case index (0 < LoadCaseId ≤ AxisVMLoadCases.Count)*
LoadLevel	*load level (increment) index*
AnalysisType	*Type of Analysis*
ReinforcementCheck	*Reinforcement checks of the surface element*
Combination	*String with name of load case.*

If successful returns SurfaceId, otherwise an error code (errDatabaseNotReady, errNotSupportedByNationalDesignCode, errIndexOutOfBounds, rceLoadCaseIdIndexOutOfBounds, rceInvalidCombinationOfLoadCaseAndLoadLevel, rceInvalidAnalysisType).

---

long **GetReinforcementChecksByLoadCombinationId** ([in] long **SurfaceId,**
[in] long **LoadCombinationId,** [in] long **LoadLevel**, [in] EAnalysisType **AnalysisType,**
[i/o] RReinforcementCheck* **ReinforcementCheck,** [out] BSTR* **Combination**)

SurfaceId	*index of the surface element*
LoadCombinationId	*load combination index (0 < LoadCombinationId ≤ AxisVMLoadCombinations.Count)*
LoadLevel	*load level (increment) index*
AnalysisType	*Type of Analysis*
ReinforcementCheck	*Reinforcement checks of the surface element*
Combination	*String with name of combination*

If successful returns SurfaceId, otherwise an error code (errDatabaseNotReady, errNotSupportedByNationalDesignCode, errIndexOutOfBounds, rceLoadCombinationIdIndexOutOfBounds, rceInvalidCombinationOfLoadCombinationAndLoadLevel, rceInvalidAnalysisType).

---

long **GetEnvelopeReinforcementChecks** ([in] long **SurfaceId,**
[in] EMinMaxType **MinMaxType**, [in] EAnalysisType **AnalysisType,**
[i/o] RReinforcementCheck* **ReinforcementCheck**)

SurfaceId	*index of the surface element*
MinMaxType	*Minimum or maximum value*
AnalysisType	*Type of Analysis*
ReinforcementCheck	*Reinforcement checks of the surface element*

Envelope is identified by MinMaxType and EnvelopeUID properties. If successful returns SurfaceId, otherwise an error code (errDatabaseNotReady, errNotSupportedByNationalDesignCode, errIndexOutOfBounds, rceInvalidAnalysisType).

---

long **GetCriticalReinforcementChecks** ([in] long **SurfaceId,** [in] EMinMaxType **MinMaxType**, [in] ECombinationType **CombinationType,** [in] EAnalysisType **AnalysisType,**
[i/o] RReinforcementCheck* **ReinforcementCheck**)

SurfaceId	*index of the surface element*
MinMaxType	*Minimum or maximum value*
CombinationType	*Combination Type*
AnalysisType	*Type of Analysis*
ReinforcementCheck	*Reinforcement checks of the surface element*

If successful returns SurfaceId, otherwise an error code (errDatabaseNotReady, errNotSupportedByNationalDesignCode, errIndexOutOfBounds, rceInvalidAnalysisType).

---

long **GetAllReinforcementChecksByLoadCaseId** ([in] long **LoadCaseId,**
[in] long **LoadLevel**, [in] EAnalysisType **AnalysisType,** [out]
SAFEARRAY(RReinforcementCheck)* **ReinforcementChecks,**
[out] SAFEARRAY (BSTR)* **Combinations**)

LoadCaseId	*load case index (0 < LoadCaseId ≤ AxisVMLoadCases.Count)*
LoadLevel	*load level (increment) index*
AnalysisType	*Type of Analysis*
ReinforcementChecks	*Array of reinforcement checks of each surface element*
Combinations	*Array of string with name of load cases.*

If successful returns number of surface elements, otherwise an error code (errDatabaseNotReady, errNotSupportedByNationalDesignCode, rceLoadCaseIdIndexOutOfBounds, rceInvalidCombinationOfLoadCaseAndLoadLevel, rceInvalidAnalysisType).

---

long **GetAllReinforcementChecksByLoadCombinationId** ([in] long **LoadCombinationId,**
[in] long **LoadLevel**, [in] EAnalysisType **AnalysisType,**
[out] SAFEARRAY(RReinforcementCheck)* **ReinforcementChecks,**
[out] SAFEARRAY (BSTR)* **Combinations**)

LoadCombinationId	*load combination index (0 < LoadCombinationId ≤ AxisVMLoadCombinations.Count)*
LoadLevel	*load level (increment) index*
AnalysisType	*Type of Analysis*
ReinforcementChecks	*Array of reinforcement checks of each surface element*
Combinations	*Array of strings with name of combinations.*

If successful returns number of surface elements, otherwise an error code (errDatabaseNotReady, errNotSupportedByNationalDesignCode, rceLoadCombinationIdIndexOutOfBounds, rceInvalidCombinationOfLoadCombinationAndLoadLevel, rceInvalidAnalysisType).

---

long **GetAllEnvelopeReinforcementChecks** ([in] EMinMaxType **MinMaxType**,
[in] EAnalysisType **AnalysisType,**
[out] SAFEARRAY(RReinforcementCheck)* **ReinforcementChecks**)

MinMaxType	*Minimum or maximum value*
AnalysisType	*Type of Analysis*
ReinforcementChecks	*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, rceInvalidAnalysisType).

---

long **GetAllCriticalReinforcementChecks** ([in] EMinMaxType **MinMaxType**,
[in] ECombinationType **CombinationType,** [in] EAnalysisType **AnalysisType,**
[out] SAFEARRAY(RReinforcementCheck)* **ReinforcementChecks**)

MinMaxType	*Minimum or maximum value*
CombinationType	*Combination Type*
AnalysisType	*Type of Analysis*
ReinforcementChecks	*Array of reinforcement checks of each surface element*

If successful returns number of surface elements, otherwise an error code (errDatabaseNotReady, errNotSupportedByNationalDesignCode, rceInvalidAnalysisType).

---

long **ReinforcementChecksByLoadCaseId** ([in] long **SurfaceId,**
[i/o] RReinforcementCheck* **ReinforcementCheck,**
[out] BSTR* **Combination**)

SurfaceId	*index of the surface element*
ReinforcementCheck	*Reinforcement checks of the surface element*
Combination	*String with name of load case.*

If successful returns SurfaceId, otherwise an error code (errDatabaseNotReady, errNotSupportedByNationalDesignCode, errIndexOutOfBounds, rceLoadCaseIdIndexOutOfBounds, rceInvalidCombinationOfLoadCaseAndLoadLevel, rceInvalidAnalysisType).

---

long **ReinforcementChecksByLoadCombinationId** ([in] long **SurfaceId,**
[i/o] RReinforcementCheck* **ReinforcementCheck,** [out] BSTR* **Combination**)

SurfaceId	*index of the surface element*
ReinforcementCheck	*Reinforcement checks of the surface element*
Combination	*String with name of load combination.*

If successful returns SurfaceId, otherwise an error code (errDatabaseNotReady, errNotSupportedByNationalDesignCode, errIndexOutOfBounds, rceLoadCombinationIdIndexOutOfBounds, rceInvalidCombinationOfLoadCombinationAndLoadLevel, rceInvalidAnalysisType).

---

long **EnvelopeReinforcementChecks** ([in] long **SurfaceId,**
[i/o] RReinforcementCheck* **ReinforcementCheck**)

SurfaceId	*index of the surface element*
ReinforcementCheck	*Reinforcement checks of the surface element*

Envelope is identified by MinMaxType and EnvelopeUID properties. If successful returns SurfaceId, otherwise an error code (errDatabaseNotReady, errNotSupportedByNationalDesignCode, errIndexOutOfBounds, rceInvalidAnalysisType).

---

long **CriticalReinforcementChecks** ([in] long **SurfaceId,**
[i/o] RReinforcementCheck* **ReinforcementCheck**)

SurfaceId	*index of the surface element*
ReinforcementCheck	*Reinforcement checks of the surface element*

If successful returns SurfaceId, otherwise an error code (errDatabaseNotReady, errNotSupportedByNationalDesignCode, errIndexOutOfBounds, rceInvalidAnalysisType).

---

long **AllReinforcementChecksByLoadCaseId** ([out] SAFEARRAY(RReinforcementCheck)* **ReinforcementChecks,** [out] SAFEARRAY (BSTR)* **Combinations**)

ReinforcementChecks	*Array of reinforcement checks of each surface element*
Combinations	*Array of string with name of load cases.*

If successful returns number of surface elements, otherwise an error code (errDatabaseNotReady, errNotSupportedByNationalDesignCode, rceLoadCaseIdIndexOutOfBounds, rceInvalidCombinationOfLoadCaseAndLoadLevel, rceInvalidAnalysisType).

---

long **AllReinforcementChecksByLoadCombinationId** ([out] SAFEARRAY(RReinforcementCheck)* **ReinforcementChecks,**
[out] SAFEARRAY (BSTR)* **Combinations**)

ReinforcementChecks	*Array of reinforcement checks of each surface element*
Combinations	*Array of string with name of combinations.*

If successful returns number of surface elements, otherwise an error code (errDatabaseNotReady, errNotSupportedByNationalDesignCode, rceLoadCombinationIdIndexOutOfBounds, rceInvalidCombinationOfLoadCombinationAndLoadLevel, rceInvalidAnalysisType).

---

long **AllEnvelopeReinforcementChecks** ([out] SAFEARRAY(RReinforcementCheck)* **ReinforcementChecks**)

ReinforcementChecks *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, rceInvalidAnalysisType).*

long **AllCriticalReinforcementChecks** ([out] SAFEARRAY(RReinforcementCheck)* **ReinforcementChecks**)

ReinforcementChecks *Array of reinforcement checks of each surface element If successful returns number of surface elements, otherwise an error code (errDatabaseNotReady, errNotSupportedByNationalDesignCode, rceInvalidAnalysisType).*

Properties

EAnalysisType	**AnalysisType** • *Get or set type of analysis*
long	**Count**
	Get number of surface elements in the model
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*
	(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*
EMinMaxType	**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	*shear resistance*
scVEdMinusVRdc = 1	*resultant shear force minus shear resistance*
scVRdmax = 2	*maximum shear resistance*
scVEdDivVRdmax = 3	*resultant shear force divided by maximum shear resistance*
scaVEd = 4}	*angle of resultant shear force*

Shear capacity

Records / structures

RShearCapacities = (

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*
RShearCapacityValues	**scvCenterPoint**	*shear capacity at centre point*
RShearCapacityValues	**scvContourPoint1**	*shear capacity at contour point 1*
RShearCapacityValues	**scvContourPoint2**	*shear capacity at contour point 2*
RShearCapacityValues	**scvContourPoint3**	*shear capacity at contour point 3*
RShearCapacityValues	**scvContourPoint4**	*shear capacity at contour point 4*
RShearCapacityValues	**scvContourLineMidPoint1**	*shear capacity at contour line midpoint 1*
RShearCapacityValues	**scvContourLineMidPoint2**	*shear capacity at contour line midpoint 2*
RShearCapacityValues	**scvContourLineMidPoint3**	*shear capacity at contour line midpoint 3*
RShearCapacityValues	**scvContourLineMidPoint4**	*shear capacity at contour line midpoint 4*

)

RShearCapacityValues = (

double	**VRdc**	*shear resistance [kN/m]*
double	**VEdMinusVRdc**	*resultant shear force minus shear resistance [kN/m]*
double	**VRdmax**	*maximum shear resistance [kN/m]*
double	**VEdDivVRdmax**	*resultant shear force divided by maximum shear resistance [kN/m]*
double	**aVEd**	*angle of resultant shear force [°]*

)

Functions

long **GetShearCapacitiesByLoadCaseId** ([in] long **SurfaceId,** [in] long **LoadCaseId,**
[in] long **LoadLevel**, [in] EAnalysisType **AnalysisType,** [i/o] RShearCapacities* **ShearCapacities,**
[out] BSTR* **Combination**)

SurfaceId	*index of the surface element*
LoadCaseId	*load case index (0 < LoadCaseId ≤ 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 SurfaceId, otherwise an error code (errDatabaseNotReady, errNotSupportedByNationalDesignCode, errIndexOutOfBounds, sceLoadCaseIdIndexOutOfBounds, sceInvalidAnalysisType, sceInvalidCombinationOfLoadCaseAndLoadLevel).

---

long **GetShearCapacitiesByLoadCombinationId** ([in] long **SurfaceId,** [in] long **LoadCombinationId,**
[in] long **LoadLevel**, [in] EAnalysisType **AnalysisType,** [i/o] RShearCapacities* **ShearCapacities,**
[out] BSTR* **Combination**)

SurfaceId	*index of the surface element*
LoadCombinationId	*load combination index (0 < LoadCombinationId ≤ 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 SurfaceId, otherwise an error code (errDatabaseNotReady, errNotSupportedByNationalDesignCode, errIndexOutOfBounds, sceLoadCombinationIdIndexOutOfBounds, sceInvalidAnalysisType, sceInvalidCombinationOfLoadCombinationAndLoadLevel).

---

long **GetEnvelopeShearCapacities** ([in] long **SurfaceId,** [in] EMinMaxType **MinMaxType,**
[in] EAnalysisType **AnalysisType,** [in] EShearCapacity **Component,**
[i/o] RShearCapacities* **ShearCapacities,** [out] BSTR* **Combination**)

SurfaceId	*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 SurfaceId, 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**)

SurfaceId	*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*
LoadCaseOrCombinationId	*load case or load combination index of midpoint results, if index > IAxisVMLoadcases.count then Load combination index = LoadCaseOrCombinationId – IAxisVMLoadcases.count*
LoadLevel	*load level*

Envelope is identified by MinMaxType, Component and EnvelopeUID properties.If successful returns SurfaceId, otherwise an error code (errDatabaseNotReady, errNotSupportedByNationalDesignCode, errIndexOutOfBounds,sceInvalidAnalysisType).

---

long **GetCriticalShearCapacities** ([in] long **SurfaceId,** [in] EMinMaxType **MinMaxType,**
[in] ECombinationType **CombinationType,** [in] EAnalysisType **AnalysisType,**
[in] EShearCapacity **Component,** [i/o] RShearCapacities* **ShearCapacities,**
[out] BSTR* **Combination**)

SurfaceId	*index of the surface element*
MinMaxType	*Minimum or maximum value*
CombinationType	*Combination Type*
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*

If successful returns SurfaceId, otherwise an error code (errDatabaseNotReady, errNotSupportedByNationalDesignCode, errIndexOutOfBounds,sceInvalidAnalysisType).

---

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**)

SurfaceId	*index of the surface element*
MinMaxType	*Minimum or maximum value*
CombinationType	*Combination Type*
AnalysisType	*Type of Analysis*
Component	*Type of shear capacities*
ShearCapacities	*Shear capacities of the surface element*
CriticalCombinationType	*combination type corresponding to critical result*
Factors	*load factors of the critical load combination for midpoint results*
LoadCaseIds	*load case indexes of the critical load combination for midpoint results*

If successful returns SurfaceId, otherwise an error code (errDatabaseNotReady, errNotSupportedByNationalDesignCode, errIndexOutOfBounds,sceInvalidAnalysisType).

---

long **GetAllShearCapacitiesByLoadCaseId** ([in] long **LoadCaseId,** [in] long **LoadLevel**, [in] EAnalysisType **AnalysisType,** [out] SAFEARRAY(RShearCapacities)* **ShearCapacities,** [out] SAFEARRAY (BSTR)* **Combinations**)

LoadCaseId	*load case index (0 < LoadCaseId ≤ AxisVMLoadCases.Count)*
LoadLevel	*load level (increment) index*
AnalysisType	*Type of Analysis*
ShearCapacities	*Shear capacities of the surface element*
Combinations	*Array of strings with names of load cases.*

If successful returns number of surface elements, otherwise an error code (errDatabaseNotReady, errNotSupportedByNationalDesignCode, errIndexOutOfBounds, sceLoadCaseIdIndexOutOfBounds, sceInvalidAnalysisType, sceInvalidCombinationOfLoadCaseAndLoadLevel).

---

long **GetAllShearCapacitiesByLoadCombinationId** ([in] long **LoadCombinationId,**
[in] long **LoadLevel**, [in] EAnalysisType **AnalysisType,**
[out] SAFEARRAY(RShearCapacities)* **ShearCapacities,**
[out] SAFEARRAY (BSTR)* **Combinations**)

LoadCombinationId	*load combination index (0 < LoadCombinationId ≤ AxisVMLoadCombinations.Count)*
LoadLevel	*load level (increment) index*
AnalysisType	*Type of Analysis*
ShearCapacities	*Shear capacities of the surface element*
Combinations	*Array of strings with combination names.*

If successful returns number of surface elements, otherwise an error code (errDatabaseNotReady, errNotSupportedByNationalDesignCode, errIndexOutOfBounds, sceLoadCombinationIdIndexOutOfBounds, sceInvalidAnalysisType, sceInvalidCombinationOfLoadCombinationAndLoadLevel).

---

long **GetAllEnvelopeShearCapacities** ([in] EMinMaxType **MinMaxType,**
[in] EAnalysisType **AnalysisType,** [in] EShearCapacity **Component,**
[out] SAFEARRAY(RShearCapacities)* **ShearCapacities,**
[out] SAFEARRAY (BSTR)* **Combinations**)

MinMaxType	*Minimum or maximum value*
AnalysisType	*Type of Analysis*
Component	*Type of shear capacities*
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*

Envelope is identified by MinMaxType, Component and EnvelopeUID properties. If successful returns number of surface elements, otherwise an error code (errDatabaseNotReady, errNotSupportedByNationalDesignCode, errIndexOutOfBounds,sceInvalidAnalysisType).

---

long **GetAllCriticalShearCapacities** ([in] EMinMaxType **MinMaxType,**
[in] ECombinationType **CombinationType,** [in] EAnalysisType **AnalysisType,**
[in] EShearCapacity **Component,** [out] SAFEARRAY(RShearCapacities)* **ShearCapacities,**
[out] SAFEARRAY (BSTR)* **Combinations**)

MinMaxType	*Minimum or maximum value*
CombinationType	*Combination Type*
AnalysisType	*Type of Analysis*
Component	*Type of shear capacities*
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*

If successful returns number of surface elements, otherwise an error code (errDatabaseNotReady, errNotSupportedByNationalDesignCode, errIndexOutOfBounds,sceInvalidAnalysisType).

---

long **ShearCapacitiesByLoadCaseId** ([in] long **SurfaceId,** [i/o] RShearCapacities* **ShearCapacities,**
[out] BSTR* **Combination**)

SurfaceId	*index of the surface element*
ShearCapacities	*shear capacities of the surface element*
Combination	*string with name of load case.*

If successful returns SurfaceId, otherwise an error code (errDatabaseNotReady, errNotSupportedByNationalDesignCode, errIndexOutOfBounds, sceLoadCaseIdIndexOutOfBounds, sceInvalidAnalysisType, sceInvalidCombinationOfLoadCaseAndLoadLevel).

---

long **ShearCapacitiesByLoadCombinationId** ([in] long **SurfaceId,**
[i/o] RShearCapacities* **ShearCapacities,** [out] BSTR* **Combination**)

SurfaceId	*index of the surface element*
ShearCapacities	*shear capacities of the surface element*
Combination	*string with name of combination.*

If successful returns SurfaceId, otherwise an error code (errDatabaseNotReady, errNotSupportedByNationalDesignCode, errIndexOutOfBounds, sceLoadCombinationIdIndexOutOfBounds, sceInvalidAnalysisType, sceInvalidCombinationOfLoadCombinationAndLoadLevel).

---

long **EnvelopeShearCapacities** ([in] long **SurfaceId,** [i/o] RShearCapacities* **ShearCapacities,**
[out] BSTR* **Combination**)

SurfaceId	*index of the surface element*
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 SurfaceId, otherwise an error code (errDatabaseNotReady, errNotSupportedByNationalDesignCode, errIndexOutOfBounds,sceInvalidAnalysisType).

---

long **EnvelopeShearCapacities2** ([in] long **SurfaceId,** [i/o] RShearCapacities* **ShearCapacities,**
[out] long **LoadCaseOrCombinationId,** [out] long **LoadLevel**)

SurfaceId	*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*

Envelope is identified by MinMaxType, Component and EnvelopeUID properties.If successful returns SurfaceId, otherwise an error code (errDatabaseNotReady, errNotSupportedByNationalDesignCode, errIndexOutOfBounds,sceInvalidAnalysisType).

---

long **CriticalShearCapacities** ([in] long **SurfaceId,** [i/o] RShearCapacities* **ShearCapacities,**
[out] BSTR* **Combination**)

SurfaceId	*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*

If successful returns SurfaceId, otherwise an error code (errDatabaseNotReady, errNotSupportedByNationalDesignCode, errIndexOutOfBounds,sceInvalidAnalysisType).

---

long **CriticalShearCapacities2** ([in] long **SurfaceId,** [i/o] RShearCapacities* **ShearCapacities,**
[out] ECombinationType **CriticalCombinationType,** [out] SAFEARRAY(double) **Factors,**
[out] SAFEARRAY(long) **LoadCaseIds**)

CriticalCombinationType	*combination type corresponding to critical load combination*
Factors	*load factors of the critical load combination of midpoint results*
LoadCaseIds	*load case indexes of the critical load combination of midpoint results*

Similar to CriticalShearCapacities with more parameters described above.

---

long **AllShearCapacitiesByLoadCaseId** ([out] SAFEARRAY(RShearCapacities)* **ShearCapacities,**
[out] SAFEARRAY (BSTR)* **Combinations**)

ShearCapacities	*shear capacities of the surface element*
Combinations	*array of strings with name of load cases.*

If successful returns number of surface elements, otherwise an error code (errDatabaseNotReady, errNotSupportedByNationalDesignCode, errIndexOutOfBounds, sceLoadCaseIdIndexOutOfBounds, sceInvalidAnalysisType, sceInvalidCombinationOfLoadCaseAndLoadLevel).

---

long **AllShearCapacitiesByLoadCombinationId** (out] SAFEARRAY(RShearCapacities)*
ShearCapacities, [out] SAFEARRAY (BSTR)* **Combinations**)

ShearCapacities	*shear capacities of the surface element*
Combinations	*array of strings with combination names.*

If successful returns number of surface elements, otherwise an error code (errDatabaseNotReady, errNotSupportedByNationalDesignCode, errIndexOutOfBounds, sceLoadCombinationIdIndexOutOfBounds, sceInvalidAnalysisType, sceInvalidCombinationOfLoadCombinationAndLoadLevel).

---

long **AllEnvelopeShearCapacities** ([out] SAFEARRAY(RShearCapacities)* **ShearCapacities,**
[out] SAFEARRAY (BSTR)* **Combinations**)

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*

Envelope is identified by MinMaxType, Component and EnvelopeUID properties.If successful returns number of surface elements, otherwise an error code (errDatabaseNotReady, errNotSupportedByNationalDesignCode, errIndexOutOfBounds,sceInvalidAnalysisType).

---

long **AllCriticalShearCapacities** ([out] SAFEARRAY(RShearCapacities)* **ShearCapacities,**
[out] SAFEARRAY (BSTR)* **Combinations**)

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*

If successful returns number of surface elements, otherwise an error code (errDatabaseNotReady, errNotSupportedByNationalDesignCode, errIndexOutOfBounds, sceInvalidAnalysisType).

---

long **SetUserCreep** ([in] ELongBoolean **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 type of analysis*
EShearCapacity	**Component** • *Get or set type of shear capacity component*
long	**Count** *Get number of surface elements in the model, if 0 then results not calculated or invalid.*
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 < LoadCaseId ≤ AxisVMLoadCases.Count)*
long	**LoadCombinationId** • *Get or set load combination index* *(0 < LoadCombinationId ≤ 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** *Returns lbTrue if nonlinear analysis results consider concrete creep. More here…*

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	*line index is out of bounds*
steLoadCaseIndexOutofBounds = –100002	*LoadCaseId is out of bounds*
steLoadCombinationIndexOutofBounds = –100003	*LoadCombinationId is out of bounds*
steNotValidLineType = –100004	*IAxisVMLine.LineType is not compatible with the reader function*
steSectionIndexOutOfBounds = –100005	*section index is out of bounds*
steCombinationTypeNotValidForCurrentNationalDesignCode = –100006	*CombinationType is incompatible with the current design code*
steCOMError = –100007	*internal COM error when calling GetRecordInfoFromGUIDs,SafeArrayCreate Ex, SafeArrayAccesssData*
steLineHasNoSections = –100008	*line element has no cross-sections (e.g. link element)*
steNoValidLinesInTheModel = –100009	*no valid line elements in the model*
steLineStressComponentNotValidForThisLineType = –100010	*LineType is not compatible with the line stress component*
steInvalidAnalysisType = –100011	*AnalysisType is incompatible with the function*
steInvalidCombinationOfLoadCaseAndLoadLevel = –100012	*results are not available for the given LoadCaseId and LoadLevelOrModeShape or this*
steInvalidCombinationOfLoadCombinationAndLoadLevel = –100013	*results are not available for the given LoadCombinationId and LoadLevelOrModeShape or this*
steNoResultBlocksInTheModel = –100014	*no result blocks in the model*
steNodeIndexOutOfBounds = –100015	*node index out of bounds*
steSurfaceIndexOutOfBounds = –100016	*surface index out of bounds*
steNoSurfacesInTheModel = –100017	*no surface elements in the model*
steMemberIndexOutOfBounds = –100018	*member index is out of bounds*
steReadXLAMSurfaceStresses = –100019	*read stress results for XLAM elements with dedicated XLAM functions*
steInvalidSurfaceVertexType = –100020	*Invalid Surface Vertex Type*
steReadSurfaceStresses = –100021	*XLAM stress reading functions can return this, read normal surface stress results with dedicated surface stress reading functions*
steNotXLAMpanel = –100022	*XLAM efficiency reading functions can return this, read efficiencys with dedicated functions*
steXLAMmoduleNotAvailable= –100023	*XLAM efficiency reading functions can return this, read efficiencys with dedicated functions*
steStressPointIDOutOfBounds= –100024 }	*stress point's index is out of bounds*

Enumerated types

enum **ELineStress** = {

lsSmin = 0x00	*Smin cross-section minimum of the axial stress [kN/m²]*
lsSmax = 0x01	*Smax cross-section maximum of the axial stress [kN/m²]*
lsVmin = 0x02	*Vmin cross-section minimum of shear stress [kN/m²]*
lsVmax = 0x03	*Vmax cross-section maximum of shear stress [kN/m²]*
lsSomin = 0x04	*Somin cross-section minimum of VonMises stress [kN/m²]*
lfSomax = 0x05	*Somax cross-section maximum of VonMises stress [kN/m²]*
lfVymean = 0x06	*Vy mean shear stress in local y direction [kN/m²]*
lfVzmean = 0x07 }	*Vz mean shear stress in local z direction [kN/m²]*

Line stress component identifiers

enum **ESurfaceStress** = {

ssSxx = 0x00	*Sxx axial stress [kN/m²]*
ssSyy = 0x01	*Syy axial stress [kN/m²]*
ssSxy = 0x02	*Sxy shear stress [kN/m²]*
ssSxz = 0x03	*Sxz shear stress [kN/m²]*
ssSyz = 0x04	*Syz shear stress [kN/m²]*
ssSVM = 0x05	*VonMises stress [kN/m²]*
ssS1 = 0x06	*1st principal stress [kN/m²]*
ssS2 = 0x07	*2nd principal stress [kN/m²]*
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/m²]*
xssSyy_m_T = 0x01	*Syy flexural stresses at top [kN/m²]*
xssSxx_m_B = 0x02	*Sxx flexural stresses at bottom [kN/m²]*
xssSyy_m_B = 0x03	*Syy flexural stresses at bottom [kN/m²]*
xssSxx_n = 0x04	*Sxx axial stresses at centre [kN/m²]*
xssSyy_n = 0x05	*Syy axial stresses at centre [kN/m²]*
xssSxy_max = 0x06	*Syz shear stress [kN/m²]*
xssSxz_max = 0x07	*Sxz shear stress [kN/m²]*
xssSrx_max = 0x08	*rolling shear stress $\tau_{rx.max}$ [kN/m²]*
xssSry_max = 0x09 }	*rolling shear stress $\tau_{ry.max}$ [kN/m²]*

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	**lsvLineType**	*line type*
long	**lsvPointCount**	*number of cross-section's stress calculation points (SCPs)*
double	**lsvS1**	*axial stress in SCP 1 [kN/m^2]*
double	**lsvS2**	*axial stress in SCP 2 [kN/m^2]*
double	**lsvS3**	*axial stress in SCP 3 [kN/m^2]*
double	**lsvS4**	*axial stress in SCP 4 [kN/m^2]*
double	**lsvS5**	*axial stress in SCP 5 [kN/m^2]*
double	**lsvS6**	*axial stress in SCP 6 [kN/m^2]*
double	**lsvS7**	*axial stress in SCP 7 [kN/m^2]*
double	**lsvS8**	*axial stress in SCP 8 [kN/m^2]*
double	**lsvS9**	*axial stress in SCP 9 [kN/m^2]*
double	**lsvV1**	*shear stress in SCP 1 [kN/m^2]*
double	**lsvV2**	*shear stress in SCP 2 [kN/m^2]*
double	**lsvV3**	*shear stress in SCP 3 [kN/m^2]*
double	**lsvV4**	*shear stress in SCP 4 [kN/m^2]*
double	**lsvV5**	*shear stress in SCP 5 [kN/m^2]*
double	**lsvV6**	*shear stress in SCP 6 [kN/m^2]*
double	**lsvV7**	*shear stress in SCP 7 [kN/m^2]*
double	**lsvV8**	*shear stress in SCP 8 [kN/m^2]*
double	**lsvV9**	*shear stress in SCP 9 [kN/m^2]*
double	**lsvSo1**	*VonMises stress in SCP 1 [kN/m^2]*
double	**lsvSo2**	*VonMises stress in SCP 2 [kN/m^2]*
double	**lsvSo3**	*VonMises stress in SCP 3 [kN/m^2]*
double	**lsvSo4**	*VonMises stress in SCP 4 [kN/m^2]*
double	**lsvSo5**	*VonMises stress in SCP 5 [kN/m^2]*
double	**lsvSo6**	*VonMises stress in SCP 6 [kN/m^2]*
double	**lsvSo7**	*VonMises stress in SCP 7 [kN/m^2]*
double	**lsvSo8**	*VonMises stress in SCP 8 [kN/m^2]*
double	**lsvSo9**	*VonMises stress in SCP 9 [kN/m^2]*
double	**lsvSeff1**	*nonlinear effective stress in SCP 1 [kN/m^2]*
double	**lsvSeff2**	*nonlinear effective stress in SCP 2 [kN/m^2]*
double	**lsvSeff3**	*nonlinear effective stress in SCP 3 [kN/m^2]*
double	**lsvSeff4**	*nonlinear effective stress in SCP 4 [kN/m^2]*
double	**lsvSeff5**	*nonlinear effective stress in SCP 5 [kN/m^2]*
double	**lsvSeff6**	*nonlinear effective stress in SCP 6 [kN/m^2]*
double	**lsvSeff7**	*nonlinear effective stress in SCP 7 [kN/m^2]*
double	**lsvSeff8**	*nonlinear effective stress in SCP 8 [kN/m^2]*
double	**lsvSeff9**	*nonlinear effective stress in SCP 9 [kN/m^2]*
double	**lsvfy1**	*nonlinear actual yield strength in SCP 1 [kN/m^2]*
double	**lsvfy2**	*nonlinear actual yield strength in SCP 2 [kN/m^2]*
double	**lsvfy3**	*nonlinear actual yield strength in SCP 3 [kN/m^2]*
double	**lsvfy4**	*nonlinear actual yield strength in SCP 4 [kN/m^2]*
double	**lsvfy5**	*nonlinear actual yield strength in SCP 5 [kN/m^2]*
double	**lsvfy6**	*nonlinear actual yield strength in SCP 6 [kN/m^2]*
double	**lsvfy7**	*nonlinear actual yield strength in SCP 7 [kN/m^2]*
double	**lsvfy8**	*nonlinear actual yield strength in SCP 8 [kN/m^2]*
double	**lsvfy9**	*nonlinear actual yield strength in SCP 9 [kN/m^2]*
double	**lsvKih1**	*nonlinear utilization in SCP 1 [kN/m^2]*
double	**lsvKih2**	*nonlinear utilization in SCP 2 [kN/m^2]*
double	**lsvKih3**	*nonlinear utilization in SCP 3 [kN/m^2]*
double	**lsvKih4**	*nonlinear utilization in SCP 4 [kN/m^2]*
double	**lsvKih5**	*nonlinear utilization in SCP 5 [kN/m^2]*
double	**lsvKih6**	*nonlinear utilization in SCP 6 [kN/m^2]*
double	**lsvKih7**	*nonlinear utilization in SCP 7 [kN/m^2]*
double	**lsvKih8**	*nonlinear utilization in SCP 8 [kN/m^2]*
double	**lsvKih9**	*nonlinear utilization in SCP 9 [kN/m^2]*
double	**lsvVymean**	*mean shear stress in SCPs [kN/m^2]*
double	**lsvVzmean**	*mean shear stress in SCPs [kN/m^2]*
long	**lsvSmin**	*lsvSmin = index of SCP with minimum axial stress*
long	**lsvSmax**	*lsvSmax = index of SCP with maximum axial stress*
long	**lsvVmin**	*lsvVmin – 9 = index of SCP with minimum shear stress*
long	**lsvVmax**	*lsvVmax – 9 = index of SCP with maximum shear stress*
long	**lsvSomin**	*lsvSomin – 18 = index of SCP with minimum VonMises stress*
long	**lsvSomax**	*lsvSomax – 18 = index of SCP with maximum VonMises stress*

)

NOTE:

SCP - stress calculation point

RSurfaceStressValues = (

double	**ssvSxx**	*Sxx axial stress [kN/m²]*
double	**ssvSyy**	*Syy axial stress [kN/m²]*
double	**ssvSxy**	*Sxy shear stress [kN/m²]*
double	**ssvSxz**	*Sxz shear stress [kN/m²]*
double	**ssvSyz**	*Syz shear stress [kN/m²]*
double	**ssvSVM**	*SVM VonMises stress [kN/m²]*
double	**ssvS1**	*1st principal stress [kN/m²]*
double	**ssvS2**	*2st principal stress [kN/m²]*
double	**ssvAs**	*aS principal direction angle [°]*

)

RSurfaceStressValuesTMB = (

RSurfaceStressValues	**ssvTop**	*stress values at the top of the surface*
RSurfaceStressValues	**ssvMiddle**	*stress values in the middle layer of the surface*
RSurfaceStressValues	**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*
RSurfaceStressValuesTMB	**ssvtmbCenterPoint**	*stresses at the surface centerpoint*
RSurfaceStressValuesTMB	**ssvtmbContourPoint1**	*stresses at the 1st contour node*
RSurfaceStressValuesTMB	**ssvtmbContourPoint2**	*stresses at the 2nd contour node*
RSurfaceStressValuesTMB	**ssvtmbContourPoint3**	*stresses at the 3rd contour node*
RSurfaceStressValuesTMB	**ssvtmbContourPoint4**	*stresses at the 4th contour node*
RSurfaceStressValuesTMB	**ssvtmbContourLineMidPoint1**	*stresses at the 1st contour line midpoint*
RSurfaceStressValuesTMB	**ssvtmbContourLineMidPoint2**	*stresses at the 2nd contour line midpoint*
RSurfaceStressValuesTMB	**ssvtmbContourLineMidPoint3**	*stresses at the 3rd contour line midpoint*
RSurfaceStressValuesTMB	**ssvtmbContourLineMidPoint4**	*stresses at the 4th contour line midpoint*

)

RXLAMSurfaceStressValues = (

double	**xssvSxx_m_T**	*stress from moment in local xx direction at top[kN/m²]*
double	**xssvSyy_m_T**	*stress from moment in local yy direction at top[kN/m²]*
double	**xssvSxy_m_T**	*stress from moment in local xy direction at top[kN/m²]*
double	**xssvSxx_m_B**	*stress from moment in local xx direction at bottom[kN/m²]*
double	**xssvSyy_m_B**	*stress from moment in local yy direction at bottom [kN/m²]*
double	**xssvSxy_m_B**	*stress from moment in local xy direction at bottom [kN/m²]*
double	**xssvSxx_n**	*stress from axial force in local xx direction [kN/m²]*
double	**xssvSyy_n**	*stress from axial force in local yy direction [kN/m²]*
double	**xssvSxy_n**	*stress from axial force in local xy direction [kN/m²]*
double	**xssvSxz_max**	*shear stress in local x direction [kN/m²]*
double	**xssvSyz_max**	*shear stress in local y direction [kN/m²]*
double	**xssvSrx_max**	*rolling shear stress $\tau_{rx.max}$ [kN/m²]*
double	**xssvSry_max**)	*rolling shear stress $\tau_{ry.max}$ [kN/m²]*

RXLAMSurfaceStresses = (

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*
RXLAMSurfaceStressValues	**xssvCenterPoint**	*XLAM stresses at the surface centerpoint*
RXLAMSurfaceStressValues	**xssvContourPoint1**	*XLAM stresses at the 1st contour node*
RXLAMSurfaceStressValues	**xssvContourPoint2**	*XLAM stresses at the 2nd contour node*
RXLAMSurfaceStressValues	**xssvContourPoint3**	*XLAM stresses at the 3rd contour node*
RXLAMSurfaceStressValues	**xssvContourPoint4**	*XLAM stresses at the 4th contour node*
RXLAMSurfaceStressValues	**xssvContourLineMidPoint1**	*XLAM stresses at the 1st contour line midpoint*
RXLAMSurfaceStressValues	**xssvContourLineMidPoint2**	*XLAM stresses at the 2nd contour line midpoint*
RXLAMSurfaceStressValues	**xssvContourLineMidPoint3**	*XLAM stresses at the 3rd contour line midpoint*
RXLAMSurfaceStressValues	**xssvContourLineMidPoint4**	*XLAM stresses at the 4th contour line midpoint*

)

RXLAMSurfaceEfficiencyValues = (

double	**xsev_M_N_0**	*M-N efficiency in grain direction[-]*
double	**xsev_M_N_90**	*M-N efficiency perpendicular to grain direction[-]*
double	**xsev_V_T**	*shear and torsion efficiency[-]*
double	**xsev_Vr_N**	*rolling shear and normal force efficiency[-]*
double	**xsev_Max**	*Maximum efficiency [-]*

)

RXLAMSurfaceEfficiencies = (

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*
RXLAMSurfaceEfficiencyValues	**xsevCenterPoint**	*XLAM efficiencies at the surface centerpoint*
RXLAMSurfaceEfficiencyValues	**xsevContourPoint1**	*XLAM efficiencies at the 1st contour node*
RXLAMSurfaceEfficiencyValues	**xsevContourPoint2**	*XLAM efficiencies at the 2nd contour node*
RXLAMSurfaceEfficiencyValues	**xsevContourPoint3**	*XLAM efficiencies at the 3rd contour node*
RXLAMSurfaceEfficiencyValues	**xsevContourPoint4**	*XLAM efficiencies at the 4th contour node*
RXLAMSurfaceEfficiencyValues	**xsevContourLineMidPoint1**	*XLAM efficiencies at the 1st contour line midpoint*
RXLAMSurfaceEfficiencyValues	**xsevContourLineMidPoint2**	*XLAM efficiencies at the 2nd contour line midpoint*
RXLAMSurfaceEfficiencyValues	**xsevContourLineMidPoint3**	*XLAM efficiencies at the 3rd contour line midpoint*
RXLAMSurfaceEfficiencyValues	**xsevContourLineMidPoint4**	*XLAM efficiencies at the 4th contour line midpoint*

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 LineId 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 **LineId,** [in] long **SectionId,** [in] long **LoadCaseId,**
[in] long **LoadLevelOrTimeStep,** [in] EAnalysisType **AnalysisType,**
[i/o] RLineStressValues **Stress,** [out] double **PosX,** [out] BSTR **Combination**)

LineId	*line index (0 < LineId ≤AxisVMModel.Lines.Count)*
SectionId	*section index (0 < SectionId ≤AxisVMModel.Lines[LineId].SectionCount)*
LoadCaseId	*load case index*
LoadLevelOrTimeStep	*for load level (0 < LoadLevelOrTimeStep ≤ LoadLevelCount)* *for timestep (0 < LoadLevelOrTimeStep ≤ 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 LineId or an error code (errDatabaseNotReady or see EStressesError).

---

long **GetLineStressByLoadCombinationId** ([in] long **LineId,** [in] long **SectionId,**
[in] long **LoadCombinationId,** [in] long **LoadLevelOrTimeStep,**
[in] EAnalysisType **AnalysisType,** [i/o] RLineStressValues **Stress,**
[out] double **PosX,** [out] BSTR **Combination**)

LineId	*line index (0 < LineId ≤AxisVMModel.Lines.Count)*
SectionId	*section index (0 < SectionId ≤AxisVMModel.Lines[LineId].SectionCount)*
LoadCombinationId	*load combination index*
LoadLevelOrTimeStep	*for load level (0 < LoadLevelOrTimeStep ≤ LoadLevelCount)* *for timestep (0 < LoadLevelOrTimeStep ≤ 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 LineId or an error code (errDatabaseNotReady or see EStressesError).

---

long **GetEnvelopeLineStress** ([in] long **LineId,** [in] long **SectionId,** [in] EMinMaxType **MinMaxType,**
[in] EAnalysisType **AnalysisType,** [in] ELineStress **Component,** [i/o] RLineStressValues **Stress,**
[out] double **PosX,** [out] BSTR **Combination**)

LineId	*line index (0 < LineId ≤AxisVMModel.Lines.Count)*
SectionId	*section index (0 < SectionId ≤AxisVMModel.Lines[LineId].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 maximumdepending 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 LineId or an error code (errDatabaseNotReady or see EStressesError).

---

long **GetEnvelopeLineStress2** ([in] long **LineId,** [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**)

LineId	*line index (0 < LineId ≤ AxisVMModel.Lines.Count)*
SectionId	*section index (0 < SectionId ≤ AxisVMModel.Lines[LineId].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*
LoadCaseOrCombinationId	*load case or load combination index, if index is > IAxisVMLoadcases.count then Load combination index = LoadCaseOrCombinationId – IAxisVMLoadcases.count*
LoadLevel	*load level*

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 LineId or an error code (errDatabaseNotReady or see EStressesError).

---

long **GetCriticalLineStress** ([in] long **LineId,** [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**)

LineId	*line index (0 < LineId ≤ AxisVMModel.Lines.Count)*
SectionId	*section index (0 < SectionId ≤ AxisVMModel.Lines[LineId].SectionCount)*
MinMaxType	*Minimum or maximum value*
CombinationType	*combination type*
AnalysisType	*analysis type*
Component	*Stress component*
Stress	*stress 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 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 LineId or an error code (errDatabaseNotReady or see EStressesError).

---

long **GetCriticalLineStress2** ([in] long **LineId,** [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) **LoadCaseIds**)

LineId	*line index (0 < LineId ≤ AxisVMModel.Lines.Count)*
SectionId	*section index (0 < SectionId ≤ AxisVMModel.Lines[LineId].SectionCount)*
MinMaxType	*Minimum or maximum value*
CombinationType	*combination type*
AnalysisType	*analysis type*
Component	*Stress component*
Stress	*stress results*
PosX	*position of SectionId in m according to the local x direction*
CriticalCombinationType	*combination typecorresponding to critical stress*
Factors	*load factors of the critical load combination*
LoadCaseIds	*load case indexes of the critical load combination*

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 LineId or an error code (errDatabaseNotReady or see EStressesError).

---

long **LineStressByLoadCaseId** ([in] long **LineId,** [in] long **SectionId,** [i/o] RLineStressValues **Stress,** [out] double **PosX,** [out] BSTR **Combination**)

Similar to GetLineStressByLoadCaseId function but, some parameters are set in properties of this interface or IAxisVMResults.

---

long **LineStressByLoadCombinationId** ([in] long **LineId,** [in] long **SectionId,** [i/o] RLineStressValues **Stress,** [out] double **PosX,** [out] BSTR **Combination**)

Similar to GetLineStressByLoadCombinationId function but, some parameters are set in properties of this interface or IAxisVMResults.

---

long **EnvelopeLineStress** ([in] long **LineId,** [in] long **SectionId,** [i/o] RLineStressValues **Stress,** [out] double **PosX,** [out] BSTR **Combination**)

Similar to GetEnvelopeLineStress function but, some parameters are set in properties of this interface or IAxisVMResults.

---

long **EnvelopeLineStress2** ([in] long **LineId,** [in] long **SectionId,** [i/o] RLineStressValues **Stress,** [out] double **PosX,** [out] long **LoadCaseOrCombinationId,** [out] long **LoadLevel**)

Similar to GetEnvelopeLineStress2 function but, some parameters are set in properties of this interface or IAxisVMResults.

---

long **CriticalLineStress** ([in] long **LineId,** [in] long **SectionId,** [i/o] RLineStressValues **Stress,** [out] double **PosX,** [out] BSTR **Combination**)

Similar to GetCriticalLineStress function but, some parameters are set in properties of this interface or IAxisVMResults.

---

long **CriticalLineStress2** ([in] long **LineId,** [in] long **SectionId,** [i/o] RLineStressValues **Stress,** [out] double **PosX,** [out] ECombinationType **CriticalCombinationType,** [out] SAFEARRAY(double) **Factors,** [out] SAFEARRAY(long) **LoadCaseIds**)

Similar to GetCriticalLineStress2 function but, some parameters are set in properties of this interface or IAxisVMResults.

Single ELEMENT reader functions

long **LineStressesByLoadCaseId** ([in] long **LineId,**
[i/o] SAFEARRAY(RLineStressValues) **Stresses,** [out] SAFEARRAY(double) **PosX**)

LineId	*line index (0 < LineId ≤ AxisVMModel.Lines.Count)*
Stresses	*array of line stresses for the line element.* *Length of the array is SectionCount[LineId]*
PosX	*array of cross-section positions in m according to the local x direction* *Length of the array is SectionCount[LineId]*

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 **LineStressesByLoadCombinationId** ([in] long **LineId,**
[i/o] SAFEARRAY(RLineStressValues) **Stresses,** [out] SAFEARRAY(double) **PosX**)

LineId	*line index (0 < LineId ≤ AxisVMModel.Lines.Count)*
Stresses	*array of line stresses for the line element.* *Length of the array is SectionCount[LineId]*
PosX	*array of cross-section positions in m according to the local x direction* *Length of the array is SectionCount[LineId]*

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 **EnvelopeLineStresses** ([in] long **LineId,**
[i/o] SAFEARRAY(RLineStressValues) **Stresses,** [out] SAFEARRAY(double) **PosX**)

LineId	*line index (0 < LineId ≤ AxisVMModel.Lines.Count)*
Stresses	*array of envelope line stresses for the line element.* *Length of the array is SectionCount[LineId]*
PosX	*array of cross-section positions in m according to the local x direction* *Length of the array is SectionCount[LineId]*

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.
Returns the total number of cross-sections or an error code (errDatabaseNotReady or see EStressesError).

---

long **CriticalLineStresses** ([in] long **LineId,**
[i/o] SAFEARRAY(RLineStressValues) **Stresses,** [out] SAFEARRAY(double) **PosX**)

LineId	*line index (0 < LineId ≤ AxisVMModel.Lines.Count)*
Stresses	*array of critical line stresses for the line element.* *Length of the array is SectionCount[LineId]*
PosX	*array of cross-section positions in m according to the local x direction* *Length of the array is SectionCount[LineId]*

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.
Returns the total number of cross-sections or an error code (errDatabaseNotReady or see EStressesError).

Multiple element reader functions

long **AllLineStressesByLoadCaseId** ([out] SAFEARRAY(long) **SectionCounts,**
[i/o] SAFEARRAY(RLineStressValues) **Stresses,** [out] SAFEARRAY(double) **PosX**)

SectionCounts	*array containing the section counts of line elements.* *Length of the array = AxisVMModel.Lines.Count*
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*
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 stresses 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 EStressesError).

---

long **AllLineStressesByLoadCombinationId** ([out] SAFEARRAY(long) **SectionCounts,**
[i/o] SAFEARRAY(RLineStressValues) **Stresses,** [out] SAFEARRAY(double) **PosX**)

SectionCounts *array containing the section counts of line elements.*
Length of the array = AxisVMModel.Lines.Count

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

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 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).

---

long **AllEnvelopeLineStresses** ([out] SAFEARRAY(long) **SectionCounts,**
[i/o] SAFEARRAY(RLineStressValues) **Stresses,** [out] SAFEARRAY(double) **PosX**)

SectionCounts *array containing the section counts of line elements.*
Length of the array = AxisVMModel.Lines.Count

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

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 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.
Returns the total number of cross-sections or an error code (errDatabaseNotReady or see EStressesError).

---

long **AllCriticalLineStresses** ([out] SAFEARRAY(long) **SectionCounts,**
[i/o] SAFEARRAY(RLineStressValues) **Stresses,** [out] SAFEARRAY(double) **PosX**)

SectionCounts *array containing the section counts of line elements.*
Length of the array = AxisVMModel.Lines.Count

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

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 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.
Returns the total number of cross-sections or an error code (errDatabaseNotReady or see EStressesError).

Multiple BLOCK reader functions

long **LineStressesForResultBlocks** ([in] long **LineId,** [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 < SurfaceVertexId ≤AxisVMNodes.Count)*
svtContourLinePoint	*line index (0 < SurfaceVertexId ≤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 **GetSurfaceStressByLoadCaseId** ([in] long **SurfaceId,**
[in] ESurfaceVertexType **SurfaceVertexType,** [in] long **SurfaceVertexId,**
[in] ESurfaceStressPosition **SurfaceStressPosition,** [in] long **LoadCaseId,**
[in] long **LoadLevelOrTimeStep,** [in] EAnalysisType **AnalysisType,**
[i/o] RSurfaceStressValues **Stress,** [out] BSTR **Combination**)

SurfaceId	*surface index (0 < SurfaceId ≤AxisVMSurfaces.Count)*
SurfaceVertexType	*vertex type*
SurfaceVertexId	*vertex identifier, see here*
SurfaceStressPosition	*layer of the surface element (top, middle, bottom)*
LoadCaseId	*load case index*
LoadLevelOrTimeStep	*for load level (0 < LoadLevelOrTimeStep ≤ LoadLevelCount)* *for timestep (0 < LoadLevelOrTimeStep ≤ 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 SurfaceId or an error code (errDatabaseNotReady or see EStressesError).

---

long **GetSurfaceStressByLoadCombinationId** ([in] long **SurfaceId,**
[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**)

SurfaceId	*surface index (0 < SurfaceId ≤AxisVMSurfaces.Count)*
SurfaceVertexType	*vertex type*
SurfaceVertexId	*vertex identifier, see here*
SurfaceStressPosition	*layer of the surface element (top, middle, bottom)*
LoadCombinationId	*load combination index*
LoadLevel	*for load level (0 < LoadLevelOrTimeStep ≤ LoadLevelCount)*
AnalysisType	*analysis type*
Stress	*stress results*
Combination	*name of the load combination*

Retrieves stresses at a point and in one layer of a surface according to the LoadCaseId (and LoadLevelOrTimeStep) property. Returns SurfaceId or an error code (errDatabaseNotReady or see EStressesError).

---

long **GetEnvelopeSurfaceStress** ([in] long **SurfaceId,** [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**)

SurfaceId	*surface index (0 < SurfaceId ≤AxisVMSurfaces.Count)*
SurfaceVertexType	*vertex type*
SurfaceVertexId	*vertex identifier, see here*
SurfaceStressPosition	*layer of the surface element (top, middle, bottom)*
MinMaxType	*Minimum or maximum value*
AnalysisType	*analysis type*
Component	*surface stress component*
Stress	*stress results*
Combination	*load case or combination in which Component has its minimum or maximum*

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 SurfaceId or an error code (errDatabaseNotReady or see EStressesError).

---

long **GetEnvelopeSurfaceStress2** ([in] long **SurfaceId,** [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**)

SurfaceId	*surface index (0 < SurfaceId ≤ AxisVMSurfaces.Count)*
SurfaceVertexType	*vertex type*
SurfaceVertexId	*vertex identifier, see here*
SurfaceStressPosition	*layer of the surface element (top, middle, bottom)*
MinMaxType	*Minimum or maximum value*
AnalysisType	*analysis type*
Component	*surface stress component*
Stress	*stress results*
LoadCaseOrCombinationId	*load case or load combination index, if index is > IAxisVMLoadcases.count then Load combination index = LoadCaseOrCombinationId – IAxisVMLoadcases.count*
LoadLevel	*load case or combination in which Component has its minimum or maximum*

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 SurfaceId or an error code (errDatabaseNotReady or see EStressesError).

---

long **GetCriticalSurfaceStress** ([in] long **SurfaceId,** [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**)

SurfaceId	*surface index (0 < SurfaceId ≤ AxisVMSurfaces.Count)*
SurfaceVertexType	*vertex type*
SurfaceVertexId	*vertex identifier, see here*
SurfaceStressPosition	*layer of the surface element (top, middle, bottom)*
MinMaxType	*Minimum or maximum value*
CombinationType	*combination type*
AnalysisType	*analysis type*
Component	*surface stress component*
Stress	*stress results*
Combination	*critical combination in which Component has its minimum or maximum*

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 SurfaceId or an error code (errDatabaseNotReady or see EStressesError).

---

long **GetCriticalSurfaceStress2** ([in] long **SurfaceId,** [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) **LoadCaseIds**)

SurfaceId	*surface index (0 < SurfaceId ≤ AxisVMSurfaces.Count)*
SurfaceVertexType	*vertex type*
SurfaceVertexId	*vertex identifier, see here*
SurfaceStressPosition	*layer of the surface element (top, middle, bottom)*
MinMaxType	*Minimum or maximum value*
CombinationType	*combination type*
AnalysisType	*analysis type*
Component	*surface stress component*
Stress	*stress results*
CriticalCombinationType	*combination type corresponding to the critical load combination*
Factors	*load factors of the critical load combination*
LoadCaseIds	*load case indexes of the critical load combination*

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 SurfaceId or an error code (errDatabaseNotReady or see EStressesError).

---

long **SurfaceStressByLoadCaseId** ([in] long **SurfaceId,** [in] ESurfaceVertexType **SurfaceVertexType,** [in] long **SurfaceVertexId,** [in] ESurfaceStressPosition **SurfaceStressPosition,** [i/o] RSurfaceStressValues **Stress,** [out] BSTR **Combination**)

Similar to GetSurfaceStressByLoadCaseId function but, some parameters are set in properties of this interface or IAxisVMResults.

---

long **SurfaceStressByLoadCombinationId** ([in] long **SurfaceId,** [in] ESurfaceVertexType **SurfaceVertexType,** [in] long **SurfaceVertexId,** [in] ESurfaceStressPosition **SurfaceStressPosition,** [i/o] RSurfaceStressValues **Stress,** [out] BSTR **Combination**)

Similar to GetSurfaceStressByLoadCombinationId function but, some parameters are set in properties of this interface or IAxisVMResults.

---

long **EnvelopeSurfaceStress** ([in] long **SurfaceId,** [in] ESurfaceVertexType **SurfaceVertexType,** [in] long **SurfaceVertexId,** [in] ESurfaceStressPosition **SurfaceStressPosition,** [i/o] RSurfaceStressValues **Stress,** [out] BSTR **Combination**)

Similar to GetEnvelopeSurfaceStress function but, some parameters are set in properties of this interface or IAxisVMResults.

---

long **EnvelopeSurfaceStress2** ([in] long **SurfaceId,** [in] ESurfaceVertexType **SurfaceVertexType,** [in] long **SurfaceVertexId,** [in] ESurfaceStressPosition **SurfaceStressPosition,** [i/o] RSurfaceStressValues **Stress,** [out] long **LoadCaseOrCombinationId,** [out] long **LoadLevel**)

Similar to GetEnvelopeSurfaceStress2 function but, some parameters are set in properties of this interface or IAxisVMResults.

---

long **CriticalSurfaceStress** ([in] long **SurfaceId,** [in] ESurfaceVertexType **SurfaceVertexType,** [in] long **SurfaceVertexId,** [in] ESurfaceStressPosition **SurfaceStressPosition,** [i/o] RSurfaceForceValues **Stress,** [out] BSTR **Combination**)

Similar to GetCriticalSurfaceStress function but, some parameters are set in properties of this interface or IAxisVMResults.

long **CriticalSurfaceStress2** ([in] long **SurfaceId,** [in] ESurfaceVertexType **SurfaceVertexType,**
[in] long **SurfaceVertexId,** [in] ESurfaceStressPosition **SurfaceStressPosition,**
[i/o] RSurfaceForceValues **Stress,**
[out] ECombinationType **CriticalCombinationType**, [out] SAFEARRAY(double) **Factors,**
[out] SAFEARRAY(long) **LoadCaseIds**)

Similar to GetCriticalSurfaceStress2 function but, some parameters are set in properties of this interface or IAxisVMResults.

Single ELEMENT reader functions

long **SurfaceStressesByLoadCaseId** ([in] long **SurfaceId,** [i/o] RSurfaceStresses **Stresses**)

SurfaceId *surface index (0 < SurfaceId ≤ AxisVMModel.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 LoadCaseId (and LoadLevelOrTimeStep) property. Returns SurfaceId or an error code (errDatabaseNotReady or see EStressesError).

---

long **SurfaceStressesByLoadCombinationId** ([in] long **SurfaceId,**
[i/o] RSurfaceStresses **Stresses**)

SurfaceId *surface index (0 < SurfaceId ≤ AxisVMModel.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 SurfaceId or an error code (errDatabaseNotReady or see EStressesError).

---

long **EnvelopeSurfaceStresses** ([in] long **SurfaceId,** [i/o] RSurfaceStresses **Stresses**)

SurfaceId *surface index (0 < SurfaceId ≤ AxisVMModel.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 SurfaceId or an error code (errDatabaseNotReady or see EStressesError).

---

long **CriticalSurfaceStresses** ([in] long **SurfaceId,** [i/o] RSurfaceStresses **Stresses**)

SurfaceId *surface index (0 < SurfaceId ≤ AxisVMModel.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 SurfaceId or an error code (errDatabaseNotReady or see EStressesError).

Multiple element reader functions

long **AllSurfaceStressesByLoadCaseId** ([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 LoadCaseId (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 **SurfaceId,**
[i/o] SAFEARRAY(RResultBlockInfo) **ResultBlockInfo,**
[i/o] SAFEARRAY(RSurfaceStresses) **Stresses**)

SurfaceId *surface index (0 < SurfaceId ≤ 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 **SurfaceId,**
[in] ESurfaceVertexType **SurfaceVertexType,** [in] long **SurfaceVertexId,**
[in] ESurfaceStressPosition **SurfaceStressPosition,**
[i/o] SAFEARRAY(RResultBlockInfo) **ResultBlockInfo,**
[i/o] SAFEARRAY(RSurfaceStressValues) **Stresses**)

SurfaceId *surface index (0 < SurfaceId ≤ 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 ≤ AxisVMModel.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 ≤ AxisVMModel.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 ≤ AxisVMModel.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	*member index (0 < MemberId ≤ AxisVMModel.Members.Count)*
MinMaxType	*Minimum or maximum value*
CombinationType	*combination type*
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 critical stresses in each cross-section of a given element. Critical combination is identified by Component and MinMaxType properties (e.g. lsSmin). 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.
Returns the total number of cross-sections or an error code (errDatabaseNotReady or see EStressesError).

XLAM panel stresses

long **GetXLAMSurfaceStressByLoadCaseId** ([in] long **SurfaceId,**
[in] ESurfaceVertexType **SurfaceVertexType,** [in] long **SurfaceVertexId,**
[in] long **LoadCaseId,** [in] long **LoadLevelOrTimeStep,** [in] EAnalysisType **AnalysisType,**
[i/o] RXLAMSurfaceStressValues **Stress,** [out] BSTR **Combination**)

SurfaceId	*surface index (0 < SurfaceId ≤ AxisVMSurfaces.Count)*
SurfaceVertexType	*vertex type*
SurfaceVertexId	*vertex identifier, see here*
LoadCaseId	*load case index*
LoadLevelOrTimeStep	*for load level (0 < LoadLevelOrTimeStep ≤ LoadLevelCount)* *for timestep (0 < LoadLevelOrTimeStep ≤ TimeStepCount)*
AnalysisType	*analysis type*
Stress	*XLAM surface stress results*
Combination	*name of the load case*

Retrieves stresses at a point of a surface.Returns SurfaceId or an error code (EGeneralError or see EStressesError).

long **GetXLAMSurfaceStressByLoadCombinationId** ([in] long **SurfaceId,**
[in] ESurfaceVertexType **SurfaceVertexType,** [in] long **SurfaceVertexId,**
[in] long **LoadCombinationId,** [in] long **LoadLevel,** [in] EAnalysisType **AnalysisType,**
[i/o] RXLAMSurfaceStressValues **Stress,** [out] BSTR **Combination**)

SurfaceId	*surface index (0 < SurfaceId ≤ AxisVMSurfaces.Count)*
SurfaceVertexType	*vertex type*
SurfaceVertexId	*vertex identifier, see here*
LoadCombinationId	*load combination index*
LoadLevel	*for load level (0 < LoadLevelOrTimeStep ≤ LoadLevelCount)*
AnalysisType	*analysis type*
Stress	*XLAM surface stress results*
Combination	*name of the load case*

Retrieves XLAM stresses at a point of a surface. Returns SurfaceId or an error code (EGeneralError or see EStressesError).

long **GetEnvelopeXLAMSurfaceStress** ([in] long **SurfaceId,** [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**)

SurfaceId	*surface index (0 < SurfaceId ≤ 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*

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 SurfaceId or an error code (EGeneralError or see EStressesError).

---

long **GetCriticalXLAMSurfaceStress** ([in] long **SurfaceId,** [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) **LoadCaseIds**)

SurfaceId	*surface index (0 < SurfaceId ≤ 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*
LoadCaseIds	*load case indexes of the critical load combination*

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 SurfaceId or an error code (EGeneralError or see EStressesError).

---

long **GetXLAMSurfaceStressValuesForResultBlocks** ([in] long **SurfaceId,**
[in] ESurfaceVertexType **SurfaceVertexType,** [in] long **SurfaceVertexId,**
[in] EAnalysisType **AnalysisType,** [out] SAFEARRAY(RResultBlockInfo) **ResultBlockInfo,**
[out] SAFEARRAY(RXLAMSurfaceStressValues) **Stresses**)

SurfaceId	*surface index (0 < SurfaceId ≤AxisVMSurfaces.Count)*
SurfaceVertexType	*vertex type*
SurfaceVertexId	*vertex identifier, see here*
AnalysisType	*analysis type*
ResultBlockInfo	*array of description records for result blocks*
Stresses	*XLAM surface stress results*

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.
Returns the common length of ResultBlockInfo and Stresses arrays or an error code (EGeneralError or see EStressesError).

---

long **GetXLAMSurfaceStressesByLoadCaseId** ([in] long **SurfaceId,**
[in] long **LoadCaseId,** [in] long **LoadLevelOrTimeStep,** [in] EAnalysisType **AnalysisType,**
[i/o] RXLAMSurfaceStresses **Stresses,** [out] BSTR **Combination**)

SurfaceId	*surface index (0 < SurfaceId ≤AxisVMSurfaces.Count)*
LoadCaseId	*load case index*
LoadLevelOrTimeStep	*for load level (0 < LoadLevelOrTimeStep ≤ LoadLevelCount)* *for timestep (0 < LoadLevelOrTimeStep ≤ TimeStepCount)*
AnalysisType	*analysis type*
Stresses	*XLAM surface stress results*
Combination	*name of the load case*

Retrieves stresses of a surface. Returns SurfaceId or an error code (EGeneralError or see EStressesError).

---

long **GetXLAMSurfaceStressesByLoadCombinationId** ([in] long **SurfaceId,**
[in] long **LoadCombinationId,** [in] long **LoadLevel,** [in] EAnalysisType **AnalysisType,**
[i/o] RXLAMSurfaceStresses **Stresses,** [out] BSTR **Combination**)

SurfaceId	*surface index (0 < SurfaceId ≤AxisVMSurfaces.Count)*
LoadCombinationId	*load combination index*
LoadLevel	*for load level (0 < LoadLevelOrTimeStep ≤ LoadLevelCount)*
AnalysisType	*analysis type*
Stresses	*XLAM surface stress results*
Combination	*name of the load case*

Retrieves stresses of a surface. Returns SurfaceId or an error code (EGeneralError or see EStressesError).

---

long **GetEnvelopeXLAMSurfaceStresses** ([in] long **SurfaceId,** [in] long **EnvelopeUID,**
[in] EMinMaxType **MinMaxType,** [in] EAnalysisType **AnalysisType,**
[in] EXLAMSurfaceStress **Component,** [i/o] RXLAMSurfaceStresses **Stresses**)

SurfaceId	*surface index (0 < SurfaceId ≤AxisVMSurfaces.Count)*
EnvelopeUID	*unique envelope index*
MinMaxType	*Minimum or maximum value*
AnalysisType	*analysis type*
Component	*surface stress component*
Stresses	*XLAM surface stress results*

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 SurfaceId or an error code (EGeneralError or see EStressesError).

---

long **GetCriticalXLAMSurfaceStresses** ([in] long **SurfaceId,** [in] EMinMaxType **MinMaxType,**
[in] ECombinationType **CombinationType,** [in] EAnalysisType **AnalysisType,**
[in] EXLAMSurfaceStress **Component,**[i/o] RXLAMSurfaceStresses **Stresses**)

SurfaceId	*surface index (0 < SurfaceId ≤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 SurfaceId or an error code (EGeneralError or see EStressesError).

---

long **GetXLAMSurfaceStressesForResultBlocks** ([in] long **SurfaceId,**
[in] EAnalysisType **AnalysisType,** [out] SAFEARRAY(RResultBlockInfo) **ResultBlockInfo,**
[out] SAFEARRAY(RXLAMSurfaceStresses) **Stresses**)

SurfaceId	*surface index (0 < SurfaceId ≤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 **GetXLAMSurfaceEfficiencyByLoadCaseId** ([in] long **SurfaceId,**
[in] ESurfaceVertexType **SurfaceVertexType,** [in] long **SurfaceVertexId,**
[in] long **LoadCaseId,** [in] long **LoadLevelOrTimeStep,** [in] EAnalysisType **AnalysisType,**
[i/o] RXLAMSurfaceEfficiencyValues **Efficiency,** [out] BSTR **Combination**)

SurfaceId	*surface index (0 < SurfaceId ≤AxisVMSurfaces.Count)*
SurfaceVertexType	*vertex type*
SurfaceVertexId	*vertex identifier, see here*
LoadCaseId	*load case index*
LoadLevelOrTimeStep	*for load level (0 < LoadLevelOrTimeStep ≤ LoadLevelCount)* *for timestep (0 < LoadLevelOrTimeStep ≤ TimeStepCount)*
AnalysisType	*analysis type*
Efficiency	*XLAM surface efficiency*
Combination	*name of the load case*

Retrieves XLAM efficiencies at a point of a surface. Returns SurfaceId or an error code (EGeneralError or see EStressesError).

---

long **GetXLAMSurfaceEfficiencyByLoadCombinationId** ([in] long **SurfaceId,**
[in] ESurfaceVertexType **SurfaceVertexType,** [in] long **SurfaceVertexId,**
[in] long **LoadCombinationId,** [in] long **LoadLevelOrTimeStep,** [in] EAnalysisType **AnalysisType,**
[i/o] RXLAMSurfaceEfficiencyValues **Efficiency,** [out] BSTR **Combination**)

SurfaceId	*surface index (0 < SurfaceId ≤AxisVMSurfaces.Count)*
SurfaceVertexType	*vertex type*
SurfaceVertexId	*vertex identifier, see here*
LoadCombinationId	*load combination index*
LoadLevelOrTimeStep	*for load level (0 < LoadLevelOrTimeStep ≤ LoadLevelCount)* *for timestep (0 < LoadLevelOrTimeStep ≤ TimeStepCount)*
AnalysisType	*analysis type*
Efficiency	*XLAM surface efficiency*
Combination	*name of the load case*

Retrieves XLAM efficiencies at a point of a surface. Returns SurfaceId or an error code (EGeneralError or see EStressesError).

---

long **GetEnvelopeXLAMSurfaceEfficiency** ([in] long **SurfaceId,** [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**)

SurfaceId	*surface index (0 < SurfaceId ≤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 > 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 SurfaceId or an error code (EGeneralError or see EStressesError).

---

long **GetCriticalXLAMSurfaceEfficiency** ([in] long **SurfaceId,** [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) **LoadCaseIds**)

SurfaceId	*surface index (0 < SurfaceId ≤ AxisVMSurfaces.Count)*
SurfaceVertexType	*vertex type*
SurfaceVertexId	*vertex identifier, see here*
MinMaxType	*Minimum or maximum value*
CombinationType	*combination type*
AnalysisType	*analysis type*
Component	*surface efficiency component*
Efficiency	*XLAM surface efficiency*
CriticalCombinationType	*combination type corresponding to the critical load combination*
Factors	*load factors of the critical load combination*
LoadCaseIds	*load case indexes of the critical load combination*

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 SurfaceId or an error code (EGeneralError or see EStressesError).

---

long **GetCriticalXLAMSurfaceStress** ([in] long **SurfaceId,** [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) **LoadCaseIds**)

SurfaceId	*surface index (0 < SurfaceId ≤ AxisVMSurfaces.Count)*
SurfaceVertexType	*vertex type*
SurfaceVertexId	*vertex identifier, see here*
MinMaxType	*Minimum or maximum value*
CombinationType	*combination type*
AnalysisType	*analysis type*
Component	*surface efficiency component*
Efficiency	*XLAM surface efficiency*
CriticalCombinationType	*combination type corresponding to the critical load combination*
Factors	*load factors of the critical load combination*
LoadCaseIds	*load case indexes of the critical load combination*

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 SurfaceId or an error code (EGeneralError or see EStressesError).

---

long **GetXLAMSurfaceEfficienciesByLoadCaseId** ([in] long **SurfaceId,**
[in] long **LoadCaseId,** [in] long **LoadLevelOrTimeStep,** [in] EAnalysisType **AnalysisType,**
[i/o] RXLAMSurfaceEfficiencies **Efficiencies,** [out] BSTR **Combination**)

SurfaceId	*surface index (0 < SurfaceId ≤AxisVMSurfaces.Count)*
LoadCaseId	*load case index*
LoadLevelOrTimeStep	*for load level (0 < LoadLevelOrTimeStep ≤ LoadLevelCount)* *for timestep (0 < LoadLevelOrTimeStep ≤ TimeStepCount)*
AnalysisType	*analysis type*
Efficiencies	*XLAM surface efficiency results*
Combination	*name of the load case*

Retrieves efficiencies of an XLAM surface. Returns SurfaceId or an error code (EGeneralError or see EStressesError).

---

long **GetXLAMSurfaceEfficienciesByLoadCombinationId** ([in] long **SurfaceId,**
[in] long **LoadCombinationId,** [in] long **LoadLevelOrTimeStep,** [in] EAnalysisType **AnalysisType,** [i/o] RXLAMSurfaceEfficiencies **Efficiencies**)

SurfaceId	*surface index (0 < SurfaceId ≤AxisVMSurfaces.Count)*
LoadCombinationId	*load combination index*
LoadLevelOrTimeStep	*for load level (0 < LoadLevelOrTimeStep ≤ LoadLevelCount)* *for timestep (0 < LoadLevelOrTimeStep ≤ TimeStepCount)*
AnalysisType	*analysis type*
Efficiencies	*XLAM surface efficiency results*

Retrieves efficiencies of an XLAM surface. Returns SurfaceId or an error code (EGeneralError or see EStressesError).

---

long **GetXLAMSurfaceStressesByLoadCombinationId** ([in] long **SurfaceId,**
[in] long **LoadCombinationId,** [in] long **LoadLevel,** [in] EAnalysisType **AnalysisType,**
[i/o] RXLAMSurfaceEfficiencies **Efficiencies,** [out] BSTR **Combination**)

SurfaceId	*surface index (0 < SurfaceId ≤AxisVMSurfaces.Count)*
LoadCombinationId	*load combination index*
LoadLevel	*for load level (0 < LoadLevelOrTimeStep ≤ LoadLevelCount)*
AnalysisType	*analysis type*
Efficiencies	*XLAM surface efficiency results*
Combination	*name of the load case*

Retrieves efficiencies of an XLAM surface. Returns SurfaceId or an error code (EGeneralError or see EStressesError).

---

long **GetEnvelopeXLAMSurfaceEfficiencies** ([in] long **SurfaceId,** [in] EMinMaxType **MinMaxType,**
[in] EAnalysisType **AnalysisType,** [in] EXLAMSurfaceEfficiency **Component,**
[i/o] RXLAMSurfaceStresses **Efficiencies**)

SurfaceId	*surface index (0 < SurfaceId ≤AxisVMSurfaces.Count)*
MinMaxType	*Minimum or maximum value*
AnalysisType	*analysis type*
Component	*surface efficency component*
Efficiencies	*XLAM surface efficiency results*

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 SurfaceId or an error code (EGeneralError or see EStressesError).

---

long **GetCriticalXLAMSurfaceEfficiencies** ([in] long **SurfaceId,** [in] EMinMaxType **MinMaxType,** [in] ECombinationType **CombinationType,** [in] EAnalysisType **AnalysisType,** [in] EXLAMSurfaceEfficiency **Component,** [i/o] RXLAMSurfaceStresses **Efficiencies**)

SurfaceId	*surface index (0 < SurfaceId ≤ 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 SurfaceId 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] ELongBoolean **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] ELongBoolean **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	*name of the file with extension emf*
MemberId	*member index*
StressPointId	*stress point index*
LoadCaseId	*load case index*
LoadLevelOrTimeStep	*for load level (0 < LoadLevelOrTimeStep ≤ LoadLevelCount)* *for timestep (0 < LoadLevelOrTimeStep ≤ 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 member*
ColourMode	*colour mode*

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	*name of the file with extension emf*
MemberId	*member index*
StressPointId	*stress point index*
LoadCombinationId	*load combination index*
LoadLevelOrTimeStep	*for load level (0 < LoadLevelOrTimeStep ≤ LoadLevelCount)* *for timestep (0 < LoadLevelOrTimeStep ≤ 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 member*
ColourMode	*colour mode*

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 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*
long	**EnvelopeUID** • *Get or set the unique index of the envelope used in functions for reading envelope results*
ELongBoolean	**IndependentStressPoints** *When a RlineStressValues is returned for envelope or critical cases, the various stress points will have the respective extreme if IndependentStressPoints is lbTrue. The returned combination will still be for the queried component. If IndependentStressPoints is lbFalse, the various stress points will have the value corresponding to the querried extreme component's combination. The default value is False.*

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	**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)*
EXLAMSurfaceStress	**XLAMSurfaceStressComponent** • *Get or set XLAM panel surface stress component (for envelope or critical values)*
ELongBoolean	**UserCreep** *Returns lbTrue if nonlinear analysis results consider concrete creep. More here…*

IAxisVMSteelDesignResults

Interface for reading steel design results

Error codes

enum **ESteelDesignResultsError**= {

sdreCOMError = -100001	*COM server error*
sdreLoadCaseIdIndexOutOfBounds = -100002	*Invalid LoadCaseID while reading results*
sdreLoadCombinationIdIndexOutOfBounds = -100003	*Invalid CombinationID while reading results*
sdreInvalidAnalysisType = -100004	*Invalid type of results for used analysis*
sdreCombinationTypeNotValidForCurrentNationalDesignCode = -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

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 [m2]
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 [m3]
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 [m3]
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 [m3]
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 [m3]
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 [m2]
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 compresion 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 [m3]
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 [m3]
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 [m3]
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 [m3]
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 **GetEfficiencyAndCombinationByLoadCaseId** ([in] long **SteelDesignMemberId,** [in] long **LoadCaseId,** [in] long **LoadLevel**, [out] double **Efficiency,** [out] BSTR **Combination**)

SteelDesignMemberId	*index of steel 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 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 < LoadCaseId ≤ 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**)

SteelDesignMemberId	*index of steel 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 SteelDesignMemberId if successful, otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBounds, sdreInvalidAnalysisType, errCriticalCombinationNotAllowed, sdreCombinationTypeNotValidForCurrentNationalDesignCode).

---

long **GetSteelDesignResultsByLoadCaseId** ([in] long **SteelDesignMemberId,** [in] long **LoadCaseId,**
[in] long **LoadLevel**, [in] EAnalysisType **AnalysisType,** [out] long * **ResultsPerSection,**
[out] SAFEARRAY(RSteelDesignResult)* **SteelDesignResults,** [out] BSTR* **Combination**)

SteelDesignMemberId	*index of steel design member*
LoadCaseId	*load case index (0 < LoadCaseId ≤ AxisVMLoadCases.Count)*
LoadLevel	*load level (increment) index*
AnalysisType	*Type of Analysis*
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 load case*

*Returns ResultsPerSection*n where n is number of sections, otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBounds, sdreLoadCaseIdIndexOutOfBounds, sdreInvalidAnalysisType, sdreCOMError).*

---

long **GetSteelDesignResultsByLoadCaseId_Abs** ([in] long **SteelDesignMemberId,**
[in] long **LoadCaseId,** [in] long **LoadLevel**, [in] EAnalysisType **AnalysisType,**
[out] long * **ResultsPerSection,** [out] SAFEARRAY(RSteelDesignResult)* **SteelDesignResults,**
[out] BSTR* **Combination**)

Same as GetSteelDesignResultsByLoadCaseId but PosX is absolute distance from start node of the steel design member

---

long **GetSteelDesignResultsByLoadCombinationId** ([in] long **SteelDesignMemberId,**
[in] long **LoadCombinationId,** [in] long **LoadLevel**, [in] EAnalysisType **AnalysisType,**
[out] long * **ResultsPerSection,** [out] SAFEARRAY(RSteelDesignResult)* **SteelDesignResults,**
[out] BSTR* **Combination**)

SteelDesignMemberId	*index of steel design member*
LoadCombinationId	*load combination index (0 < LoadCombinationId ≤ AxisVMLoadCombinations.Count)*
LoadLevel	*load level (increment) index*
AnalysisType	*Type of Analysis*
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 **GetSteelDesignResultsByLoadCombinationId_Abs** ([in] long **SteelDesignMemberId,**
[in] long **LoadCombinationId,** [in] long **LoadLevel**, [in] EAnalysisType **AnalysisType,**
[out] long * **ResultsPerSection,** [out] SAFEARRAY(RSteelDesignResult)* **SteelDesignResults,**
[out] BSTR* **Combination**)

Same as GetSteelDesignResultsByLoadCombinationId, but PosX is absolute distance from start node of the steel design member

---

long **GetEnvelopeSteelDesignResults** ([in] long **SteelDesignMemberId,**
[in] EAnalysisType **AnalysisType,** [out] long* **ResultsPerSection,**
[out] SAFEARRAY(RSteelDesignResult)* **SteelDesignResults,** [out] BSTR* **Combination**)

SteelDesignMemberId	*index of steel design member*
AnalysisType	*Type of Analysis*
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*

*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 **GetEnvelopeSteelDesignResults_Abs** ([in] long **SteelDesignMemberId,**
[in] EAnalysisType **AnalysisType,** [out] long* **ResultsPerSection,**
[out] SAFEARRAY(RSteelDesignResult)* **SteelDesignResults,** [out] BSTR* **Combination**)

Same as GetEnvelopeSteelDesignResults , but PosX is absolute distance from start node of the steel design member

---

long **GetEnvelopeSteelDesignResults2** ([in] long **SteelDesignMemberId,**
[in] long **EnvelopeUID,** [in] EAnalysisType **AnalysisType,** [out] long* **ResultsPerSection,**
[out] SAFEARRAY(RSteelDesignResult)* **SteelDesignResults,** [out] BSTR* **Combination**)

SteelDesignMemberId	*index of steel design member*
EnvelopeUID	*unique index of the envelope used in functions for reading envelope results*
AnalysisType	*Type of Analysis*
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).*

---

long **GetCriticalSteelDesignResults** ([in] long **SteelDesignMemberId,**
[in] ECombinationType **CombinationType,** [in] EAnalysisType **AnalysisType,**
[out] long* **ResultsPerSection,** [out] SAFEARRAY(RSteelDesignResult)* **SteelDesignResults,**
[out] BSTR* **Combination**)

SteelDesignMemberId	*index of steel design member*
CombinationType	*Combination Type*
AnalysisType	*Type of Analysis*
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 **GetCriticalSteelDesignResults_Abs** ([in] long **SteelDesignMemberId,**
[in] ECombinationType **CombinationType,** [in] EAnalysisType **AnalysisType,**
[out] long* **ResultsPerSection,** [out] SAFEARRAY(RSteelDesignResult)* **SteelDesignResults,**
[out] BSTR* **Combination**)

Same as GetCriticalSteelDesignResults, but PosX is absolute distance from start node of the steel design member

---

long **GetAllSteelDesignResultsByLoadCaseId** ([in] long **LoadCaseId,** [in] long **LoadLevel**,
[in] EAnalysisType **AnalysisType,** [out] SAFEARRAY(long)* **SectionCounts,**
[out] long * **ResultsPerSection,** [out] SAFEARRAY(RSteelDesignResult)* **SteelDesignResults,**
[out] SAFEARRAY(BSTR)* **Combinations**)

LoadCaseId	*load case index (0 < LoadCaseId ≤ AxisVMLoadCases.Count)*
LoadLevel	*load level (increment) index*
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 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 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, sdreLoadCaseIdIndexOutOfBounds, sdreInvalidAnalysisType, sdreCOMError).*

---

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**)

LoadCombinationId	*load combination index (0 < LoadCombinationId ≤ AxisVMLoadCombinations.Count)*
LoadLevel	*load level (increment) index*
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 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 combination 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, sdreLoadCaseIdIndexOutOfBounds, sdreInvalidAnalysisType, sdreCOMError).*

---

long **GetAllEnvelopeSteelDesignResults** ([in] EAnalysisType **AnalysisType,**
[out] SAFEARRAY(long)* **SectionCounts,** [out] long* **ResultsPerSection,**
[out] SAFEARRAY(RSteelDesignResult)* **SteelDesignResults,**
[out] SAFEARRAY(BSTR)* **Combinations**)

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 of strings with names of critical combinations*

*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).*

---

long **GetAllCriticalSteelDesignResults** ([in] ECombinationType **CombinationType,** [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).*

---

long **SteelDesignResultsByLoadCaseId** ([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 loadcase*

*Returns ResultsPerSection*n where n is number of sections, otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBounds, sdreLoadCaseIdIndexOutOfBounds, sdreInvalidAnalysisType, sdreCOMError).*

---

long **SteelDesignResultsByLoadCombinationId** ([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*

*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 **AllSteelDesignResultsByLoadCaseId** ([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:* - *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, sdreLoadCaseIdIndexOutOfBounds, sdreInvalidAnalysisType, sdreCOMError).*

---

long **AllSteelDesignResultsByLoadCombinationId** ([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:*
- *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 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).*

---

long **AllEnvelopeSteelDesignResults** ([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:*
- *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 with names of combinations*

*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).*

---

long **AllCriticalSteelDesignResults** ([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:*
- *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 critical 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).*

Properties

EAnalysisType	**AnalysisType** • *Get or set type of analysis*
double	**Count** *Get number of steel design members in the model, if 0 then results not calculated or invalid.*
ECombinationType	**CombinationType** • *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 < LoadCaseId ≤ AxisVMLoadCases.Count)*
long	**LoadCombinationId** • *Get or set load combination index (0 < LoadCaseId ≤ 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, errIndexOutOfBounds, 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, errIndexOutOfBounds, tdreLoadCaseIdIndexOutOfBounds, tdreInvalidAnalysisType, tdreLoadCombinationIdIndexOutOfBounds).

---

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, errIndexOutOfBounds, tdreInvalidAnalysisType, sdreLoadCaseIdIndexOutOfBounds).

---

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 < LoadCaseId ≤ 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, errIndexOutOfBounds, tdreInvalidAnalysisType, tdreLoadCombinationIdIndexOutOfBounds).

---

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 **GetTimberDesignResultsByLoadCaseId** ([in] long **TimberDesignMemberId,** [in] long **LoadCaseId,**
[in] long **LoadLevel**, [in] EAnalysisType **AnalysisType,** [out] long **ResultsPerSection,**
[out] SAFEARRAY(RTimberDesignResult)* **TimberDesignResult,** [out] BSTR* **Combination**)

TimberDesignMemberId	*index of timber design member*
LoadCaseId	*load case index (0 < LoadCaseId ≤ 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* *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 **GetTimberDesignResultsByLoadCaseId_Abs** ([in] long **TimberDesignMemberId,**
[in] long **LoadCaseId,** [in] long **LoadLevel**, [in] EAnalysisType **AnalysisType,**
[out] long **ResultsPerSection,** [out] SAFEARRAY(RTimberDesignResult)* **TimberDesignResult,**
[out] BSTR* **Combination**)

Same as GetTimberDesignResultsByLoadCaseId, 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 < LoadCombinationId ≤ 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 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 **GetTimberDesignResultsByLoadCombinationId_Abs** ([in] long **TimberDesignMemberId,**
[in] long **LoadCombinationId,** [in] long **LoadLevel**, [in] EAnalysisType **AnalysisType,**
[out] long **ResultsPerSection,** [out] SAFEARRAY(RTimberDesignResult)* **TimberDesignResult,**
[out] BSTR* **Combination**)

Same as GetTimberDesignResultsByLoadCombinationId, but PosX is absolute distance from start node of the steel design member

---

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 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, errIndexOutOfBounds, tdreInvalidAnalysisType, tdreCOMError).*

---

long **GetEnvelopeTimberDesignResults_Abs** ([in] long **TimberDesignMemberId,**
[in] EAnalysisType **AnalysisType,** [out] long* **ResultsPerSection,**
[out] SAFEARRAY(RTimberDesignResult)* **TimberDesignResult,** [out] BSTR* **Combination**)

Same as GetEnvelopeTimberDesignResults, but PosX is absolute distance from start node of the steel design member.

---

long **GetEnvelopeTimberDesignResults2** ([in] long **TimberDesignMemberId,** [in] long **EnvelopeUID,**
[in] EAnalysisType **AnalysisType,** [out] long* **ResultsPerSection,**
[out] SAFEARRAY(RTimberDesignResult)* **TimberDesignResult,** [out] BSTR* **Combination**)

TimberDesignMemberId	*index of timber design member*
EnvelopeUID	*unique index of the envelope used in functions for reading envelope results*
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* *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).*

---

long **GetCriticalTimberDesignResults** ([in] long **TimberDesignMemberId,**
[in] ECombinationType **CombinationType,** [in] EAnalysisType **AnalysisType,**
[out] long* **ResultsPerSection,** [out] SAFEARRAY(RTimberDesignResult)* **TimberDesignResult,**
[out] BSTR* **Combination**)

TimberDesignMemberId	*index of timber design member*
CombinationType	*Combination Type*
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* *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 **GetCriticalTimberDesignResults_Abs** ([in] long **TimberDesignMemberId,**
[in] ECombinationType **CombinationType,** [in] EAnalysisType **AnalysisType,**
[out] long* **ResultsPerSection,** [out] SAFEARRAY(RTimberDesignResult)* **TimberDesignResult,**
[out] BSTR* **Combination**)

Same as GetCriticalTimberDesignResults, but PosX is absolute distance from start node of the steel design member.

---

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**)

LoadCaseId	*load case index (0 < LoadCaseId ≤ AxisVMLoadCases.Count)*
LoadLevel	*load level (increment) index*
AnalysisType	*Type of Analysis*
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 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 timber design member* - *Until the last timber design member* *See here* *PosX is relative distance from start node of the line*
Combinations	*Array of loadcase names*

*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, tdreLoadCaseIdIndexOutOfBounds).*

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**)

LoadCombinationId	*load combination index (0 < LoadCombinationId ≤ AxisVMLoadCombinations.Count)*
LoadLevel	*load level (increment) index*
AnalysisType	*Type of Analysis*
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 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 timber design member* - *Until the last timber design member* *See here* *PosX is relative distance from start node of the line*
Combinations	*Array of loadcase names*

*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, tdreLoadCombinationIdIndexOutOfBounds).*

long **GetAllEnvelopeTimberDesignResults** ([in] EAnalysisType **AnalysisType,**
[out] SAFEARRAY(long)* **SectionCounts,** [out] long* **ResultsPerSection,** [out] SAFEARRAY(RTimberDesignResult) * **TimberDesignResults,** [out] SAFEARRAY(BSTR)* **Combinations**)

AnalysisType	*Type of Analysis*
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 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 timber design member* *Until the last timber design member. See here* *PosX is relative distance from start node of the line*
Combinations	*Array of strings with names of critical combinations*

*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).*

---

long **GetAllCriticalTimberDesignResults** ([in] ECombinationType **CombinationType,**
[in] EAnalysisType **AnalysisType,** [out] SAFEARRAY(long) * **SectionCounts,**
[out] long * **ResultsPerSection,** [out] SAFEARRAY(RTimberDesignResult) * **TimberDesignResults,**
[out] SAFEARRAY(BSTR)* **Combinations**)

CombinationType	*Combination Type*
AnalysisType	*Type of Analysis*
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 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 timber design member* *Until the last timber 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 timber design members and n is number of sections. Otherwise returns an error code (errDatabaseNotReady, tdreInvalidAnalysisType, tdreCOMError, tdreCombinationTypeNotValidForCurrentNationalDesignCode).*

---

long **TimberDesignResultsByLoadCaseId** ([in] long **TimberDesignMemberId,**
[out] long * **ResultsPerSection,** [out] SAFEARRAY(RTimberDesignResult) * **TimberDesignResults,**
[out] BSTR* **Combination**)

TimberDesignMemberId	*index of 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 **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 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 **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 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 **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 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, 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 ≤ AxisVMLoadCases.Count

long **LoadCombinationId** • *Get or set the load combination index*

0 < LoadCaseId ≤ 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** = {

veeLoadCaseIdIndexOutOfBounds = –100001	*LoadCaseId is out of bounds*
veeInvalidCombinationOfLoadCaseAndTimeStep = –100002	*invalid combination of LoadCaseId and TimeStep*
veeInvalidAnalysisType = –100003	*AnalysisType is incompatible with the function*
veeLoadCombinationHasNoDynamicResult = –100004 }	*dynamic results for the LoadCaseId 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 velocit*
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 **NodalVelocityByLoadCaseId** ([in] long **NodeId,**
[i/o] RVelocityValues **VelocityValues,** [out] BSTR **Combination**)

NodeId	*node index or line midpoint index* *(0 < NodeId ≤ AxisVMModel.Nodes.Count) or a value returned by AxisVMModel.Lines.MidpointId[LineIndex]*
VelocityValues	*velocity results*
Combination	*name of the load case*

Retrieves velocity values of a node or line midpoint according to the LoadCaseId (and TimeStep) property. Returns NodeId if successful l, otherwise an error code code (errDatabaseNotReady, errIndexOutOfBounds, veeLoadCaseIdIndexOutOfBounds, veeInvalidCombinationOfLoadCaseAndTimeStep, veeInvalidAnalysisType, veeLoadCombinationHasNoDynamicResult).

long **EnvelopeNodalVelocity** ([in] long **NodeId,**
[i/o] RVelocityValues **VelocityValues,** [out] BSTR **Combination**)

NodeId	*node index or line midpoint index* *(0 < NodeId ≤ AxisVMModel.Nodes.Count) or a value returned by AxisVMModel.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 l, otherwise an error code (errDatabaseNotReady, errIndexOutOfBounds, veeLoadCaseIdIndexOutOfBounds, 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 < LoadCaseId ≤ AxisVMModel.LoadCases.Count)*
EMinMaxType	**MinMaxType** • *Get or set if minimum or maximum values of the component should be read*
long	**TimeStep** • *Get or set time step increment.*

IAxisVMVerticalDisplacements

Vertical displacements results. It should be used only when vertical displacements are available in the model. See the “6.1.6.1 Nonlinear calculation of total deflection (wtot) for RC plates” in the user manual for the requirements.

Error codes

enum **EVerticalDisplacementsError** = {

vdeNoNonlinearResults = –100001	*missing nonlinear results*
vdeNoVerticalDisplacements = –100002}	*not all conditions were met for vertical displacements*

Enumerated types

enum **EVerticalDisplacement** = {

vd_w1 = 1	*initial part of the deflection under permanent loads [m]*
vd_w2 = 2	*long-term part of the deflection under permanent loads [m]*
vd_w3 = 3	*additional part of the deflection due to the variable actions of the relevant combinations [m]*
vd_wtot = 4	*total deflection [m]*
vd_wbij = 5 }	*wbij =w2+w3 [m]*

Records / structures

RVerticalDisplacementValues = (

double	**w1**	*initial part of the deflection under permanent loads [m]*
double	**w2**	*long-term part of the deflection under permanent loads [m]*
double	**w3**	*additional part of the deflection due to the variable actions of the relevant combinations [m]*
double	**wtot**	*total deflection [m]*
double	**wbij**	*wbij =w2+w3 [m]*

)

Functions

long **AllEnvelopeNodalDisplacements** ([out] SAFEARRAY(RVerticalDisplacementValues) * **VerticalDisplacements**)

VerticalDisplacements *the vertical displacements in all the nodes and midpoints. The first AxisVMModel.Nodes.Count items will be for the nodes, the next AxisVMModel.Lines.MidpointCount items for the midpoints*

Retrieves the vertical displacement values for all nodes and midpoints in one step. The envelope and component has to be set prior this call through the EnvelopeUID and Component properties. Returns the length of VerticalDisplacements if successful, otherwise an error code code (errDatabaseNotReady, vdeNoNonlinearResults , vdeNoVerticalDisplacements , errInvalidEnvelopeUID, errEnvelopeIdOutOfBounds, deNoNodesInTheModel).

---

long **EnvelopeNodalDisplacement** ([in] long **NodeId,** [i/o] RVerticalDisplacementValues **VerticalDisplacement,** [out] BSTR **Combination**)

NodeId	*node index or line midpoint index (0 < NodeId ≤ AxisVMModel.Nodes.Count) or a value returned by AxisVMModel.Lines.MidpointId[LineIndex]*
VerticalDisplacement	*the vertical displacements*
Combination	*name of the load combination*

Retrieves the vertical displacements of a node or line midpoint. The envelope and component has to be set prior this call through the EnvelopeUID and Component properties. Returns NodeId if successful l, otherwise an error code (errDatabaseNotReady, vdeNoNonlinearResults , vdeNoVerticalDisplacements , errInvalidEnvelopeUID, errEnvelopeIdOutOfBounds, deNoNodesInTheModel).

---

long **GetAllEnvelopeNodalDisplacements** ([in] long **EnvelopeId,** [in] EVerticalDisplacement **Component,** [out] SAFEARRAY(RVerticalDisplacementValues) * **VerticalDisplacements**)

EnvelopeId	*envelope UID, returned by Envelopes.EnvelopeUID*
Component	*component on which the extreme values are based*

VerticalDisplacements *the vertical displacements in all the nodes and midpoints. The first AxisVMModel.Nodes.Count items will be for the nodes, the next AxisVMModel.Lines.MidpointCount items for the midpoints*

Retrieves the vertical displacement values for all nodes and midpoints in one step. Returns the length of VerticalDisplacements if successful, otherwise an error code code (errDatabaseNotReady, vdeNoNonlinearResults , vdeNoVerticalDisplacements , errInvalidEnvelopeUID, errEnvelopeIdOutOfBounds, deNoNodesInTheModel).

---

long **GetEnvelopeNodalDisplacement** ([in] long **NodeId,** ([in] long **EnvelopeId,** [in] EVerticalDisplacement **Component,** [i/o] RVerticalDisplacementValues **VerticalDisplacement,** [out] BSTR **Combination**)

NodeId *node index or line midpoint index*
(0 < NodeId ≤ AxisVMModel.Nodes.Count) or a value returned by AxisVMModel.Lines.MidpointId[LineIndex]
EnvelopeId *envelope UID, returned by Envelopes.EnvelopeUID*
Component *component on which the extreme values are based*
VerticalDisplacement *the vertical displacements*
Combination *name of the load combination*

Retrieves the vertical displacements of a node or line midpoint. Returns NodeId if successful l, otherwise an error code (errDatabaseNotReady, vdeNoNonlinearResults , vdeNoVerticalDisplacements , errInvalidEnvelopeUID, errEnvelopeIdOutOfBounds, deNoNodesInTheModel).

Properties

EVerticalDisplacement **Component** • *Get or set the vertical displacement component, on which the extreme values.will be based. Used only for AllEnvelopeNodalDisplacements and EnvelopeNodalDisplacement.*

long **EnvelopeUID** • *Get or set the envelopeUID. This value is returned by Envelopes.EnvelopeUID. Used only for AllEnvelopeNodalDisplacements and EnvelopeNodalDisplacement.*

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	*Rebat 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 deign 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*

)

RRebarSteelGrade = (

ENationalDesignCode	**NationalDesignCode**	*National Design Code*
double	**E**	*Young's modulus of elasticity [kN/m²]*
RRebarSteelGrade_EC_ITA	**RebarSteelGrade_EC_ITA**	*Rebar steel grade parameters according to EC & ITA national design codes*
RRebarSteelGrade_MSZ	**RebarSteelGrade_MSZ**	*Rebar steel grade parameters according MSZ national design codes*
RRebarSteelGrade_STAS	**RebarSteelGrade_STAS**	*Rebar steel grade parameters according to STAS national design codes*
RRebarSteelGrade_DIN	**RebarSteelGrade_DIN**	*Rebar steel grade parameters according to DIN national design codes*
RRebarSteelGrade_SIA	**RebarSteelGrade_SIA**	*Rebar steel grade parameters according to SIA national design codes*
RRebarSteelGrade_NEN	**RebarSteelGrade_NEN**	*Rebar steel grade parameters according to NEN national design codes*

)

Functions

long **Add** ([in] BSTR **Name,** [i/o] RRebarSteelGrade **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] ENationalDesignCode **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] ENationalDesignCode **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, errIndexOutOfBounds).

---

long **GetData** ([in] long **Index,** [out] 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*

Get rebar steel material parameters. If successful, returns the rebar steel grade index, otherwise returns an error code(errDatabaseNotReady, errIndexOutOfBounds).

---

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*

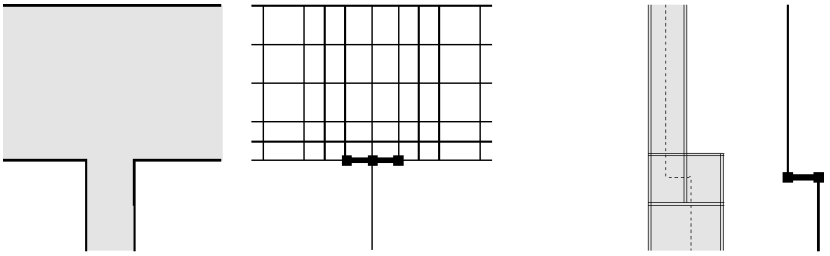
UID *unique index of the rebar steel grade*

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	*array with LineIDs is empty*
rbeNoLinesAreSelected = -100002	*when none of the lines is selected*
rbeLineIndexOutOfBounds = -100003}	*when LineID is out bounds*

Functions

long **Add**([in] SAFEARRAY(long) **LineIds**)

LineIds *Array with LineIDs*

If successful returns number of rigid bodies, otherwise returns an error code (errDatabaseNotReady, rbeLineIndexOutOfBounds)

---

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

---

long **AddSelectedLines**

If successful returns number of rigid bodies, otherwise returns an error code (errDatabaseNotReady, rbeNoLinesAreSelected)

---

long **Clear**

If successful returns number of rigid bodies before delete, otherwise returns an error code (errDatabaseNotReady)

---

long **DefineSelectedLinesAsRigidBody**

If successful returns number of rigid bodies, otherwise returns an error code (errDatabaseNotReady, rbeNoLinesAreSelected)

---

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

Index *index of the rigid body, 1 ≤ Index ≤ Count*

If successful returns index, otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBounds)

---

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

Index *index of the rigid body, 1 ≤ Index ≤ Count*

LineIds *Array with LineIDs*

If successful returns number of lines in rigid body with index Index, otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBounds)

---

long **RemoveLinesFromRigidBodies** ([in] SAFEARRAY(long) **LineIds**)

LineIds *Array with LineIDs*

If successful returns number of rigid bodies, otherwise returns an error code (errDatabaseNotReady, rbeLineIndexOutOfBounds, rbeLineListIsEmpty)

---

long **RemoveLinesFromRigidBodies_vb** (Visual Basic compatible function of **RemoveLinesFromRigidBodies**)

Properties

long **Count** *Get number of rigid bodies in the model.*

IAxisVMSections

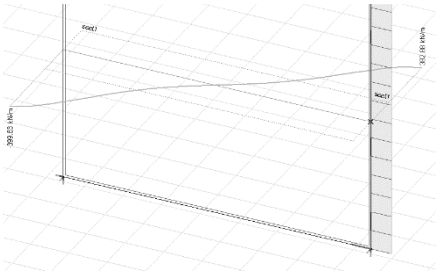
Interface for sections on the model.

Enumerated types

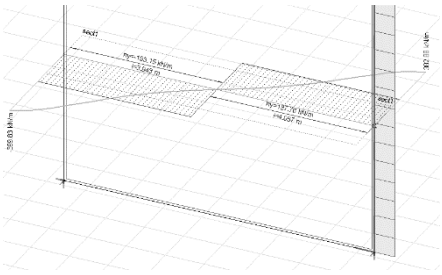
enum **ESectionType** = {
stPlane= 0x00
stSegment= 0x01 }

enum **ESectionDisplayMode**= {

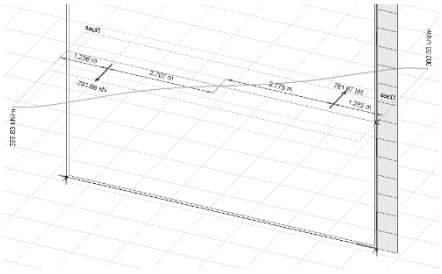
sdmDiagramOnly = 0x00 — *Displays diagram only*



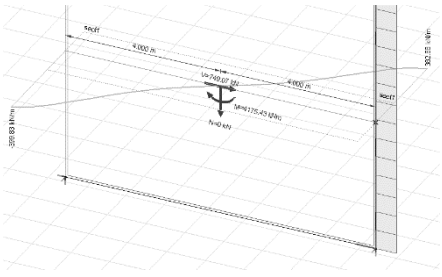
sdmDiagramAvg = 0x01 — *Displays average diagram of positive and negative values.*



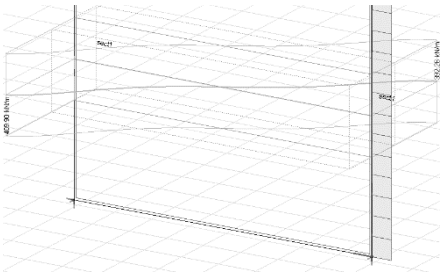
sdmDiagramRes = 0x02 — *Displays diagram and resultant value with position.*



sdmResultant = 0x03 — *Displays resultant values.*



sdmDiagramSegWidth = 0x04 } — *Displays diagram with segmented width*



enum **EResultType** = {

rtLoadCase = 0x00	*results of a load case*
rtLoadCombination = 0x01	*results of a load combination*
rtEnvelope = 0x02	*results of a specific envelope*
rtCritical = 0x03 }	*critical results*

enum **ESectionSegmentChainIntegratedResultant**= {

sscir_N = 0x00	*normal force [kN]*
sscir_Q = 0x01	*shear force [kN]*
sscir_M = 0x02 }	*bending moment [kNm]*

enum **EResultComponent** = {

ELineStress**:**

rc_lsSmin = 0	*[Smin] cross-section minimum of the axial stress [kN/m²]*
rc_lsSmax = 1	*[Smax] cross-section maximum of the axial stress [kN/m²]*
rc_lsVmin = 2	*[Vmin] cross-section minimum of shear stress [kN/m²]*
rc_lsVmax = 3	*[Vmax] cross-section maximum of shear stress [kN/m²]*
rc_lsSomin = 4	*[Somin] cross-section minimum of VonMises stress [kN/m²]*
rc_lsSomax = 5	*[Somax] cross-section maximum of VonMises stress [kN/m²]*
rc_lsVymean = 6	*[Vy] mean shear stress in local y direction [kN/m²]*
rc_lsVzmean = 7	*[Vz] mean shear stress in local z direction [kN/m²]*
rc_lsSminMax = 8	*[Sminmax] cross-section minimum and maximum of the axial stress [kN/m²]*
rc_lsVminMax = 9	*[Vminmax] cross-section minimum and maximum of the shear stress, [kN/m²]*
rc_lsSominMax = 10	*[Sominmax] cross-section minimum and maximum of the VonMises stress [kN/m²]*
rc_lsSeffMin = 11	*[(NL) Seff Min] cross-section minimum of the effective stress (for nonlinear materials) [kN/m²]*
rc_lsSeffMax = 12	*[(NL) Seff Max] cross-section maximum of the effective stress (for nonlinear materials) [kN/m²]*
rc_lsSeffMinMax = 13	*[(NL) Seff MinMax] cross-section minimum and maximum of the effective stress (for nonlinear materials) [kN/m²]*
rc_lsfyMin = 14	*[(NL) fymin] cross-section minimum of the actual yield strength (for nonlinear materials) [kN/m²]*
rc_lsfyMax = 15	*[(NL) fymax] cross-section maximum of the actual yield strength (for nonlinear materials) [kN/m²]*
rc_lsfyMinMax = 16	*[(NL) fyminmax] cross-section minimum and maximum of the actual yield strength (for nonlinear materials) [kN/m²]*
rc_lsUMin = 17	*[(NL) Utilization min] cross-section minimum of the utilization (for nonlinear materials) [ ]*
rc_lsUMax = 18	*[(NL) Utilization max] cross-section maximum of the utilization (for nonlinear materials) [ ]*
rc_lsUMinMax = 19	*[(NL) Utilization minmax] cross-section minimum and maximum of the utilization (for nonlinear materials) [ ]*
rc_lsBMin = 20	*[(NL) B min] cross-section minimum of the back stress (for nonlinear materials) [kN/m²]*
rc_lsBMax = 21	*[(NL) B max] cross-section maximum of the back stress (for nonlinear materials) [kN/m²]*
rc_lsBMax = 22	*[(NL) B minmax] cross-section minimum and maximum of the back stress (for nonlinear materials) [kN/m²]*

ESurfaceStress**:**

rc_ssSxx_Top = 100	*[Sxx] axial stress in local x direction at the top [kN/m²]*
rc_ssSyy_Top = 101	*[Syy] axial stress in local y direction at the top [kN/m²]*
rc_ssSxy_Top = 102	*[Sxy] torsional/shear stress at the top [kN/m²]*

rc_ssSxz_Top = 103 *[Sxz] torsional/shear stress at the top [kN/m²]*
rc_ssSyz_Top = 104 *[Syz] torsional/shear stress at the top [kN/m²]*
rc_ssSVM_Top = 105 *[SVM] VonMises stress at the top [kN/m²]*
rc_ssS1_Top = 106 *[S1] 1st principal stress at the top [kN/m²]*
rc_ssS2_Top = 107 *[S2] 2st principal stress at the top [kN/m²]*
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/m2]*
rc_ssSyy_Middle = 111 *[Syy] axial stress in local y direction in the middle [kN/m2]*
rc_ssSxy_Middle = 112 *[Sxy] torsional/shear stress in the middle [kN/m2]*
rc_ssSxz_Middle = 113 *[Sxz] torsional/shear stress in the middle [kN/m2]*
rc_ssSyz_Middle = 114 *[Syz] torsional/shear stress in the middle [kN/m2]*
rc_ssSVM_Middle = 115 *[SVM] VonMises stress in the middle [kN/m2]*
rc_ssS1_Middle = 116 *[S1] 1st principal stress in the middle [kN/m2]*
rc_ssS2_Middle = 117 *[S2] 2st principal stress in the middle [kN/m2]*
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/m2]*
rc_ssSyy_Bottom = 121 *[Syy] axial stress in local y direction at the bottom [kN/m2]*
rc_ssSxy_Bottom = 122 *[Sxy] torsional/shear stress at the bottom [kN/m2]*
rc_ssSxz_Bottom = 123 *[Sxz] torsional/shear stress at the bottom [kN/m2]*
rc_ssSyz_Bottom = 124 *[Syz] torsional/shear stress at the bottom [kN/m2]*
rc_ssSVM_Bottom = 125 *[SVM] VonMises stress at the bottom [kN/m2]*
rc_ssS1_Bottom = 126 *[S1] 1st principal stress at the bottom [kN/m2]*
rc_ssS2_Bottom = 127 *[S2] 2st principal stress at the bottom [kN/m2]*
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/m²]*
rc_ssSeff_Top = 130 *[(NL) Seff] effective stress (for nonlinear materials) at the top [kN/m²]*
rc_ssfy_Top = 131 *[(NL) fy] actual yield strength (for nonlinear materials) at the top [kN/m²]*
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/m²]*
rc_ssBzz_Top = 135 *[(NL) Bzz] back stress in local z direction (for nonlinear materials) at the top [kN/m²]*
rc_ssBxy_Top = 136 *[(NL) Bxy] shear back stress (for nonlinear materials) at the top [kN/m²]*
rc_ssSzz_Middle = 137 *[Szz] axial stress in local z direction at the middle [kN/m²]*
rc_ssSeff_Middle = 138 *[(NL) Seff] effective stress (for nonlinear materials) at the middle [kN/m²]*
rc_ssfy_Middle = 139 *[(NL) fy] actual yield strength (for nonlinear materials) at the middle [kN/m²]*
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/m²]*
rc_ssBzz_Middle = 143 *[(NL) Bzz] back stress in local z direction (for nonlinear materials) at the middle [kN/m²]*
rc_ssBxy_Middle = 144 *[(NL) Bxy] shear back stress (for nonlinear materials) at the middle [kN/m²]*
rc_ssSzz_Bottom = 145 *[Szz] axial stress in local z direction at the bottom [kN/m²]*
rc_ssSeff_Bottom = 146 *[(NL) Seff] effective stress (for nonlinear materials) at the bottom [kN/m²]*
rc_ssfy_Bottom = 147 *[(NL) fy] actual yield strength (for nonlinear materials) at the bottom [kN/m²]*

rc_ssU_Bottom = 148 *[(NL) Utilization] utilization (for nonlinear materials) at the bottom [ ]*

rc_ssState_Bottom = 149 *[(NL) State] stress state (for nonlinear materials) at the bottom [ ]*

rc_ssByy_Bottom = 150 *[(NL) Byy] back stress in local y direction (for nonlinear materials) at the bottom [kN/m²]*

rc_ssBzz_Bottom = 151 *[(NL) Bzz] back stress in local z direction (for nonlinear materials) at the bottom [kN/m²]*

rc_ssBxy_Bottom = 152 *[(NL) Bxy] shear back stress (for nonlinear materials) at the bottom [kN/m²]*

ELineForce:

rc_lfNx = 200 *[Nx] normal force [kN]*

rc_lfVy = 201 *[Vy] shear force [kN]*

rc_lfVz = 202 *[Vz] shear force [kN]*

rc_lfTx = 203 *[Tx] twisting moment [kNm]*

rc_lfMy = 204 *[My] bending moment [kNm]*

rc_lfMz = 205 *[Mz] bending moment [kNm]*

rc_lfMyD = 206 *[MyD] design flexural moment about local y axis [kNm]*

rc_lfVxz = 207 *[Vxz] longitudinal shear connection force [kN/m]*

ESurfaceForce:

rc_sfNx = 300 *[nx] cross-section force [kN/m]*

rc_sfNy = 301 *[ny] cross-section force [kN/m]*

rc_sfNxy = 302 *[nxy] cross-section torsional force [kN/m]*

rc_sfMx = 303 *[mx] bending moment [kNm/m]*

rc_sfMy = 304 *[my] bending moment [kNm/m]*

rc_sfMxy = 305 *[mxy] torsional moment [kNm/m]*

rc_sfVxz = 306 *[vxz] shear force [kN/m]*

rc_sfVyz = 307 *[vyz] shear force [kN/m]*

rc_sfvSz = 308 *[vRz] resultant shear force [kN/m]*

rc_sfN1 = 309 *[n1] 1st principal force [kN/m]*

rc_sfN2 = 310 *[n2] 2nd principal force [kN/m]*

rc_sfAn = 311 *[an] principal force direction [°]*

rc_sfM1 = 312 *[m1] 1st principal moment [kNm/m]*

rc_sfM2 = 313 *[m2] 2nd principal moment [kNm/m]*

rc_sfAm = 314 *[am] principal moment direction [°]*

rc_sfNxD = 315 *[nxD] design force [kN/m]*

rc_sfNyD = 316 *[nyD] design force [kN/m]*

rc_sfMxDp = 319 *[mxD+] design moment (plus) [kNm/m]*

rc_sfMxDm = 320 *[mxD-] design moment (minus) [kNm/m]*

rc_sfMyDp = 321 *[myD+] design moment (plus) [kNm/m]*

rc_sfMyDm = 322 *[myD-] design moment (minus) [kNm/m]*

rc_sfAvRz = 323 *[avRz] direction of the resultant shear force [°]*

rc_sfAn1= 324 *[an1] direction of principal force 1 [°]*

rc_sfAn2= 325 *[an2] direction of principal force 2 [°]*

rc_sfAm1= 326 *[am1] direction of principal moment 1 [°]*

rc_sfAm2= 327 *[am2] direction of principal moment 2 [°]*

ENodalSupportForce:

rc_nsfRx = 400 *[Rx] support force in local x direction [kN]*

rc_nsfRy = 401 *[Ry] support force in local y direction [kN]*

rc_nsfRz = 402 *[Rz] support force in local z direction [kN]*

rc_nsfRxx = 403 *[Rxx] support moment about local x axis [kNm]*

rc_nsfRyy = 404 *[Ryy] support moment about local y axis [kNm]*

rc_nsfRzz = 405 *[Rzz] support moment about local z axis [kNm]*

rc_nsfRr = 406 *[Rr] resultant support force [kN]*

rc_nsfRrr = 407 *[Rrr] resultant support moment [kNm]*

rc_nsfRxyz = 408 *[Rxyz] support forces in local directions [kN]*

rc_nsfRxxyyzz = 409 *[Rxxyyzz] support forces about local directions [kN]*

rc_nsfRalpha = 410 *[α_R] ratio of support force component in loc. xy plane to support force in loc. z direction [-]*

ELineSupportForce**:**

rc_lsfRx = 500 *[Rx] support force in local x direction [kN/m]*

rc_lsfRy = 501 *[Ry] support force in local y direction [kN/m]*

rc_lsfRz = 502 *[Rz] support force in local z direction [kN/m]*

rc_lsfRxx = 503 *[Rxx] support moment about local x axis [kNm/m]*

rc_lsfRyy = 504 *[Ryy] support moment about local y axis [kNm/m]*

rc_lsfRzz = 505 *[Rzz] support moment about local z axis [kNm/m]*

rc_lsfRr = 506 *[Rr] resultant support force [kN/m]*

rc_lsfRrr = 507 *[Rrr] resultant support moment [kNm/m]*

ESurfaceSupportForce:

rc_ssfRx = 600 *[Rx] support force in local x direction [kN/m²]*
rc_ssfRy = 601 *[Ry] support force in local y direction [kN/m²]*
rc_ssfRz = 602 *[Rz] support force in local z direction [kN/m²]*

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_gfNx = 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 *[R(zz) 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

	nodal displacements:	***line displacements:***
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 *[axb] Bottom reinforcement in local x direction [m²/m]*
rc_rAsby = 1201 *[ayb] Bottom reinforcement in y direction [m²/m]*
rc_rAstx = 1202 *[axt] Top reinforcement in x direction [m²/m]*
rc_rAsty = 1203 *[ayt] Top reinforcement in y direction [m²/m]*
rc_rAsxbt= 1204 *[axb,axt] Reinforcement in x direction at top and bottom [m²/m]*
rc_rAsybt= 1205 *[ayb,ayt] Reinforcement in y direction at top and bottom [m²/m]*
rc_rAsxyb = 1206 *[axb,ayb] Reinforcement in x and y direction at bottom [m²/m]*
rc_rAsxyt = 1207 *[axt,ayt] Reinforcement in x and y direction at top [m²/m]*

EShearCapacity:

rc_scVRdc = 1300 *[VRd,c] shear resistance without shear reinforcement [kN/m]*
rc_scVEdMinusVRdc = 1301 *[(vEd-VRd,c)] resultant shear force minus shear resistance [kN/m]*
rc_scVRdmax = 1302 *[VRd,max] maximum shear resistance without shear reinforcement [kN/m]*
rc_scVEdDivVRdmax = 1303 *[(vEd/VRd,max)] resultant shear force divided by maximum shear resistance [ ]*
rc_scaVEd = 1304 *[avEd] principal direction for shear [°]*
rc_scAsw = 1305 *[asw] required specific shear reinforcement [m²/m]*

ECrackWidth:

rc_cw_wkb = 1400 *[wk(b)] bottom cracking at reinforcement bar [m]*
rc_cw_wkt = 1401 *[wk(t)] top cracking at reinforcement bar [m]*
rc_cw_wk2b = 1402 *[wk2(b)] bottom cracking at extreme fibre [m]*
rc_cw_wk2t = 1403 *[wk2(t)] top cracking at extreme fibre [m]*
rc_cw_wRb = 1404 *[wR(b)] angle of primary crack relative to the local x direction at the bottom [°]*
rc_cw_wRt = 1405 *[wR(t)] angle of primary crack relative to the local x direction at the top [°]*
rc_cw_wS2b = 1406 *[S2b] stress in the bottom rebar [kN/m²]*
rc_cw_wS2t = 1407 *[S2t] stress in the top rebar [kN/m²]*
rc_cw_wkbt = 4522 *[wk(bt)] bottom and top cracking at reinforcement bar [m]*
rc_cw_wk2bt = 4523 *[wk2(bt)] bottom and top cracking at extreme fibre [m]*

EVelocity:

rc_veX= 1500 *[vX] velocity in x direction [m/s]*
rc_veY = 1501 *[vY] velocity in y direction [m/s]*
rc_veZ = 1502 *[vZ] velocity in z direction [m/s]*
rc_veXX = 1503 *[vXX] angular velocity about the x axis [rad/s]*
rc_veYY = 1504 *[vYY] angular velocity about the y axis [rad/s]*
rc_veZZ = 1505 *[vZZ] angular velocity about the z axis [rad/s]*
rc_veR = 1506 *[vR] resultant velocity [m/s]*
rc_veRR = 1507 *[vRR] resultant angular velocity [rad/s]*

EAcceleration:

rc_acX= 1600 *[aX] acceleration in x direction [m/s²]*
rc_acY = 1601 *[aY] acceleration in y direction [m/s²]*
rc_acZ = 1602 *[aZ] acceleration in z direction [m/s²]*
rc_acXX = 1603 *[aXX] angular acceleration about the x axis [rad/s²]*
rc_acYY = 1604 *[aYY] angular acceleration about the y axis [rad/s²]*
rc_acZZ = 1605 *[aZZ] angular acceleration about the z axis [rad/s²]*
rc_acR = 1606 *[aR] resultant acceleration [m/s²]*
rc_acRR = 1607 *[aRR] resultant angular acceleration [rad/s²]*

Actual reinforcement

rc_arAsxb = 1700 *[xb] actual reinforcement in x direction at bottom [m²/m]*
rc_arAsyb= 1701 *[yb] actual reinforcement in y direction at bottom [m²/m]*
rc_arAsxt= 1702 *[xt] actual reinforcement in x direction at top [m²/m]*
rc_arAsyt= 1703 *[yt] actual reinforcement in y direction at top [m²/m]*

rc_arAsxbt = 1704 *[xb,xt] actual reinforcement in x direction at top and bottom [m²/m]*
rc_arAsybt = 1705 *[yb,yt] actual reinforcement in y direction at top and bottom [m²/m]*

Reinforcement difference

rc_rdAsxb = 1800 *[xb – axb] reinforcement difference (calculated-actual) in x direction at bottom [m²/m]*
rc_rdAsyb = 1801 *[yb – ayb] reinforcement difference (calculated-actual) in y direction at bottom [m²/m]*
rc_rdAsxt = 1802 *[xt – axt] reinforcement difference (calculated-actual) in x direction at top [m²/m]*
rc_rdAsyt = 1803 *[yt – ayt] reinforcement difference (calculated-actual) in y direction at top [m²/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 exzyzR [ ]*
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 [%]*

Ddisplacements 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, *[reinforcement check] reinforcement check [ ]*

Young modulus
rc_ymEx = 2500, *[Ex] calculated modulus of elasticity in local x direction [kN/m²]*
rc_ymEy = 2501, *[Ey] calculated modulus of elasticity in local y direction [kN/m²]*

Mode shape ordinates
rc_vd_eX = 2600, *[eX] translation in X direction [ ]*
rc_vd_eY = 2601, *[eY] translation in Y direction [ ]*
rc_vd_eZ = 2602, *[eZ] translation in Z direction [ ]*
rc_vd_fX = 2603, *[fX] rotation in X direction [ ]*
rc_vd_fY = 2604, *[fY] rotation in Y direction [ ]*
rc_vd_fZ = 2605, *[fZ] rotation in Z direction [ ]*
rc_vd_eR = 2606, *[eR] resultant translation [ ]*
rc_vd_fR = 2607, *[fR] resultant rotation [ ]*

Reinforcement design forces ULS
rc_sfMxD = 2700, *[mxD] mx design reinforcement moment (ULS) [kNm/m]*
rc_sfMyD = 2701, *[myD] my design reinforcement moment (ULS) [kNm/m]*
rc_sfMxU = 2702, *[mxU] mx ultimate reinforcement moment (ULS) [kNm/m]*
rc_sfMyU = 2703, *[myU] my ultimate reinforcement moment (ULS) [kNm/m]*
rc_sfMxDU = 2704, *[mxD,mxU] mx design and ultimate reinforcement moment (ULS) [kNm/m]*
rc_sfMyDU = 2705, *[myD,myU] my design and ultimate reinforcement moment (ULS) [kNm/m]*

Reinforcement design forces SLS
rc_sfMxDcr = 2800, *[mxD(cr)] mx design reinforcement moment (SLS) [kNm/m]*
rc_sfMyDcr = 2801, *[myD(cr)] my design reinforcement moment (SLS) [kNm/m]*
rc_sfMxUcr = 2802, *[mxU(cr)] mx ultimate reinforcement moment (SLS) [kNm/m]*
rc_sfMyUcr = 2803, *[myU(cr)] my ultimate reinforcement moment (SLS) [kNm/m]*
rc_sfMxDUcr = 2804, *[mxD(cr),mxU(cr)] mx design and ultimate reinforcement moment (SLS) [kNm/m]*
rc_sfMyDUcr = 2805, *[myD(cr),myU(cr)] my design and ultimate reinforcement moment (SLS) [kNm/m]*

Arbo/cret utilization
rc_arcrUtil = 2901, *[Utilization] utilization [ ]*
rc_arcrAvgUtil = 2902, *[Average utilization] average utilization [ ]*

Arbo/cret results
rc_arcrVx = 3000, *[vx] shear force in local x direction [kN/m]*
rc_arcrN = 3001, *[n] axial force [kN/m]*
rc_arcrVz = 3002, *[vz] shear force in local z direction [kN/m]*
rc_arcrM = 3003, *[m] torsional moment [kNm/m]*
rc_arcrDez = 3004, *[CRET dez] relative displacement in local z direction [m]*

Beam end release relative displacements
rc_berrdEx = 3100, *[ex] beam end release relative displacement in x direction [m]*
rc_berrdEy = 3101, *[ey] beam end release relative displacement in y direction [m]*
rc_berrdEz = 3102, *[ez] beam end release relative displacement in z direction [m]*
rc_berrdFx = 3103, *[fx] beam end release relative rotation about local x axis [rad]*
rc_berrdFy = 3104, *[fy] beam end release relative rotation about local y axis [rad]*
rc_berrdFz = 3105, *[fz] beam end release relative rotation about local z axis [rad]*
rc_berrdEr = 3106, *[eR] beam end release resultant relative translation [m]*
rc_berrdFr = 3107, *[fR] beam end release resultant relative rotation [rad]*

Beam strain
rc_bstExx = 3200, *[exx] strain exx [ ]*
rc_bstKyy = 3201, *[kyy] curvature kyy [1/m]*
rc_bstKzz = 3202, *[kzz] curvature kzz [1/m]*

rc_bstEyz = 3203, *[eyz] strain eyz [ ]*
rc_bstExy = 3204, *[exy] strain exy [ ]*
rc_bstExz = 3205, *[exz] strain exz [ ]*

Virtual beam internal forces
rc_vbifNx = 3300, *[Nx] axial force [kN]*
rc_vbifVy = 3301, *[Vy] shear force in local y direction [kN]*
rc_vbifVz = 3302, *[Vz] shear force in local z direction [kN]*
rc_vbifTx = 3303, *[Tx] torsional moment [kNm]*
rc_vbifMy = 3304**,** *[My] flexural moment about local y axis [kNm]*
rc_vbifMz = 3305, *[Mz] flexural moment about local z axis [kNm]*

Beam strain at stress point
rc_bstspExxTMin = 3400, *[exx T Min] minimum stress point strain [ ]*
rc_bstspExxTMax = 3401, *[exx T Max] maximum stress point strain [ ]*
rc_bstspExxTMinMax = 3402, *[exx T MinMax] minimum,maximum stress point strain [ ]*
rc_bstspExxEMin = 3403, *[(NLP) exx E Min] minimum elastic stress point strain [ ]*
rc_bstspExxEMax = 3404**,** *[(NLP) exx E Max] maximum elastic stress point strain [ ]*
rc_bstspExxEMinMax = 3405, *[(NLP) exx E MinMax] minimum, maximum elastic stress point strain [ ]*
rc_bstspExxPMin = 3406, *[(NLP) exx P Min] minimum plastic stress point strain [ ]*
rc_bstspExxPMax = 3407, *[(NLP) exx P Max] maximum plastic stress point strain [ ]*
rc_bstspExxPMinMax = 3408, *[(NLP) exx P MinMax] minimum, maximum plastic stress point strain [ ]*
rc_bstspEeffMin = 3409, *[(NLP) eeff Min] minimum effective stress point strain [ ]*
rc_bstspEeffMax = 3410, *[(NLP) eeff Max] maximum effective stress point strain [ ]*
rc_bstspEeffMinMax = 3411, *[(NLP) eeff MinMax] minimum, maximum effective stress point strain [ ]*
rc_bstspDeeffMin = 3412, *[(NLP) deeff Min] minimum effective plastic stress point strain increment [ ]*
rc_bstspDeeffMax = 3413, *[(NLP) deeff Max] maximum effective plastic stress point strain increment [ ]*
rc_bstspDeeffMinMax = 3414, *[(NLP) deeff MinMax] minimum, maximum effective plastic stress point strain increment [ ]*
rc_bstspLeeffMin = 3415, *[(NLE) eeff Min] minimum effective stress point strain [ ]*
rc_bstspLeeffMax = 3416, *[(NLE) eeff Max] maximum effective stress point strain [ ]*
rc_bstspLeeffMinMax = 3417, *[(NLE) eeff MinMax] minimum, maximum effective stress point strain [ ]*

Vibration response factor analysis
rc_vrfaRSelf = 3500, *[R self] vibration response factor (self) [ ]*
rc_vrfaRExtr = 3501, *[R extr.] vibration response factor (extr.) [ ]*
rc_vrfaRFull = 3502, *[R full] vibration response factor (full) [ ]*

Virtual beam displacements
rc_vbdEx = 3600, *[eX] translation in x direction [m]*
rc_vbdEy = 3601, *[eY] translation in y direction [m]*
rc_vbdEz = 3602, *[eZ] translation in z direction [m]*
rc_vbdFx = 3603, *[fX] rotation in x direction [rad]*
rc_vbdFy = 3604, *[fY] rotation in y direction [rad]*
rc_vbdFz = 3605, *[fZ] rotation in z direction [rad]*
rc_vbdEr = 3606, *[eR] resultant translation [m]*
rc_vbdFr = 3607, *[fR] resultant rotation [rad]*

Surface strain at stress point
rc_sstspExxTT = 3700, *[exx T T] strain exx T at the top [ ]*
rc_sstspEyyTT = 3701, *[eyy T T] strain eyy T at the top [ ]*
rc_sstspExyTT = 3702, *[exy T T] strain exy T at the top [ ]*
rc_sstspE1TT = 3703, *[e1 T T] principal strain 1 at the top [ ]*
rc_sstspE2TT = 3704, *[e2 T T] principal strain 2 at the top [ ]*
rc_sstspAeTT = 3705, *[ae T T] principal strain direction at the top [°]*
rc_sstspExxET = 3706, *[(NLP) exx E T] elastic strain exx at the top [ ]*

rc_sstspEyyET = 3707, *[(NLP) eyy E T] elastic strain eyy at the top [ ]*
rc_sstspExyET = 3708, *[(NLP) exy E T] elastic strain exy at the top [ ]*
rc_sstspE1ET = 3709, *[(NLP) e1 E T] elastic principal strain 1 at the top [ ]*
rc_sstspE2ET = 3710, *[(NLP) e2 E T] elastic principal strain 2 at the top [ ]*
rc_sstspAeET = 3711, *[(NLP) ae E T] elastic principal strain direction at the top [°]*
rc_sstspExxPT = 3712, *[(NLP) exx P T] plastic strain exx at the top [ ]*
rc_sstspEyyPT = 3713, *[(NLP) eyy P T] plastic strain eyy at the top [ ]*
rc_sstspExyPT = 3714, *[(NLP) exy P T] plastic strain exy at the top [ ]*
rc_sstspE1PT = 3715, *[(NLP) e1 P T] plastic principal strain 1 at the top [ ]*
rc_sstspE2PT = 3716, *[(NLP) e2 P T] plastic principal strain 2 at the top [ ]*
rc_sstspAePT = 3717, *[(NLP) ae P T] plastic principal strain direction at the top [°]*
rc_sstspEeffPT = 3718, *[(NLP) eeff P T] effective plastic strain at the top [ ]*
rc_sstspDeeffPT = 3719, *[(NLP) deeff P T] effective plastic strain increment at the top [ ]*
rc_sstspExxTC = 3720, *[exx T C] strain exx T at the center [ ]*
rc_sstspEyyTC = 3721, *[eyy T C] strain eyy T at the center [ ]*
rc_sstspExyTC = 3722, *[exy T C] strain exy T at the center [ ]*
rc_sstspE1TC = 3723, *[e1 T C] principal strain 1 at the center [ ]*
rc_sstspE2TC = 3724, *[e2 T C] principal strain 2 at the center [ ]*
rc_sstspAeTC = 3725, *[ae T C] principal strain direction at the center [°]*
rc_sstspExxEC = 3726, *[(NLP) exx E C] elastic strain exx at the center [ ]*
rc_sstspEyyEC = 3727, *[(NLP) eyy E C] elastic strain eyy at the center [ ]*
rc_sstspExyEC = 3728, *[(NLP) exy E C] elastic strain exy at the center [ ]*
rc_sstspE1EC = 3729, *[(NLP) e1 E C] elastic principal strain 1 at the center [ ]*
rc_sstspE2EC = 3730, *[(NLP) e2 E C] elastic principal strain 2 at the center [ ]*
rc_sstspAeEC = 3731, *[(NLP) ae E C] elastic principal strain direction at the center [°]*
rc_sstspExxPC = 3732, *[(NLP) exx P C] plastic strain exx at the center [ ]*
rc_sstspEyyPC = 3733, *[(NLP) eyy P C] plastic strain eyy at the center [ ]*
rc_sstspExyPC = 3734, *[(NLP) exy P C] plastic strain exy at the center [ ]*
rc_sstspE1PC = 3735, *[(NLP) e1 P C] plastic principal strain 1 at the center [ ]*
rc_sstspE2PC = 3736, *[(NLP) e2 P C] plastic principal strain 2 at the center [ ]*
rc_sstspAePC = 3737, *[(NLP) ae P C] plastic principal strain direction at the center [°]*
rc_sstspEeffPC = 3738, *[(NLP) eeff P C] effective plastic strain at the center [ ]*
rc_sstspDeeffPC = 3739, *[(NLP) deeff P C] effective plastic strain increment at the center [ ]*
rc_sstspExxTB = 3740, *[exx T B] strain exx T at the bottom [ ]*
rc_sstspEyyTB = 3741, *[eyy T B] strain eyy T at the bottom [ ]*
rc_sstspExyTB = 3742, *[exy T B] strain exy T at the bottom [ ]*
rc_sstspE1TB = 3743, *[e1 T B] principal strain 1 at the bottom [ ]*
rc_sstspE2TB = 3744, *[e2 T B] principal strain 2 at the bottom [ ]*
rc_sstspAeTB = 3745, *[ae T B] principal strain direction at the bottom [°]*
rc_sstspExxEB = 3746, *[(NLP) exx E B] elastic strain exx at the bottom [ ]*
rc_sstspEyyEB = 3747, *[(NLP) eyy E B] elastic strain eyy at the bottom [ ]*
rc_sstspExyEB = 3748, *[(NLP) exy E B] elastic strain exy at the bottom [ ]*
rc_sstspE1EB = 3749, *[(NLP) e1 E B] elastic principal strain 1 at the bottom [ ]*
rc_sstspE2EB = 3750, *[(NLP) e2 E B] elastic principal strain 2 at the bottom [ ]*
rc_sstspAeEB = 3751, *[(NLP) ae E B] elastic principal strain direction at the bottom [°]*
rc_sstspExxPB = 3752, *[(NLP) exx P B] plastic strain exx at the bottom [ ]*
rc_sstspEyyPB = 3753, *[(NLP) eyy P B] plastic strain eyy at the bottom [ ]*
rc_sstspExyPB = 3754, *[(NLP) exy P B] plastic strain exy at the bottom [ ]*
rc_sstspE1PB = 3755, *[(NLP) e1 P B] plastic principal strain 1 at the bottom [ ]*
rc_sstspE2PB = 3756, *[(NLP) e2 P B] plastic principal strain 2 at the bottom [ ]*
rc_sstspAePB = 3757, *[(NLP) ae P B] plastic principal strain direction at the bottom [°]*
rc_sstspEeffPB = 3758, *[(NLP) eeff P B] effective plastic strain at the bottom [ ]*
rc_sstspDeeffPB = 3759, *[(NLP) deeff P B] effective plastic strain increment at the bottom [ ]*
rc_sstspLeeffT = 3760, *[(NLE) eeff T] effective strain at the top [ ]*
rc_sstspLeeffC = 3761, *[(NLE) eeff C] effective strain at the center [ ]*
rc_sstspLeeffB = 3762, *[(NLE) eeff B] effective strain at the bottom [ ]*

Vertical deflection

rc_vdW1 = 3800, *[w1] w1 [m]*
rc_vdW2 = 3801, *[w2] w2 [m]*
rc_vdW3 = 3802, *[w3] w3 [m]*
rc_vdWTot = 3803, *[wtot] wtot [m]*
rc_vdWbij = 3804, *[wbij] wbij [m]*

Design results of RC cores and walls

rc_rccwUBeam = 3900, *[Utilization beam] utilization beam [ ]*
rc_rccwUStrip = 3901, *[Utilization strip] utilization strip [ ]*
rc_rccwUOverall = 3902, *[Overall utilization] overall utilization [ ]*

Spring deformations

rc_rsdEx = 4000, *[ex] spring deformation in local x direction [m]*
rc_rsdEy = 4001, *[ey] spring deformation in local y direction [m]*
rc_rsdEz = 4002, *[ez] spring deformation in local z direction [m]*
rc_rsdFx = 4003, *[fx] spring rotational deformation about the local x axis [rad]*
rc_rsdFy = 4004, *[fy] spring rotational deformation about the local y axis [rad]*
rc_rsdFz = 4005, *[fz] spring rotational deformation about the local z axis [rad]*

Spring nonlinear results

rc_snlFyx = 4100, *[(NL) Fy,x] spring actual limit force in local x direction [kN]*
rc_snlUx = 4101, *[(NL) Util.,x] spring utilization in local x direction [ ]*
rc_snlEEx = 4102, *[(NLP) e,x E] spring elastic deformation in local x direction [m]*
rc_snlPEx = 4103, *[(NLP) e,x P] spring plastic deformation in local x direction [m]*
rc_snlPEeffx = 4104, *[(NLP) eeff,x P] spring effective plastic deformation in local x direction [m]*
rc_snlPdeeffx = 4105, *[(NLP) deeff,x P] spring effective plastic deformation increment in local x direction [m]*
rc_snlBx = 4106, *[(NLP) B,x] back stress in local x direction [kN]*
rc_snlFyy = 4107, *[(NL) Fy,y] spring actual limit force in local y direction [kN]*
rc_snlUy = 4108, *[(NL) Util.,y] spring utilization in local y direction [ ]*
rc_snlEEy = 4109, *[(NLP) e,y E] spring elastic deformation in local y direction [m]*
rc_snlPEy = 4110, *[(NLP) e,y P] spring plastic deformation in local y direction [m]*
rc_snlPEeffy = 4111, *[(NLP) eeff,y P] spring effective plastic deformation in local y direction [m]*
rc_snlPdeeffy = 4112, *[(NLP) deeff,y P] spring effective plastic deformation increment in local y direction [m]*
rc_snlBy = 4113, *[(NLP) B,y] back stress in local y direction [kN]*
rc_snlFyz = 4114, *[(NL) Fy,z] spring actual limit force in local z direction [kN]*
rc_snlUz = 4115, *[(NL) Util.,z] spring utilization in local z direction [ ]*
rc_snlEEz = 4116, *[(NLP) e,z E] spring elastic deformation in local z direction [m]*
rc_snlPEz = 4117, *[(NLP) e,z P] spring plastic deformation in local z direction [m]*
rc_snlPEeffz = 4118, *[(NLP) eeff,z P] spring effective plastic deformation in local z direction [m]*
rc_snlPdeeffz = 4119, *[(NLP) deeff,z P] spring effective plastic deformation increment in local z direction [m]*
rc_snlBz = 4120, *[(NLP) B,z] back stress in local z direction [kN]*
rc_snlMyxx = 4121, *[(NL) My,xx] spring actual limit moment about the local x axis [kNm]*
rc_snlUxx = 4122, *[(NL) Util.,xx] spring rotational utilization about the local x axis [ ]*
rc_snlEExx = 4123, *[(NLP) e,xx E] spring elastic rotational deformation about the local x axis [rad]*
rc_snlPExx = 4124, *[(NLP) e,xx P] spring plastic rotational deformation about the local x axis [rad]*
rc_snlPEeffxx = 4125, *[(NLP) eeff,xx P] spring effective plastic rotational deformation increment about the local x axis [rad]*
rc_snlPdeeffxx = 4126, *[(NLP) deeff,xx P] spring effective plastic rotational deformation about the local x axis [rad]*
rc_snlBxx = 4127, *[(NLP) B,xx] back stress about the local x axis [kNm]*
rc_snlMyyy = 4128, *[(NL) My,yy] spring actual limit moment about the local y axis [kNm]*
rc_snlUyy = 4129, *[(NL) Util.,yy] spring rotational utilization about the local y axis [ ]*
rc_snlEEyy = 4130, *[(NLP) e,yy E] spring elastic rotational deformation about the local y axis [rad]*
rc_snlPEyy = 4131, *[(NLP) e,yy P] spring plastic rotational deformation about the local y axis [rad]*
rc_snlPEeffyy = 4132, *[(NLP) eeff,yy P] spring effective plastic rotational deformation increment about the local y axis [rad]*

rc_snlPdeeffyy = 4133, *[(NLP) deeff,yy P] spring effective plastic rotational deformation about the local y axis [rad]*
rc_snlByy = 4134, *[(NLP) B,yy] back stress about the local y axis [kNm]*
rc_snlMyzz = 4135, *[(NL) My,zz] spring actual limit moment about the local z axis [kNm]*
rc_snlUzz = 4136, *[(NL) Util.,zz] spring rotational utilization about the local z axis [ ]*
rc_snlEEzz = 4137, *[(NLP) e,zz E] spring elastic rotational deformation about the local z axis [rad]*
rc_snlPEzz = 4138, *[(NLP) e,zz P] spring plastic rotational deformation about the local z axis [rad]*
rc_snlPEeffzz = 4139, *[(NLP) eeff,zz P] spring effective plastic rotational deformation increment about the local z axis [rad]*
rc_snlPdeeffzz = 4140, *[(NLP) deeff,zz P] spring effective plastic rotational deformation about the local z axis [rad]*
rc_snlBzz = 4141, *[(NLP) B,zz] back stress about the local z axis [kNm]*

XLAM stresses
rc_xsSxxmT = 4200, *[Sxx m T] normal stress from bending in local x direction at the top [kN/m²]*
rc_xsSyymT = 4201, *[Syy m T] normal stress from bending in local y direction at the top [kN/m²]*
rc_xsSxymT = 4202, *[Sxy m T] Sxy m at the top [kN/m²]*
rc_xsSxxmB = 4203, *[Sxx m B] normal stress from bending in local x direction at the bottom [kN/m²]*
rc_xsSyymB = 4204, *[Syy m B] normal stress from bending in local y direction at the bottom [kN/m²]*
rc_xsSxymB = 4205, *[Sxy m B] Sxy m at the bottom [kN/m²]*
rc_xsSxxn = 4206, *[Sxx n] normal stress from membrane forces in local x direction [kN/m²]*
rc_xsSyyn = 4207, *[Syy n] normal stress from membrane forces in local y direction [kN/m²]*
rc_xsSxyn = 4208, *[Sxy n] Sxy n [kN/m²]*
rc_xsSxzmax = 4209, *[Sxz max] maximum shear stress perpendicular to local x direction [kN/m²]*
rc_xsSyzmax = 4210, *[Syz max] maximum shear stress perpendicular to local y direction [kN/m²]*
rc_xsSrxmax = 4211, *[Srx max] relevant rolling shear stress in a direction with x normal [kN/m²]*
rc_xsSrymax = 4212, *[Sry max] relevant rolling shear stress in a direction with y normal [kN/m²]*

XLAM utilization
rc_xuMN0 = 4300, *[M-N-0 utilization] maximum utilization from bending moments and normal force parallel to grain direction [ ]*
rc_xuMN90 = 4301, *[M-N-90 utilization] maximum utilization from bending moments and normal force perpendicular to grain direction [ ]*
rc_xuVt = 4302, *[V-T utilization] maximum utilization from shear and torsion [ ]*
rc_xuVrN = 4303, *[Vr-N utilization] maximum utilization from rolling shear and normal force [ ]*
rc_xuMax = 4304, *[Maximum utilization] maximum utilization [ ]*

Design check of masonry wall
rc_dcmwU = 4400, *[Utilization] utilization [ ]*

Spring dynamic results
rc_sdrVx = 4500, *[(D) v,x] spring deformation rate in local x direction [m/s²]*
rc_sdrRDx = 4501, *[(D) R,x D] damper force in local x direction [kN]*
rc_sdrRSx = 4502, *[(D) R,x S] total spring-damper force in local x direction [kN]*
rc_sdrVy = 4503, *[(D) v,y] spring deformation rate in local y direction [m/s²]*
rc_sdrRDy = 4504, *[(D) R,y D] damper force in local y direction [kN]*
rc_sdrRSy = 4505, *[(D) R,y S] total spring-damper force in local y direction [kN]*
rc_sdrVz = 4506, *[(D) v,z] spring deformation rate in local z direction [m/s²]*
rc_sdrRDz = 4507, *[(D) R,z D] damper force in local z direction [kN]*

rc_sdrRSz = 4508, *[(D) R,z S] total spring-damper force in local z direction [kN]*
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]*

Design results of RC columns
rc_rccolUNM = 4518, *[Utilization] utilization N-M (axial force + bending moment) [ ]*
rc_rccolUVT = 4519, *[Utilization] utilization V-T (shear + torsion) [ ]*
rc_rccolAsl= 4520, *[Asl] required cross-sectional area of the longitudinal reinforcement for torsion [m^2]*
rc_rccolUSum = 4521, *[Max. utilization] Maximum of utilization N-M and utilization V-T [ ]*

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 *[Utilization ULS] utilization in ultimate limit state (ULS) [ ]*
rc_sd_util_SLS = 10091 *[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 *not used*
rc_sd_22 = 10022 *not used*
rc_sd_23 = 10023 *not used*
rc_sd_24 = 10024 *not used*
rc_sd_25 = 10025 *not used*
rc_sd_26 = 10026 *not used*
rc_sd_27 = 10027 *not used*
rc_sd_28 = 10028 *not used*
rc_sd_29 = 10029 *not used*
rc_sd_30 = 10030 *not used*
rc_sd_31 = 10031 *not used*
rc_sd_util_ULS = 10090 *not used*
rc_sd_util_SLS = 10091 *not used*

Design code : MSZ
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] shear y [ ]*
rc_sd_5 = 10005 *[Vz] shear z [ ]*
rc_sd_6 = 10006 *[Sr] stress resultant in the web at the flange [ ]*
rc_sd_7 = 10007 *[Bkl-w] web buckling [ ]*
rc_sd_8 = 10008 *[Bkl-f] flange buckling [ ]*
rc_sd_9 = 10009 *[NH] axial resistance [kN]*
rc_sd_10 = 10010 *[VyH] shear resistance y [kN]*
rc_sd_11 = 10011 *[VzH] shear resistance z [kN]*
rc_sd_12 = 10012 *[MyH] moment resistance yy [kNm]*
rc_sd_13 = 10013 *[MzH] moment resistance zz [kNm]*
rc_sd_14 = 10014 *[Nkih] buckling resistance [kN]*
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 *not used*
rc_sd_22 = 10022 *not used*
rc_sd_23 = 10023 *not used*
rc_sd_24 = 10024 *not used*
rc_sd_25 = 10025 *not used*
rc_sd_26 = 10026 *not used*
rc_sd_27 = 10027 *not used*
rc_sd_28 = 10028 *not used*
rc_sd_29 = 10029 *not used*
rc_sd_30 = 10030 *not used*
rc_sd_31 = 10031 *not used*
rc_sd_util_ULS = 10090 *not used*
rc_sd_util_SLS = 10091 *not used*

Design code : STAS
rc_sd_1 = 10001 *[N-M-Strength] axial force - bending (strength) [ ]*
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 *[Sr] equivalent stress in the web at the flange [ ]*
rc_sd_6 = 10006 *[Shy] cross-section shear factor y [ ]*
rc_sd_7 = 10007 *[Shz] cross-section shear factor z [ ]*
rc_sd_8 = 10008 *[Wely] cross-section elasticity modulus y [m^3]*
rc_sd_9 = 10009 *[Welz] cross-section elasticity modulus z [m^3]*
rc_sd_10 = 10010 *[Lmax] maximum slenderness [ ]*
rc_sd_11 = 10011 *[Fimin] minimum buckling reduction factor [ ]*
rc_sd_12 = 10012 *[FiG] lateral buckling reduction factor [ ]*
rc_sd_13 = 10013 *not used]*
rc_sd_14 = 10014 *not used*
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 *not used*
rc_sd_22 = 10022 *not used*
rc_sd_23 = 10023 *not used*
rc_sd_24 = 10024 *not used*
rc_sd_25 = 10025 *not used*
rc_sd_26 = 10026 *not used*
rc_sd_27 = 10027 *not used*
rc_sd_28 = 10028 *not used*
rc_sd_29 = 10029 *not used*
rc_sd_30 = 10030 *not used*
rc_sd_31 = 10031 *not used*
rc_sd_util_ULS = 10090 *not used*
rc_sd_util_SLS = 10091 *not used*

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
rc_td_1 = 10101 244 *[N-M] a*xial force and bending [ ]
rc_td_2 = 10102 245 [N-*M-Buckl] c*ompression, bending, and flexural buckling [ ]
rc_td_3 = 10103 246 *[N-M-LTBuckl] a*xial force, bending, and lateral-torsional buckling [ ]
rc_td_4 = 10104 247 *[Vy-Vz-Tx]* shear in y and z direction and torsion [ ]
rc_td_5 = 10105 248 *[My-Vz] apex zone tensile stress perpendicular to the axis [ ]*
rc_td_6 = 10106 249 *[LambdaRely] relative slenderness ratio corresponding to bending about y axis [ ]*
rc_td_7 = 10107 250 *[LambdaRelz] relative slenderness ratio corresponding to bending about z axis [ ]*
rc_td_8 = 10108 251 *[LambdaRelm] relative slenderness for bending [ ]*
rc_td_9 = 10109 252 *[kcy] design compression strength reducing factor due to axial instability [ ]*
rc_td_10 = 10110 253 *[kcz] design compression strength reducing factor due to axial instability [ ]*
rc_td_11 = 10111 254 *[kcrit] design bending strength reducing factor due to lateral torsional instability [ ])*
rc_td_12 = 10112 255 *[kmod] design resistance modification factor for duration of load and moisture content [ ]*
rc_td_13 = 10113 256 *[st90d] tension perpendicular to the grain [kN/m^2]*
rc_td_14 = 10114 *not used*
rc_td_15 = 10115 *not used*
rc_td_util_ULS = 10190 267 *[Utilization ULS] utilization in ultimate limit state (ULS) [ ]*
rc_td_util_SLS = 10191 259 *[Utilization SLS] utilization in serviceability limit state (SLS) [ ]*

}

Error codes

enum **ESectionsError** = {

seSegmentDefinitionError = –100001	Segment definition error
seSectionTypeIsNotSegment= –100002	Sec tion type is not a segment
seLoadCaseIndexOutOfBounds= –100003	Load case index invalid
seLoadCombinationIndexOutOfBounds= –100004	Load combination index invalid
seInvalidAnalysisType= –100005 }	Analysis type invalid

Records / structures

RNodeCrackWidthValues = (

RCrackWidthValues	**covBottom**	*Crack width values at bottom*
RCrackWidthValues	**covTop**	*Crack width values at top*

)

RNodeReinforcementValues = (

double	**Asbx**	*Reinforcement area in x direction at bottom*
double	**Asby**	*Reinforcement area in y direction at bottom*
double	**Astx**	*Reinforcement area in x direction at top*
double	**Asty**	*Reinforcement area in y direction at top*

)

RResultBlock = (

EAnalysisType	**AnalysisType**	*type of analysis*
ECombinationType	**CombinationType**	*combination type*
long	**EnvelopeUID**	*unique envelope index*
long	**LoadCaseId**	*load case index (0 < LoadCaseId ≤ AxisVMLoadCases.Count)*
long	**LoadCombinationId**	*load combination index (0 < LoadCombinationId ≤ AxisVMLoadCombinations.Count)*
long	**LoadLevel**	*load level (increment) index*
EMinMaxType	**MinMaxType**	*minimum or maximum value*
EResultType	**ResultType**	*type of the result*

)

RSection = (

ESectionType	**SectionType**	*type of the section*
BSTR	**Name**	*name of the section*
~~ELongBoolean~~	~~**Visible**~~	*Supesceded in COM 9.3, use functions GetVisibleSectionIDs and SetVisibleSectionIDs in IAxisVMWindows*
RPoint3d	**P**	*start point (point in plane) of the section*
RPoint3d	**N**	*normal vector of the sections plane*
RPoint3d	**SegmentEndP**	*end point of the section's segment (only if SectionType=stSegment)*
ELongBoolean	**InAllResultBlocks**	*If true then visible for all loadcases, combinations and envelopes*
RResultBlock	**ResultBlock**	*Result block (valid only if InAllResultBlocks = lbFalse)*
ELongBoolean	**ForAllResultComponents**	*If true then visible for all types of results*
EResultComponent	**ResultComponent**	*component block (valid only if ForAllResultComponents = lbFalse)*
ESectionDisplayMode	**DisplayMode**	*display mode of the section*
double	**L**	*segment width on the left hand side of the section line (valid only if DisplayMode = sdmDiagramSegWidth) [m]*
double	**R**	*segment width on the right hand side of the section line (valid only if DisplayMode = sdmDiagramSegWidth) [m]*
ELongBoolean	**DiagOnBothSide**	*if true then shows diagrams on both sides of the strip (see sections in AxisVM manual) (valid only if DisplayMode = sdmDiagramSegWidth)*
ELongBoolean	**DiagInPlane**	*show diagram in plane of the element? (valid only if DisplayMode <> sdmResultant)*

)

RSectionElementData = (

long	**SurfaceIndex**	*Index of section-crossed surface*
long	**ContourLineID1**	*first contour line ID of section-crossed surface*
double	**ContourLineRatio1**	*first ratio of cross- section point of contour line and section to surface's vertex point*
long	**ContourLineID2**	*second contour line ID of section-crossed surfcae*
double	**ContourLineRatio2**	*second ratio of cross- section point of contour line and section to surface's vertex point*

)

RSectionSegmentIntegratedResultant = (

double	**N**	*normal force [kN]*
double	**V**	*shear force [kN]*
double	**M**	*bending moment [kNm]*

)

RSectionSegmentChainIntegratedParameters = (

RPoint3d	**StartPoint**	*first point's coordinates of section's integrated resultant chain*
RPoint3d	**EndPoint**	*end point's coordinates of section's integrated resultant chain*
double	**CentreRatio**	*centre ratio*

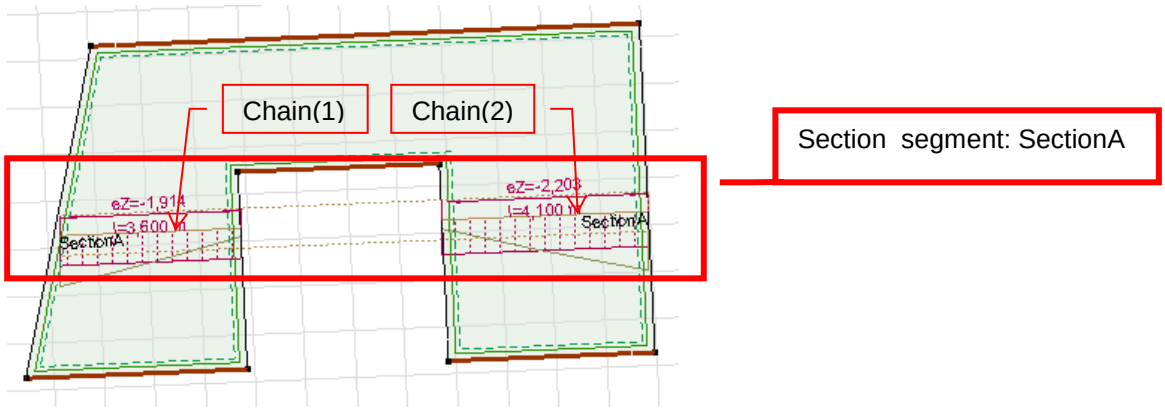
)

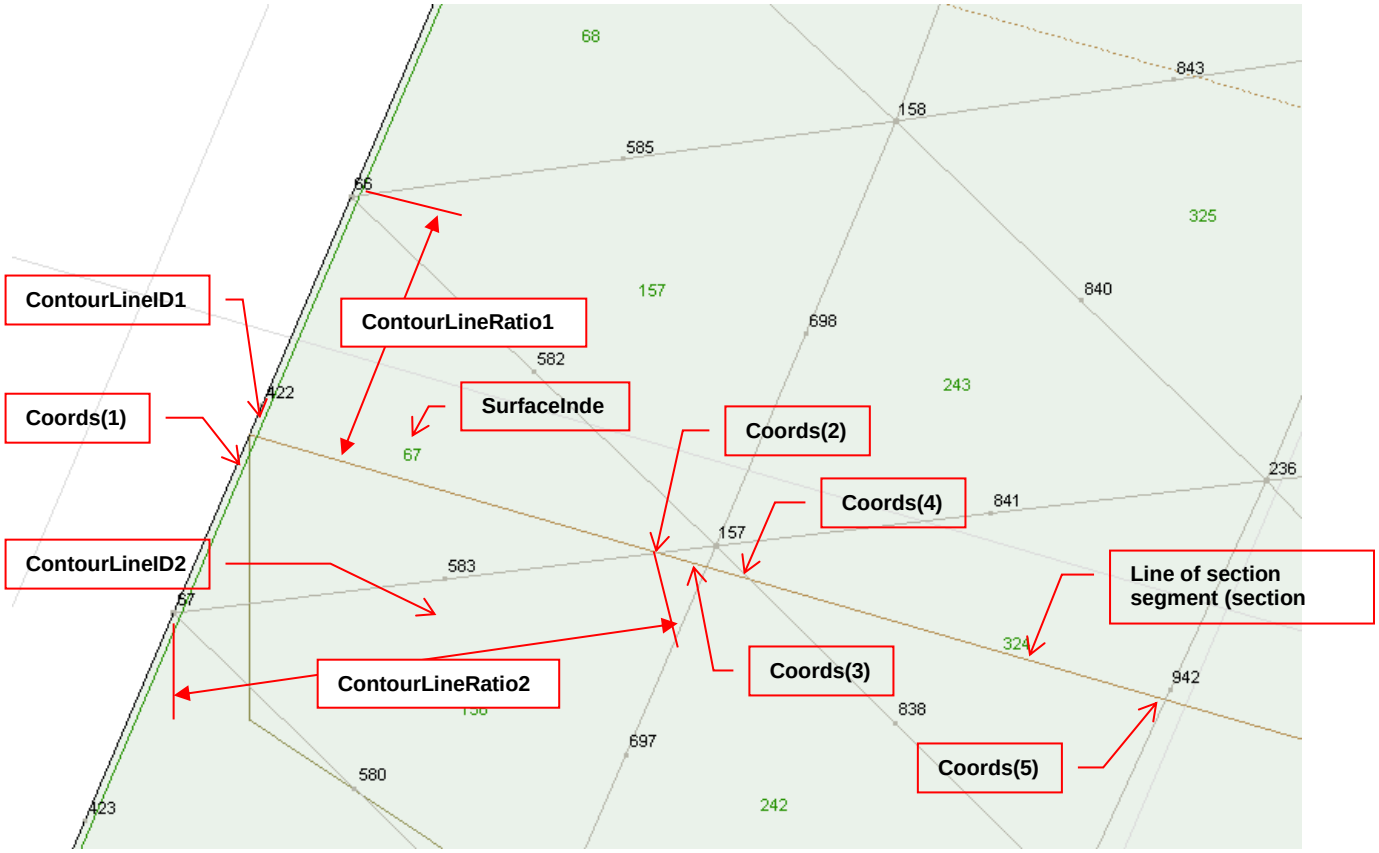
RSurfaceStressValuesTMB = (

RSurfaceStressValues	**ssvTop**	*Surface stress values at top*
RSurfaceStressValues	**ssvMiddle**	*Surface stress values in middle*
RSurfaceStressValues	**ssvBottom**	*Surface stress values at bottom*

)

Functions

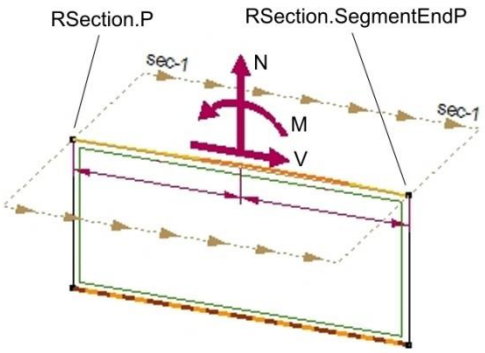




Section segment labelling

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 ≤ Index ≤ 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 ≤ ChainIndex ≤ 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 ≤ IntegratedResultantChainIndex ≤ 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 ≤ IntegratedResultantChainIndex ≤ 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 **GetSegmentChainDisplacementsByLoadCaseId** ([in] long **Index,** [in] long **ChainIndex,**
[in] long **LoadCaseId,** [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*
LoadCaseId	*load case index (0 < LoadCaseId ≤ 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 ≤ AxisVMLoadCombnations.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 **GetSegmentChainIntegratedResultantByLoadCaseId** ([in] long **Index,** [in] long **ChainIndex,**
[in] long **LoadCaseId,** [in] long **LoadLevelOrModeShapeOrTimeStep,**
[in] EAnalysisType **AnalysisType,** [i/o] RSectionSegmentIntegratedResultant **IntegratedResultant**)

Index	*section index*
ChainIndex	*section chain index*
LoadCaseId	*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 LoadCaseId. 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	*section index*
ChainIndex	*section chain index*
MinMaxType	*Minimum or maximum values*
AnalysisType	*Type of Analysis*
WithReinforcement	*True if actual reinforcement is considered (this setting is valid only if AnalysisType = atNonLinearStatic)*
Component	*Displacement type*
Coords	*section chain's coordinates*
Displacements	*section chain's displacements*
Combinations	*Array of strings with name of combinations.*

Get section chain's displacements. If successful, returns number of displacement values, otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBounds, seSectionTypeIsNotSegment).

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	*section index*
ChainIndex	*section chain index*
MinMaxType	*Minimum or maximum values*
CombinationType	*Combination type*
AnalysisType	*Type of Analysis*
WithReinforcement	*True if actual reinforcement is considered (this setting is valid only if AnalysisType = atNonLinearStatic)*
Component	*Displacement type*
Coords	*section chain's coordinates*
Displacements	*section chain's displacements*
Combinations	*Array of strings with name of combinations.*

Get section chain's displacements. If successful, returns number of displacement values, otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBounds, seSectionTypeIsNotSegment).

long **GetCriticalSegmentChainIntegratedResultant** ([in] long **Index,** [in] long **ChainIndex,** [in] ECombinationType **CombinationType**, [in] EMinMaxType **MinMaxType**, [in] EAnalysisType **AnalysisType,** [in] ESectionSegmentChainIntegratedResultant **SectionSegmentChainIntegratedResultant,** [i/o] RSectionSegmentIntegratedResultant **IntegratedResultant**, [out] ECombinationType **CriticalCombinationType,** [out] SAFEARRAY (double)* **Factors,** [out] SAFEARRAY (long)* **LoadCaseIds**)

Index	*section index*
ChainIndex	*section chain index*
CombinationType	*Combination type*
MinMaxType	*Minimum or maximum values*
AnalysisType	*Type of Analysis*
SectionSegmentChainIntegratedResultant	*governing integrated resultants for envelope: N, V and M, respectively*
IntegratedResultant	*integrated resultants: N, V and M, respectively*
CriticalCombinationType	*critical combination type*
Factors	*factors related to critical combination*
LoadCaseIds	*critical load case index*

Get section chain's critical integrated resultants. If successful, returns the section's index otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBounds, seSectionTypeIsNotSegment).

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	*section index*
ChainIndex	*section chain index*
EnvelopeUID	*unique envelope index*
MinMaxType	*Minimum or maximum values*
AnalysisType	*Type of Analysis*
SectionSegmentChainIntegratedResultant	*governing integrated resultants for envelope: N, V and M, respectively*
IntegratedResultant	*integrated resultants: N, V and M, respectively*
LoadCaseOrCombinationId	*load combination index*
LoadLevelOrModeShapeOrTimeStep	*load level (increment)*

Get section chain's integrated resultants envelope. If successful, returns the section's index ,otherwise returns an error code(errDatabaseNotReady, errIndexOutOfBounds, seSectionTypeIsNotSegment).

---

long **GetSegmentChainVelocitiesByLoadCaseId** ([in] long **Index,** [in] long **ChainIndex,**
[in] long **LoadCaseId,** [in] long **TimeStep,** [in] EAnalysisType **AnalysisType,**
[out] SAFEARRAY(RPoint3D)* **Coords,** [out] SAFEARRAY(RVelocityValues)* **VelocityValues**,
[out] SAFEARRAY (BSTR)* **Combinations**)

Index	*section index*
ChainIndex	*section chain index*
LoadCaseId	*load case index* *(0 < LoadCaseId ≤ AxisVMLoadCases.Count)*
TimeStep	*Time step*
AnalysisType	*Type of Analysis*
Coords	*section chain's coordinates*
VelocityValues	*section chain's velocity values*
Combinations	*Array of string with name of load cases.*

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	*section index*
ChainIndex	*section chain index*
MinMaxType	*minimum or maximum values*
AnalysisType	*Type of Analysis*
Component	*Velocity type*
Coords	*section chain's coordinates*
VelocityValues	*section chain's velocity values*
Combinations	*Array of string with name of load cases.*

Get section chain's velocity values. If successful, returns number of velocity values , otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBounds, seSectionTypeIsNotSegment).

---

long **GetSegmentChainAccelerationsByLoadCaseId** ([in] long **Index,** [in] long **ChainIndex,** [in] long **LoadCaseId,** [in] long **TimeStep,** [in] EAnalysisType **AnalysisType,** [out] SAFEARRAY(RPoint3D)* **Coords,** [out] SAFEARRAY(RAccelerationValues)* **AccelerationValues**, [out] SAFEARRAY (BSTR)* **Combinations**)

Index	*section index*
ChainIndex	*section chain index*
LoadCaseId	*load case index* *(0 < LoadCaseId ≤ AxisVMLoadCases.Count)*
TimeStep	*Time step*
AnalysisType	*Type of Analysis*
Coords	*section chain's coordinates*
AccelerationValues	*section chain's acceleration values*
Combinations	*Array of string with name of load cases.*

Get section chain's acceleration values.

If successful, returns number of acceleration values , otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBounds, seSectionTypeIsNotSegment).

---

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	*section index*
ChainIndex	*section chain index*
MinMaxType	*minimum or maximum values*
AnalysisType	*Type of Analysis*
Component	*acceleration type*
Coords	*section chain's coordinates*
AccelerationValues	*section chain's acceleration values*
Combinations	*Array of string with name of load cases.*

Get section chain's acceleration values

If successful, returns number of acceleration values, otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBounds, seSectionTypeIsNotSegment).

---

long **GetSegmentChainSurfaceForcesByLoadCaseId** ([in] long **Index,** [in] long **ChainIndex,** [in] long **LoadCaseId,** [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*
LoadCaseId	*load case index* *(0 < LoadCaseId ≤ AxisVMLoadCases.Count)*
LoadLevelOrModeShapeOrTimeStep	*load level (increment) index*
AnalysisType	*Type of Analysis*
Coords	*section chain's coordinates*
Forces	*section chain's forces*
Combinations	*array of string with name of load cases.*

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 **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 ≤ AxisVMLoadCombnations.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* *(0 < LoadCaseId ≤ AxisVMLoadCases.Count)*
LoadLevel	*load level (increment) index*
AnalysisType	*Type of Analysis*
Coords	*section chain's coordinates*
Reinforcements	*section chain's reinforcement values*
Combinations	*array of string with name of load cases.*

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 **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* *(0 < LoadCombinationId ≤ AxisVMLoadCombnations.Count)*
LoadLevel	*load level (increment) index*
AnalysisType	*Type of Analysis*
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 **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 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 **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 ≤ 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 ≤ AxisVMLoadCombnations.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 Analysis*
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, seSectionTypeIsNotSegment).

---

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 Analysis*
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, seSectionTypeIsNotSegment).

---

long **GetSegmentChainCrackWidthsByLoadCaseId** ([in] long **Index,** [in] long **ChainIndex,** [in] long **LoadCaseId,** [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*
LoadCaseId	*load case index* *(0 < LoadCaseId ≤ AxisVMLoadCases.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 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, seSectionTypeIsNotSegment).

---

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 ≤ AxisVMLoadCombnations.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 (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 ≤ 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 ≤ AxisVMLoadCombnations.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	*section index*
ChainIndex	*section chain index*
MinMaxType	*Minimum or maximum values*
CombinationType	*Combination type*
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 **GetSegmentChainSurfaceStressesByLoadCaseId** ([in] long **Index,** [in] long **ChainIndex,** [in] long **LoadCaseId,** [in] long **LoadLevelOrTimeStep,** [in] EAnalysisType **AnalysisType,**[out] SAFEARRAY(RPoint3D)* **Coords,** [out] SAFEARRAY(RSurfaceStressValuesTMB)* **Stresses**, [out] SAFEARRAY (BSTR)* **Combinations**)

Index	*section index*
ChainIndex	*section chain index*
LoadCaseId	*load case index* *(0 < LoadCaseId ≤ AxisVMLoadCases.Count)*
LoadLevelOrTimeStep	*load level (increment) index or time step*
AnalysisType	*Type of Analysis*
Coords	*section chain's coordinates*
Stresses	*section chain's surface stress values*
Combinations	*array of string with name of load cases.*

Get section chain's surface stress values

If successful, returns number of surface stress values, otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBounds, seSectionTypeIsNotSegment).

---

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	*section index*
ChainIndex	*section chain index*
LoadCombinationId	*load combination index* *(0 < LoadCombinationId ≤ AxisVMLoadCombnations.Count)*
LoadLevelOrTimeStep	*load level (increment) index or time step*
AnalysisType	*Type of Analysis*
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 **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 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

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 lbTrue if nonlinear analysis results consider concrete creep. More here…

IAxisVMSettings

Application settings of AxisVM

Error codes

enum **ESettingsError**= {

seteInvalidGravityDirection = –100001	*gravity direction vector is not valid*
seteInvalidGravityAcceleration = –100002	*gravity acceleration is not valid*

}

Enumerated types

enum **EPlaneToleranceType** = {

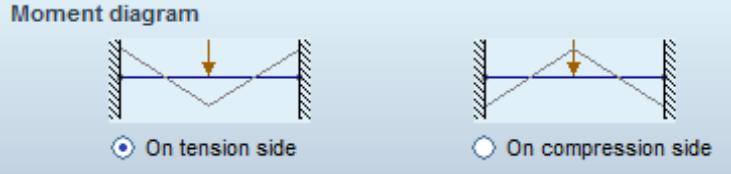
ptRelativePerThousand = 0x01,	*plane tolerance in per thousands*
ptAbsolute = 0x02 }	*plane tolerance by value*

Plane tolerance type.

enum **EMomentDiagramType** = {

mdtOnTensionSide = 0x01,	*positive value on tension side*
mdtOnCompressionSide = 0x02 }	*positive value on compression side*

Moment diagram type.



enum **ETempFolderType** = {

tftModel = 0x01,	*temp. files saved in same folder where model is saved*
tftSystem = 0x02,	*temp. files saved in local system temp. folder*
tftCustom = 0x03 }	*temp. files saved in custom folder*

Location of temp. files.

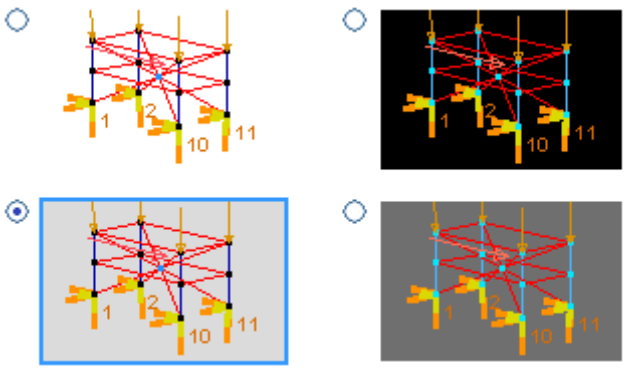
enum **EGridType** = {

gtGridLines = 0x01,	*Show background grid lines*
gtDotGrid= 0x02 }	*Show background grid dots*

Type of grid on a background.

enum **EBackgroundColour**= {

bcWhite = 0x00,	*Background colour will be white*
bcBlack = 0x01,	*Background colour will be black*
bcLightGrey = 0x02,	*Background colour will be light grey*
bcDarkGrey = 0x03 }	*Background colour will be dark grey*



These options correspond to Settings/Preferences/Colors in AxisVM menu.

enum **EGeometryUnitType** = {

gut_Geom_Distance = 0x00,	*Distance units*
gut_Geom_Angle = 0x01,	*Angular units*
gut_Geom_Struct_size = 0x02 }	*Structural units*

enum **ECrossSectionUnitType** = {

csut_Size = 0x00,	*Cross-section's dimension units*
csut_Area = 0x01,	*Cross-section's area units*
csut_Static_moment = 0x02 ,	*Cross-section's static moment units*
csut_Area_Moment_Inertia = 0x03,	*Cross-section's moment inertia units*
csut_Warping_constant = 0x04 }	*Cross-section's warping constant units*

enum **EMaterialUnitType** = {

mut_Young_modulus = 0x00,	*Material's Young modulus units*
mut_Mass = 0x01,	*Material's mass units*
mut_Limit_stress = 0x02 ,	*Material's limit stress units*
mut_Limit_strain = 0x03 }	*Material's limit strain units*

enum **EPropertiesUnitType** = {

put_Beam_length = 0x00,	*Beam length units*
put_Thickness= 0x01,	*Thickness units*
put_Surface = 0x02 ,	*Surface area units*
put_Volume = 0x03,	*Volume units*
put_Mass = 0x04,	*Mass units*
put_put_Mass_per_length = 0x05,	*Mass per length units*
put_Gap_opening = 0x06 }	*Gap width units*

enum **EStiffnessUnitType** = {

sut_Translational = 0x00,	*Translational stiffness units*
sut_Rotational= 0x01,	*Rotational stiffness units*
sut_Line_translational = 0x02 ,	*Line translational stiffness units*
sut_Line_rotational = 0x03,	*line rotational stiffness units*
sut_Surface= 0x04 }	*Surface stiffness units*

enum **ELoadsUnitType** = {

lut_Force = 0x00,	*Force units*
lut_Moment = 0x01,	*moment units*
lut_Line_force = 0x02 ,	*Linear force units*
lut_Line_force_moment = 0x03,	*Linear moment units*
lut_Surface_force = 0x04,	*Surface force units*
lut_Temperature = 0x05,	*Temperature units*
lut_Temperature_variation = 0x06,	*Temperature change units*
lut_Design_fire_load_density = 0x07,	*Fire intensity units*
lut_Specific_heat = 0x08 ,	*Specific heat units*
lut_Section_factor = 0x09,	*Section factor units*
lut_Fire_duration = 0x0A }	*Fire duration units*

enum **ELoadsUnitType** = {

lut_Force = 0x00,	*Force units*
lut_Moment = 0x01,	*moment units*
lut_Line_force = 0x02 ,	*Linear force units*
lut_Line_force_moment = 0x03,	*Linear moment units*
lut_Surface_force = 0x04,	*Surface force units*
lut_Temperature = 0x05,	*Temperature units*
lut_Temperature_variation = 0x06,	*Temperature change units*
lut_Design_fire_load_density = 0x07,	*Fire intensity units*
lut_Specific_heat = 0x08 ,	*Specific heat units*
lut_Section_factor = 0x09,	*Section factor units*
lut_Fire_duration = 0x0A }	*Fire duration units*

enum **EStaticUnitType** = {

sut_Displacement = 0x00,	*Displacement result units*
sut_Rotation = 0x01,	*Rotation result units*
sut_Force = 0x02 ,	*Force result units*
sut_Moment = 0x03,	*Moment result units*
sut_DistrForce = 0x04,	*Distributed force result units*
sut_DistrMoment = 0x05,	*Distributed moment result units*
sut_DistrSurfaceForce = 0x06,	*Distributed surface force result units*
sut_Stress = 0x07}	*Stress result units*

enum **ERCDesignUnitType** = {

rcdut_RebarDia = 0x00,	*Rebar diameter units*
rcdut_RebarDistance = 0x01,	*Rebar distance (spacing) units*
rcdut_ReinfArea = 0x02 ,	*Reinforcement area units*
rcdut_ShearReinfArea = 0x03,	*Shear reinforcement area units*
rcdut_Cracking = 0x04,	*Crack width units*
rcdut_Eccentricity = 0x05 }	*Eccentricity units*

enum **ESteelDesignUnitType** = {

sdut_Buckling_factor = 0x00,	*Steel buckling factor units*
sdut_Check_components = 0x01 }	*Steel design component units*

enum **ETimberDesignUnitType** = {

tdut_Check_components = 0x00 }	*Timber design component units*

enum **EDimensioningUnitType** = {

dut_Dim_Distance = 0x00,	*Distance units*
dut_Dim_Angle = 0x01,	*Angular units*
dut_Level_symobol = 0x02 ,	*Storey level units*
dut_Graphics_size = 0x03 }	*Graphics size units*

enum **ENL_ConsequenceClass = {**
Warning! Only for dutch national code.

nlcc_Invalid = 0x00,	*invalid consequence class (returned if the current national code is not a dutch one)*
nlcc_CC1 = 0x01,	*consequence class 1*

nIcc_CC2 = 0x02 , *consequence class 2*
nIcc_CC3 = 0x03 } *consequence class 3*

Records / structures

RPlaneTolerance = (

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

)

RGridOptions = (

ELongBoolean	**DisplayGrid**	*if lbTrue then grid is visible*
double	**DeltaX**	*Spacing of grids in global x direction*
double	**DeltaY**	*Spacing of grids in global y direction*
double	**DeltaZ**	*Spacing of grids in global z direction*
EGridType	**GridType**	*Type of grid on a background.*

)

RCursorSnap = (

ELongBoolean	**MouseSnap**	*if lbTrue then mouse snap is enabled*
double	**DeltaX**	*Cusor snaping in global x direction*
double	**DeltaY**	*Cusor snaping in global y direction*
double	**DeltaZ**	*Cusor snaping in global z direction*
double	**CtrlX**	*Fine cusor snapingwhile holding Ctrl key*

)

REditingOptions = (

double	**ConstAngle_DeltaAlpha**	*Constraint angle delta alpha*
double	**ConstAngle_CustomAlpha**	*Constraint angle custom alpha*
double	**EditingToler**	*Editing tolerance*

These options correspond to Options/Editing in AxisVM menu.

)

RUnitParameters = (

long	**ConversionBaseID**	*AvailableUnits array index of unit used as base for conversion*
long	**UsedID**	*AvailableUnits array index of unit used as current*
long	**DecimalPlaces**	*number of decimal places, -1 if exponential*
double	**Multiplier**	*Multiplier used for conversion*

)

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 *plane tolerance*

Get the plane tolerance. If unsuccessful, returns an error code (errDatabaseNotReady).

---

long **GetTempFolderType** ([out] ETempFolderType **TempFolderType,** [out] BSTR **Path**)

TempFolderType *Type of temp. file folder*

Path *Full path of temp. file folder*

Get the temp. folder. If unsuccessful, returns an error code (errDatabaseNotReady).

---

long **GetUnitParams_Geometry** ([in] EGeometryUnitType **UnitType**,
[in] ELongBoolean **MathText** , [in] ELongBoolean **Default,**
[out] SAFEARRAY(BSTR)* **AvailableUnits,** [i/o] RUnitParameters **UnitParameters**)

UnitType *Type (category) of the units*

MathText *Use MathText format for unit string*

Default *If lbTrue, then default units are returned, otherwise user units*

AvailableUnits *String array with units*

UnitParameters *All unit parameters as output*

Get units for geometry. If unsuccessful, returns an error code (errDatabaseNotReady).

---

long **GetUnitParams_CrossSection** ([in] ECrossSectionUnitType **UnitType**,
[in] ELongBoolean **MathText** , [in] ELongBoolean **Default,**
[out] SAFEARRAY(BSTR)* **AvailableUnits,** [i/o] RUnitParameters **UnitParameters**)

UnitType *Type (category) of the units*

MathText *Use MathText format for unit string*

Default *If lbTrue, then default units are returned, otherwise user units*

AvailableUnits *String array with units*

UnitParameters *All unit parameters as output*

Get units for geometry. If unsuccessful, returns an error code (errDatabaseNotReady).

---

long **GetUnitParams_Material** ([in] EMaterialUnitType **UnitType**,
[in] ELongBoolean **MathText** , [in] ELongBoolean **Default,**
[out] SAFEARRAY(BSTR)* **AvailableUnits,** [i/o] RUnitParameters **UnitParameters**)

UnitType *Type (category) of the units*

MathText *Use MathText format for unit string*

Default *If lbTrue, then default units are returned, otherwise user units*

AvailableUnits *String array with units*

UnitParameters *All unit parameters as output*

Get units for geometry. If unsuccessful, returns an error code (errDatabaseNotReady).

---

long **GetUnitParams_Properties** ([in] EPropertiesUnitType **UnitType**,
[in] ELongBoolean **MathText** , [in] ELongBoolean **Default,**
[out] SAFEARRAY(BSTR)* **AvailableUnits,** [i/o] RUnitParameters **UnitParameters**)

UnitType	*Type (category) of the units*
MathText	*Use MathText format for unit string*
Default	*If lbTrue, then default units are returned, otherwise user units*
AvailableUnits	*String array with units*
UnitParameters	*All unit parameters as output*

Get units for geometry. If unsuccessful, returns an error code (errDatabaseNotReady).

---

long **GetUnitParams_Stiffness** ([in] EStiffnessUnitType **UnitType**,
[in] ELongBoolean **MathText** , [in] ELongBoolean **Default,**
[out] SAFEARRAY(BSTR)* **AvailableUnits,** [i/o] RUnitParameters **UnitParameters**)

UnitType	*Type (category) of the units*
MathText	*Use MathText format for unit string*
Default	*If lbTrue, then default units are returned, otherwise user units*
AvailableUnits	*String array with units*
UnitParameters	*All unit parameters as output*

Get units for geometry. If unsuccessful, returns an error code (errDatabaseNotReady).

---

long **GetUnitParams_Loads** ([in] ELoadsUnitType **UnitType**,
[in] ELongBoolean **MathText** , [in] ELongBoolean **Default,**
[out] SAFEARRAY(BSTR)* **AvailableUnits,** [i/o] RUnitParameters **UnitParameters**)

UnitType	*Type (category) of the units*
MathText	*Use MathText format for unit string*
Default	*If lbTrue, then default units are returned, otherwise user units*
AvailableUnits	*String array with units*
UnitParameters	*All unit parameters as output*

Get units for geometry. If unsuccessful, returns an error code (errDatabaseNotReady).

---

long **GetUnitParams_Static** ([in] EStaticUnitType **UnitType**,
[in] ELongBoolean **MathText** , [in] ELongBoolean **Default,**
[out] SAFEARRAY(BSTR)* **AvailableUnits,** [i/o] RUnitParameters **UnitParameters**)

UnitType	*Type (category) of the units*
MathText	*Use MathText format for unit string*
Default	*If lbTrue, then default units are returned, otherwise user units*
AvailableUnits	*String array with units*
UnitParameters	*All unit parameters as output*

Get units for geometry. If unsuccessful, returns an error code (errDatabaseNotReady).

---

long **GetUnitParams_RC_design** ([in] ERCDesignUnitType **UnitType**,
[in] ELongBoolean **MathText** , [in] ELongBoolean **Default,**
[out] SAFEARRAY(BSTR)* **AvailableUnits,** [i/o] RUnitParameters **UnitParameters**)

UnitType	*Type (category) of the units*
MathText	*Use MathText format for unit string*
Default	*If lbTrue, then default units are returned, otherwise user units*
AvailableUnits	*String array with units*
UnitParameters	*All unit parameters as output*

Get units for geometry. If unsuccessful, returns an error code (errDatabaseNotReady).

---

long **GetUnitParams_Steel_design** ([in] ESteelDesignUnitType **UnitType**,
[in] ELongBoolean **MathText** , [in] ELongBoolean **Default,**
[out] SAFEARRAY(BSTR)* **AvailableUnits,** [i/o] RUnitParameters **UnitParameters**)

UnitType *Type (category) of the units*
MathText *Use MathText format for unit string*
Default *If lbTrue, then default units are returned, otherwise user units*
AvailableUnits *String array with units*
UnitParameters *All unit parameters as output*

Get units for geometry. If unsuccessful, returns an error code (errDatabaseNotReady).

---

long **GetUnitParams_Timber_design** ([in] ETimberDesignUnitType **UnitType**,
[in] ELongBoolean **MathText** , [in] ELongBoolean **Default,**
[out] SAFEARRAY(BSTR)* **AvailableUnits,** [i/o] RUnitParameters **UnitParameters**)

UnitType *Type (category) of the units*
MathText *Use MathText format for unit string*
Default *If lbTrue, then default units are returned, otherwise user units*
AvailableUnits *String array with units*
UnitParameters *All unit parameters as output*

Get units for geometry. If unsuccessful, returns an error code (errDatabaseNotReady).

---

long **GetUnitParams_Dimensioning** ([in] EDimensioningUnitType **UnitType**,
[in] ELongBoolean **MathText** , [in] ELongBoolean **Default,**
[out] SAFEARRAY(BSTR)* **AvailableUnits,** [i/o] RUnitParameters **UnitParameters**)

UnitType *Type (category) of the units*
MathText *Use MathText format for unit string*
Default *If lbTrue, then default units are returned, otherwise user units*
AvailableUnits *String array with units*
UnitParameters *All unit parameters as output*

Get units for geometry. If unsuccessful, returns an error code (errDatabaseNotReady).

---

long **SetEditingOptions** ([i/o] REditingOptions **Value**)

Value *editing options*

Set the editing options. If unsuccessful, returns an error code (errDatabaseNotReady).

---

long **SetCursorSnap** ([i/o] RGridOptions **Value**)

Value *All cursor snap options*

Set the cursor snap options. If unsuccessful, returns an error code (errDatabaseNotReady).

---

long **SetGravity** ([i/o] RPoint3d **Direction,** [out] double **Acceleration**)

Direction *Gravity direction*
Acceleration *Gravity acceleration*

Set the direction and acceleration of gravity. If unsuccessful, returns an error code (seteInvalidGravityAcceleration, seteInvalidGravityDirection, errDatabaseNotReady).

---

long **SetGridOptions** ([i/o] RGridOptions **Value**)

Value *All grid options*

Set the grid options. If unsuccessful, returns an error code (errDatabaseNotReady).

---

long **SetPlaneTolerance** ([i/o] RPlaneTolerance **Value**)

Value *plane tolerance*

Set the plane tolerance. If unsuccessful, returns an error code (errDatabaseNotReady).

---

long **SetTempFolderType** ([in] ETempFolderType **TempFolderType,** [in] BSTR **Path**)

TempFolderType *Type of temp. file folder*

Path *Full path of temp. file folder*

Set the temp. folder. If unsuccessful, returns an error code (errDatabaseNotReady).

ELongBoolean **EnvironmentClassIsValid** ([in] EEnvironmentClass **EnvironmentClass**)

EnvironmentClass *environment class*

Determines whether EnvironmentClass is supported by the used national design code or not.

Properties

Specifying invalid values triggers an error event (See IAxisVMSettingsEvents)

EBackgroundColour **BackgroundColour** • *Get or set the colour of the background*

double **CrossSectionEditingTolerance** • *Get or set the editing tolerance in the cross-section editor.*

double **Editing Tolerance** • *Get or set the editing tolerance.*

ELongBoolean **FixedMesh** • *Get or set whether meshes are created and removed automatically (True). If False, meshes remain editable.*

EMomentDiagramType **MomentDiagramType** • *Get or set the type of the moment diagram.*

ENationalDesignCode **NationalDesignCode** • *Get or set the current national design code.*

ELongBoolean **NodeBreaksUnmeshedMember** • *Get or set whether node insertion breaks unmeshed structural members.*

ENL_ConsequenceClass **NL_ConsequenceClass** • *Get or set the consequence class. Valid only if the national code is a dutch one.*

ELanguage **Program Language** • *Get or set the used in dialog boxes, menus, etc.*

ELanguage **ReportLanguage** • *Get or set the used in documentation.*

double **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 = 0,	*results are OK*
ssreEmptyStorey = -100001	*empty storey in the model*
ssreInclinedElementExists = -100002	*inclined elements in the model*
ssreTorsionResultsMissing = -100003 }	*no torsion results*

Error type of seismic sensitivity results

Records / structures

RSeismicSensitivityResults = (

Warning! *This record was superseded by RSeismicSensitivityResults_V171*

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 gavity in x direction*
double	**Gm_y**	*coordinate of centre of gavity 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	**Error**	*Error type*

)

RSeismicSensitivityResults_V171 = (

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	**S_z**	*coordinate of shear (torsion) centre in z direction*
double	**Gm_x**	*coordinate of centre of gavity in x direction*
double	**Gm_y**	*coordinate of centre of gavity 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	**Error**	*Error type*

)

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).

---

long **GetSeismicSensitivityResults** ([in] long **index**,
[i/o] RSeismicSensitivityResults **SeismicSensitivityResults**)

Warning! *This function has become obsolete, was superseded by GetSeismicSensitivityResults_V171*

index *Storey index*

SeismicSensitivityResults *All seismic sensitivity results*

Get seismic sensitivity results. If successful, returns index of seismic storey, otherwise an error code (errDatabaseNotReady or errIndexOutOfBounds).

---

long **GetSeismicSensitivityResults_V171** ([in] long **index**,
[i/o] RSeismicSensitivityResults_V171 **SeismicSensitivityResults**)

index *Storey index*

SeismicSensitivityResults *All seismic sensitivity results*

Get seismic sensitivity results. If successful, returns index of seismic 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 = -100006	*only parametric spectrum is allowed*

}

Records / structures

RSpectrumData_EC = (

ESubsoilClass	**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	**gammaI**	*See importance factor γ_I 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 = (

ESubsoilClass	**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	**gammaI**	*See importance factor γ_I 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)*
ELongBoolean	**LocalSpectrum**	*local spectrum according to the Hungarian Chamber of Engineers Local spectra manual*
double	**F0**	*F0 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	**gammaI**	*See importance factor γ_I 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 = (

ESubsoilClass	**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*
ETopographicCategory	**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 = (

ESubsoilClass	**SubsoilClass**	*Subsoil class*
double	**agr**	*Gravity acceleration [m/s^2]*
double	**S**	*see seismic calculation based on SIA in AxisVM manual*
double	**TB**	*see seismic calculation based on SIA in AxisVM manual*
double	**TC**	*see seismic calculation based on SIA in AxisVM manual*
double	**TD**	*see seismic calculation based on SIA in AxisVM manual*
double	**gammaI**	*Importance factor, see seismic calculation based on SIA in AxisVM manual*
double	**qx**	*See q in seismic calculation based on SIA in AxisVM manual (Behaviour factor in x direction)*
double	**qy**	*See q in seismic calculation based on SIA in AxisVM manual (Behaviour factor in y direction)*

)

RSpectrumData_STAS = (

ESubsoilClass	**SubsoilClass**	*Subsoil class*
double	**agr**	*Gravity acceleration [m/s^2]*
double	**beta0**	*see seismic calculation based on STAS in AxisVM manual*
double	**TB**	*see seismic calculation based on STAS in AxisVM manual*
double	**TC**	*see seismic calculation based on STAS in AxisVM manual*
double	**TD**	*see seismic calculation based on STAS in AxisVM manual*
double	**gammaI**	*Importance factor, see seismic calculation based on STAS in AxisVM manual*
double	**qx**	*Behaviour factor in x direction, see seismic calculation in AxisVM manual*
double	**qy**	*Behaviour factor in y direction, see seismic calculation in AxisVM manual*

)

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 = (

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_ECHU	**SpectrumData_ECHU**	*Spectrum parameters according to hungarian EC*
RSpectrumData_ECNL	**SpectrumData_ECNL**	*Spectrum parameters according to dutch EC*
RSpectrumData_STAS	**SpectrumData_STAS**	*Spectrum parameters according to Romanian national code*
RSpectrumData_DIN	**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	*Non linear elastic spring with contradictory parameters*
speNegativeValueMustBePositive = -100002	*A negative value is present in a field where only positive values are valid*
speNLPInconsistency = -100003	*Non linear plastic spring with contradictory parameters*
speWFOutOfRange = -100004	*Warping factor is out of the range [-1.00...+1.00]*
speInconsistency = -100005	*elastic spring with contradictory parameters*

Enumerated types

enum **ESpringParType** = {

sptNodal = 0	*Spring characteristics for nodal supports (but not for seismic isolators)*
sptIsolator = 1	*Spring characteristics for nodal seismic isolators*
sptConstraint = 2	*Spring characteristics for warping constraint*
sptLine = 3}	*Spring characteristics for line supports (like for EdgeConnections)*

enum **ESpringParNNType** = {

spnntLinear = 0	*Linear spring characteristics*
spnntNonLinearElastic = 1	*Non linear elastic spring characteristics*
spnntNonLinearPlastic = 2}	*Non linear plastic spring characteristics*

enum **ESpringParDOFType** = {

spdoftTranslation = 0	*Translational spring characteristics*
spdoftRotation = 1}	*Rotational spring characteristics*

enum **ESpringParDampingType** = {

spdtKelvin = 0	*Type of dynamic damping : Kelvin-Voigt*
spdtMaxwell = 1}	*Type of dynamic damping : Maxwell*

enum **ESpringParNonLinearity** = {

spnlTensionAndCompression = 0	*Type of nonlinearity : both tension and compression*
spnlTensionOnly = 1	*Type of nonlinearity : only for tension*
spnlCompressionOnly = 2}	*Type of nonlinearity : only for compression*

enum **ESpringParNLDefType** = {

spnldtByParam = 0	*Non linear behaviour is defined through numeric parameters*
spnldtByFunction = 1}	*Non linear behaviour is defined through a function*

enum **ESpringParHardeningRule** = {

sphrIsotropic = 0	*Hardening rule for plastic spring model : isotropic*
sphrKinematic = 1}	*Hardening rule for plastic spring model : Kinematic*

enum **ESpringParMatrixType** = {

spmtTangentMatrix = 0	*Stiffness for plastic spring model : tangent stiffness*
spmtInitialMatrix = 1}	*Stiffness for plastic spring model : initial stiffness*

enum **ESpringParIsolatorType** = {

spitRubber = 0	*Seismic isolator type : rubber bearing*
spitSlider = 1	*Seismic isolator type : curved surface slider*
spitCustom = 2}	*Seismic isolator type : custom*

Records / structures

RSpringParam = (

Warning! This record has become obsolete, it was superseded by RSpringParam_V161

ESpringParType	**SpringType**	*Spring characteristic type (nodal, seismic isolator, …)*

ESpringParNNType	**NNType**	*Linear / non linear type of the spring characteristic*
ESpringParDOFType	**DOFType**	*Translational / rotational spring characteristic*
ELongBoolean	**NLESimplified**	*User selection for non linear elastic case : simplified or not*
double	**K**	*Initial stiffness [KN/m]*
double	**KVib**	*Vibration stiffness [KN/m]*
ESpringParDampingType	**DampingType**	*Type of dynamic damping*
double	**C**	*Dynamic damping [KN/m/s]*
ESpringParNonLinearity	**NonLinearity**	*Non linearity type : linear, non linear elastic, non linear plastic*
ESpringParNLDefType	**NLDefType**	*Non linear characteristics are defined by parameters or by a function*
double	**K_T**	*Stiffness for tension [KN/m]*
double	**K_C**	*Sfiffness for compression [KN/m]*
ELongBoolean	**ResistanceDef_T**	*Resistance for tension is defined*
ELongBoolean	**ResistanceDef_C**	*Resistance for compression is defined*
double	**TangentStiffness_T**	*Stiffness for tension tangent [KN/m]*
double	**TangentStiffness_C**	*Stiffness for compression tangent [KN/m]*
double	**Resistance_T**	*Resistance for tension [KN]*
double	**Resistance_C**	*Resistance for compression [KN]*
ELongBoolean	**ResistanceDef_T**	*Resistance for tension is defined*
ELongBoolean	**ResistanceDef_C**	*Resistance for compression is defined*
ESpringParHardeningRule	**HardeningRule**	*Hardening rule for plastic spring model*
ESpringParMatrixType	**MatrixType**	*Type of stiffness for plastic spring model*
double	**C_T**	*Damping for tension [KN/(m/s)]*
double	**C_C**	*Damping for compression [KN/(m/s)]*
double	**VerticalStiffness**	*Vertical stiffness for seismic isolator [KN/m]*
ESpringParIsolatorType	**IsolatorType**	*Seismic isolator type*
double	**K1**	*Initial stiffness (only for rubber seismic isolator) [KN/m]*
double	**KT**	*Tangent stiffness (only for rubber seismic isolator) [KN/m]*
double	**F1**	*Resistance (only for rubber seismic isolator) [KN]*
double	**Mu**	*Friction coefficient (only for slider seismic isolator) [ ]*
double	**R**	*Radius (only for slider seismic isolator) [m]*
double	**HorizontalStiffness**	*Horizontal stiffness (only for custom seismic isolator) [KN/m]*
double	**Ksi**	*Damping ratio (only for custom seismic isolator) [ ]*
	)	

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, Ksi

RSpringParam_V161 = (

ESpringParType	**SpringType**	*Spring characteristic type (nodal, seismic isolator, …)*
ESpringParNNType	**NNType**	*Linear / non linear type of the spring characteristic*
ESpringParDOFType	**DOFType**	*Translational / rotational spring characteristic*
ELongBoolean	**NLESimplified**	*User selection for non linear elastic case : simplified or not*
double	**K**	*Initial stiffness [KN/m]*
double	**KVib**	*Vibration stiffness [KN/m]*
ESpringParDampingType	**DampingType**	*Type of dynamic damping*
double	**C**	*Dynamic damping [KN/m/s]*
ESpringParNonLinearity	**NonLinearity**	*Non linearity type : linear, non linear elastic, non linear plastic*
ESpringParNLDefType	**NLDefType**	*Non linear characteristics are defined by parameters or by a function*
double	**K_T**	*Stiffness for tension [KN/m]*
double	**K_C**	*Sfiffness for compression [KN/m]*
ELongBoolean	**ResistanceDef_T**	*Resistance for tension is defined*
ELongBoolean	**ResistanceDef_C**	*Resistance for compression is defined*
double	**TangentStiffness_T**	*Stiffness for tension tangent [KN/m]*
double	**TangentStiffness_C**	*Stiffness for compression tangent [KN/m]*
double	**Resistance_T**	*Resistance for tension [KN]*
double	**Resistance_C**	*Resistance for compression [KN]*
ELongBoolean	**ResistanceDef_T**	*Resistance for tension is defined*
ELongBoolean	**ResistanceDef_C**	*Resistance for compression is defined*
ESpringParHardeningRule	**HardeningRule**	*Hardening rule for plastic spring model*
ESpringParMatrixType	**MatrixType**	*Type of stiffness for plastic spring model*
double	**C_T**	*Damping for tension [KN/(m/s)]*
double	**C_C**	*Damping for compression [KN/(m/s)]*
double	**VerticalStiffness**	*Vertical stiffness for seismic isolator [KN/m]*
ESpringParIsolatorType	**IsolatorType**	*Seismic isolator type*
double	**K1**	*Initial stiffness (only for rubber seismic isolator) [KN/m]*
double	**KT**	*Tangent stiffness (only for rubber seismic isolator) [KN/m]*
double	**F1**	*Resistance (only for rubber seismic isolator) [KN]*
double	**Mu**	*Friction coefficient (only for slider seismic isolator) [ ]*
double	**R**	*Radius (only for slider seismic isolator) [m]*
double	**HorizontalStiffness**	*Horizontal stiffness (only for custom seismic isolator) [KN/m]*
double	**Ksi**	*Damping ratio (only for custom seismic isolator) [ ]*
double	**WF**	*Warping factor (only for SpringType = sptConstraint) [ ]*

)

RSpringParam_V161 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, Ksi

*SpringType = **sptConstraint** : fields [WF] :*

Functions

long **Add** ([in] BSTR **Name**, [in] RSpringParam* **Value**)

Warning! *This function has become obsolete, was superseded by Add_V161*

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, speNLPInconsistency, speNegativeValueMustBePositive).

---

long **Add_V161** ([in] BSTR **Name**, [in] RSpringParam_V161* **Value**)

Name *Name of the spring characteristics (should be unique)*
Value *The RSpringParam_V161 record defining the spring characteristics. See RSpringParam_V161 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, speNLPInconsistency, speNegativeValueMustBePositive, speWFOutOfRange).

---

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 **IndexOfName** ([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**

Warning! *This property has become obsolete, was superseded by FullRec_V161*

The properties of the spring characteristic. See RSpringParam valid field combinations for further details

RSpringParam_V161 **FullRec_V161** *The properties of the spring characteristic. See RSpringParam_V161 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 is empty*
sdmeInvalidNationalDesignCode = -100003	*IAxisVMSteelDesignMembers does not support the selected design code*
sdmeNotConnectingLines = -100004	*LineIDs are not connected*
sdmeDesignParametersNotValidForUsedDesignCode = -100005	*Design parameters are not valid for used design code*
sdmeInvalidLines = -100006	*among the LineIDs some are not a valid target for steel design*
sdmeArrayLengthMismatch = -100007	*the length of input SafeArrays is different*
sdmeFseMustBePositive = -100008	*fse must be positive (the default value is 1)*

Enumerated types

enum **EDesignApproach** = {

daClass = 0	*Depending on classof the section*
daElastic = 1}	*Elastic design approach*

Approach used for steel design

enum **EMcrMethod** = {

mcrmAuto = 0	*Mcr (Auto Mcr option in AxisVM) is automatically calculated based on FE analysis (C1,C2 and C3 are ignored) Used prior to version 13 release 2*
mcrmC1Lopez = 1	*C1 value calculated using Lopez's method*
mcrmC1C2C3User = 2	*C1,C2 and C3 values must be set by user*
mcrmDutch = 3	*C1 and C2 values are automatically calculated*
mcrmDutchUser = 4	*C1 and C2 values must be set by user*
mcrmAutoLS = 5	*updated version of mcrmAuto method, considering* **ESteelLateralSupports** *Used from version 13 release 2 (Auto Mcr option in AxisVM)*
mcrmUser = 6 }	*Mcr is given by the user*

What method will be used for calculation of C1, C2 and C3 parameters see the AxisVM manual and
Appendix F1.2 of ENV 1993-1-1.

enum **EStiffeners** = {

sNo = 0	*No stiffeners*
sTransversal = 1}	*Transversal stiffeners*

Type of stiffener.

enum **ETorsion** = {

tFree = 0	*Free to rotate about local x*
tPartial = 1	*One end of beam fixed in torsion*
tFixed = 2}	*Both ends of beam fixed in torsion*

Type of torsion.

enum **ESteelBucklingCurves** = {

sbc_Auto = 0,	*auto*
sbc_a0 = 1,	*a0*
sbc_a = 2,	*a*
sbc_b = 1,	*b*
sbc_c = 2,	*c*
sbc_d = 3}	*d*

enum **ESteelBucklingLengthMode**= {

sblm_Factor= 0,	*buckling length given by factors*
sblm_Length= 1,	*buckling length given by length*
sblm_Auto= 2,	*automatic buckling length calculation*
sblm_None = 3}	*none*

enum **ESteelCantileverFixedEnd**= {

scfeStartNode = 0,	*start node of the element is fixed*
scfeEndNode = 1}	*end node of the element is fixed*

enum **ESteelLateralSupports**= {

ammAuto = 0,	*automatic*
ammEstimatedFromKzKw = 1,	*estimated from kz, kw*
ammForkSupports = 2,	*fork supports*
ammUserDefined = 3}	*user defined (lateral supports must be defined)*

enum **ESteelLTBucklingCurves** = {

sltbc_Auto = 0,	*auto*
sltbc_a = 1,	*a*
sltbc_b = 2,	*b*
sltbc_c = 1,	*c*
sltbc_d = 2}	*d*

enum **ESteelLoadPosition**= {

slp_Actual = 0,	*actual position (as is in the model)*
slp_Top = 1,	*top of cross-section*
slp_CenterOfGravity = 2,	*center of gravity of cross-section*
slp_Bottom = 3,	*bottom of cross-section*
slp_Custom = 4}	*user defined position*

enum **ESteelFireParBetaMethod**= {

sfpbm_Auto = 0,	*factors depending on the moment diagram : automatic*
sfpbm_User = 1}	*factors depending on the moment diagram : user defined*

enum **ESteelSLSHMethod**= {

sslshm_Member = 0,	*height is the member length*
sslshm_Structure = 1,	*height is the height of the entire structure*
sslshm_Custom = 2,	*height is measured form height h*
sslshm_Level = 3}	*not used*

enum **ESteelSLSEMethod**= {

sslsem_No = 0,	*deflection based on actual displacements*
sslsem_2 = 1,	*deflection based on displacement relative to both left and right*
sslsem_Left = 2,	*deflection based on displacement relative to the left*
sslsem_Right = 3}	*deflection based on displacement relative to the right*

enum **ESteelSLSLMethod**= {

sslslm_Member = 0,	*design member length is the actual length of the member*
sslslm_Custom = 1,	*design member length is specified by the slsCustomLy and/or slsCustomLz*
sslslm_Conn = 2}	*design member length is based on connecting members and supports*

enum **ESteelSLSPreCamberCurve**= {

sslspcc_Quadratic = 0,	*quadratic*
sslslm_Linear = 1}	*linear*

Records / structures

RSteelDesignParameters_EC_SIA_ITA = (

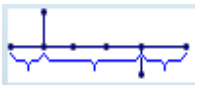
Warning! *This record has become obsolete, it was superseded by RSteelDesignParameters_EC_SIA_ITA_V153*

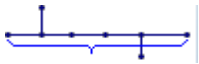
ELongBoolean	**BreakAtElements**	*If Ttrue:* *Break members at side at side element connections or nodal supports.*  *If False:* *The member becomes only one structural element irrespective of other elements (emmbers etc.) connecting to its nodes.* 
double	**a**	*distance of trans. stiffeners (used only if Stiffeners = sTransversal)*
double	**akr**	*critical load parameter factor (not used)*

double	**C1**	*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**	*Design approach*
double	**Ky**	*buckling length factor about local y axis (used only if BucklingLengthModeY = sblm_Factor)*
double	**Kz**	*buckling length factor about local z axis (used only if BucklingLengthModeZ = sblm_Factor)*
double	**kt**	*shear buckling coefficient (not used)*
double	**Kw**	*warping constraint factor (see steel design in Axis VM manual; used only if McrMethod <> mcrmAuto)*
EMcrMethod	**McrMethod**	*Define how to determine Mcr*
EStiffeners	**Stiffeners**	*type of stiffeners (used only if WebShearBuckling = lbTrue)*
long	**SDP_Class**	*section class (automatic if 0)*
ELongBoolean	**YBraced**	*True if braced Y direction*
ELongBoolean	**ZBraced**	*True if braced Z direction*
double	**Za**	*Relative load position in local z axis (see steel design parameters - load position in Axis VM manual)*
doube	**fse**	*load factor for seismic forces (default is 1 =>no change); note: it was ks before.*
ELongBoolean	**Cantilever**	*cantilever checkbox*
ESteelCantileverFixedEnd	**CantileverFixedEnd**	*start or end node of cantilever is fixed*
ELongBoolean	**FlexuralBuckling**	*flexural buckling*
ELongBoolean	**LateralTorsionalBuckling**	*lateral torsional buvkling*
ELongBoolean	**WebShearBuckling**	*web shear buckling*
ESteelBucklingLengthMode	**BucklingLengthModeY**	*buckling length calculation mode related to y-y axis*
ESteelBucklingLengthMode	**BucklingLengthModeZ**	*buckling length calculation mode related to z-z axis*
double	**Ly**	*buckling length related to y-y axis (used only if BucklingLengthModeY = sblm_Length)*
double	**Lz**	*buckling length related to z-z axis (used only if BucklingLengthModeZ = sblm_Length)*
ELongBoolean	**ConsiderN**	*consider the effect of normal force*
double	**Eta**	*η factor for web shear buckling checks (used only if Stiffeners = sTransversal)*
ESteelLateralSupports	**LateralSupports**	*lateral supports*
double	**Mcr**	*used defined Mcr (used only if LateralSupports = ammUserDefined)*

Steel design parameters in accordance with Euro code, Swiss and Italian national codes. For more info, see steel design in Axis VM manual.

RSteelDesignParameters_EC_SIA_ITA_V153 = (

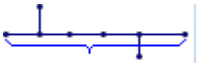
ELongBoolean	**BreakAtElements**	*If Ttrue:* *Break members at side at side element connections or nodal supports.* 

		If False: *The member becomes only one structural element irrespective of other elements (emmbers etc.) connecting to its nodes.* 
double	**a**	*distance of trans. stiffeners (used only if Stiffeners = sTransversal)*
double	**akr**	*critical load parameter factor (not used)*
double	**C1**	*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**	*Design approach*
double	**Ky**	*buckling length factor about local y axis (used only if BucklingLengthModeY = sblm_Factor)*
double	**Kz**	*buckling length factor about local z axis (used only if BucklingLengthModeZ = sblm_Factor)*
double	**kt**	*shear buckling coefficient (not used)*
double	**Kw**	*warping constraint factor (see steel design in Axis VM manual; used only if McrMethod <> mcrmAuto)*
EMcrMethod	**McrMethod**	*Define how to determine Mcr*
EStiffeners	**Stiffeners**	*type of stiffeners (used only if WebShearBuckling = lbTrue)*
long	**SDP_Class**	*section class (automatic if 0)*
ELongBoolean	**YBraced**	*True if braced Y direction*
ELongBoolean	**ZBraced**	*True if braced Z direction*
double	**Za**	*Relative load position in local z axis (see steel design parameters - load position in Axis VM manual)*
doube	**fse**	*load factor for seismic forces (default is 1 =>no change); note: it was ks before.*
ELongBoolean	**Cantilever**	*cantilever checkbox*
ESteelCantileverFixedEnd	**CantileverFixedEnd**	*start or end node of cantilever is fixed*
ELongBoolean	**FlexuralBuckling**	*flexural buckling*
ELongBoolean	**LateralTorsionalBuckling**	*lateral torsional buvkling*
ELongBoolean	**WebShearBuckling**	*web shear buckling*
ESteelBucklingLengthMode	**BucklingLengthModeY**	*buckling length calculation mode related to y-y axis*
ESteelBucklingLengthMode	**BucklingLengthModeZ**	*buckling length calculation mode related to z-z axis*
double	**Ly**	*buckling length related to y-y axis (used only if BucklingLengthModeY = sblm_Length)*
double	**Lz**	*buckling length related to z-z axis (used only if BucklingLengthModeZ = sblm_Length)*
ELongBoolean	**ConsiderN**	*consider the effect of normal force*
double	**Eta**	*η factor for web shear buckling checks (used only if Stiffeners = sTransversal)*
ESteelLateralSupports	**LateralSupports**	*lateral supports*
double	**Mcr**	*user defined Mcr (used only if LateralSupports = ammUserDefined)*
ELongBoolean	**FireResistDef**	*fire resistance design is active*

ELongBoolean	**fpOnlyPrescribed**	*check only prescribed failure modes*
double	**fpKy**	*flexural buckling factor about local y axis, in case of fire (used only if BucklingLengthModeY = sblm_Factor) [ ]*
double	**fpKz**	*flexural buckling factor about local z axis, in case of fire (used only if BucklingLengthModeZ = sblm_Factor) [ ]*
double	**fpKw**	*warping constraint factor, in case of fire(see steel design in Axis VM manual; used only if McrMethod <> mcrmAuto) [ ]*
double	**fpC1**	*steel design factor in case of fire, see steel design in Axis VM manual (depend on* **McrMethod** *and cross-section, see Calculation of Mcr in AxisVM manual*) [ ]
double	**fpC2**	*steel design factor in case of fire, see steel design in Axis VM manual (depend on* ***McrMethod*** *and cross-section, see Calculation of Mcr in AxisVM manual) [ ]*
double	**fpC3**	*steel design factor in case of fire, see steel design in Axis VM manual (depend on* ***McrMethod*** *and cross-section, see Calculation of Mcr in AxisVM manual) [ ]*
double	**fpLy**	*buckling length related to y-y axis, in case of fire (used only if BucklingLengthModeY = sblm_Length) [m]*
double	**fpLz**	*buckling length related to z-z axis, in case of fire (used only if BucklingLengthModeZ = sblm_Length) [m]*
double	**fpMcr**	*user defined Mcr (used only if LateralSupports = ammUserDefined) [kNm]*
ESteelFireParBetaMethod	**fpBetaMethod**	*handling of factors depending on the moment diagram*
double	**fpBetaMy**	*factor $\beta_{M,y}$ (used only if fpBetaMethod = sfpbm_User) [ ]*
double	**fpBetaMz**	*factor $\beta_{M,z}$ (used only if fpBetaMethod = sfpbm_User) [ ]*
double	**fpBetaLT**	*factor β_{LT} (used only if fpBetaMethod = sfpbm_User) [ ]*
double	**slsAngle**	*alfa value for checking inclined members [deg]*
ELongBoolean	**slsEyLimitDef**	*allowed deflection limit in y direction is active*
ELongBoolean	**slsEzLimitDef**	*allowed deflection limit in z direction is active*
ELongBoolean	**slsHxLimitDef**	*allowed horizontal displacement in x direction is active*
ELongBoolean	**slsHyLimitDef**	*allowed horizontal displacement in y direction is active*
ELongBoolean	**slsUyDef**	*pre-camber in y direction is active*
ELongBoolean	**slsUzDef**	*pre-camber in z direction is active*
ELongBoolean	**slsHGlob**	*horizontal displacement direction is global*
ESteelSLSHMethod	**slsHMode**	*calculation method of the height used for displacement cheks*
ESteelSLSEMethod	**slsEMode**	*deflection interpretation*
ESteelSLSLMethod	**slsLMode**	*member length interpretation*
ESteelSLSPreCamberCurve	**slsPreCamberCurve**	*type of the pre-camber curve*
double	**slsEyLimit**	*allowed deflection limit in y direction [m]*
double	**slsEzLimit**	*allowed deflection limit in z direction [m]*
double	**slsHxLimit**	*allowed horizontal displacement in x direction [m]*
double	**slsHyLimit**	*allowed horizontal displacement in y direction [m]*
double	**slsUy**	*pre-camber in y direction [m]*
double	**slsUz**	*pre-camber in z direction [m]*
double	**slsCustomLy**	*when slsLMode=sslslm_Custom,the user specified member length in y direction [m]*
double	**slsCustomLz**	*when slsLMode=sslslm_Custom,the user specified member length in z direction [m]*

double	**slsCustomH**	*h value when slsHMode=sslshm_Custom [m]*
double	**slsRatio)**	*position of the maximum value of pre-camber, given as a ratio [ ]*

RSteelDesignParameters_EC_SIA_ITA_V172 = (

ELongBoolean	**BreakAtElements**	*If Ttrue:* *Break members at side at side element connections or nodal supports.* 
		If False: *The member becomes only one structural element irrespective of other elements (emmbers etc.) connecting to its nodes.* 
double	**a**	*distance of trans. stiffeners (used only if Stiffeners = sTransversal)*
double	**akr**	*critical load parameter factor (not used)*
double	**C1**	*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**	*Design approach*
double	**Ky**	*buckling length factor about local y axis (used only if BucklingLengthModeY = sblm_Factor)*
double	**Kz**	*buckling length factor about local z axis (used only if BucklingLengthModeZ = sblm_Factor)*
double	**kt**	*shear buckling coefficient (not used)*
double	**Kw**	*warping constraint factor (see steel design in Axis VM manual; used only if McrMethod <> mcrmAuto)*
EMcrMethod	**McrMethod**	*Define how to determine Mcr*
EStiffeners	**Stiffeners**	*type of stiffeners (used only if WebShearBuckling = lbTrue)*
long	**SDP_Class**	*section class (automatic if 0)*
ELongBoolean	**YBraced**	*True if braced Y direction*
ELongBoolean	**ZBraced**	*True if braced Z direction*
double	**Za**	*Relative load position in local z axis (see steel design parameters - load position in Axis VM manual)*
doube	**fse**	*load factor for seismic forces (default is 1, 0 isn't a valid value); note: it was ks before.*
ELongBoolean	**Cantilever**	*cantilever checkbox*
ESteelCantileverFixedEnd	**CantileverFixedEnd**	*start or end node of cantilever is fixed*
ELongBoolean	**FlexuralBuckling**	*flexural buckling*
ELongBoolean	**LateralTorsionalBuckling**	*lateral torsional buvkling*
ELongBoolean	**WebShearBuckling**	*web shear buckling*
ESteelBucklingLengthMode	**BucklingLengthModeY**	*buckling length calculation mode related to y-y axis*
ESteelBucklingLengthMode	**BucklingLengthModeZ**	*buckling length calculation mode related to z-z axis*

double	**Ly**	*buckling length related to y-y axis (used only if BucklingLengthModeY = sblm_Length)*
double	**Lz**	*buckling length related to z-z axis (used only if BucklingLengthModeZ = sblm_Length)*
ELongBoolean	**ConsiderN**	*consider the effect of normal force*
double	**Eta**	*η factor for web shear buckling checks (used only if Stiffeners = sTransversal)*
ESteelLateralSupports	**LateralSupports**	*lateral supports*
double	**Mcr**	*user defined Mcr (used only if LateralSupports = ammUserDefined)*
ELongBoolean	**BucklLocXZGeom**	*taking into account buckling in local XZ plane by running nonlinear analysis on imperfect geometry*
ELongBoolean	**BucklLocXYGeom**	*taking into account buckling in local XY plane by running nonlinear analysis on imperfect geometry*
ELongBoolean	**LTBWarp**	*take into account warping torsion (only for 7DOF elements)*
ESteelLoadPosition	**LoadPosition**	*load positioning on the cross-section*
ESteelBucklingCurves	**BucklingCurvesY**	*buckling curve about the y axis*
ESteelBucklingCurves	**BucklingCurvesZ**	*buckling curve about the z axis*
ESteelLTBucklingCurves	**LTBucklingCurves**	*lateral torsional buckling curves*
ELongBoolean	**FireResistDef**	*fire resistance design is active*
ELongBoolean	**fpOnlyPrescribed**	*check only prescribed failure modes*
double	**fpKy**	*flexural buckling factor about local y axis, in case of fire (used only if BucklingLengthModeY = sblm_Factor) [ ]*
double	**fpKz**	*flexural buckling factor about local z axis, in case of fire (used only if BucklingLengthModeZ = sblm_Factor) [ ]*
double	**fpKw**	*warping constraint factor, in case of fire(see steel design in Axis VM manual; used only if McrMethod <> mcrmAuto) [ ]*
double	**fpC1**	*steel design factor in case of fire, see steel design in Axis VM manual (depend on* **McrMethod** *and cross-section, see Calculation of Mcr in AxisVM manual*) [ ]
double	**fpC2**	*steel design factor in case of fire, see steel design in Axis VM manual (depend on* ***McrMethod*** *and cross-section, see Calculation of Mcr in AxisVM manual) [ ]*
double	**fpC3**	*steel design factor in case of fire, see steel design in Axis VM manual (depend on* ***McrMethod*** *and cross-section, see Calculation of Mcr in AxisVM manual) [ ]*
double	**fpLy**	*buckling length related to y-y axis, in case of fire (used only if BucklingLengthModeY = sblm_Length) [m]*
double	**fpLz**	*buckling length related to z-z axis, in case of fire (used only if BucklingLengthModeZ = sblm_Length) [m]*
double	**fpMcr**	*user defined Mcr (used only if LateralSupports = ammUserDefined) [kNm]*
ESteelFireParBetaMethod	**fpBetaMethod**	*handling of factors depending on the moment diagram*
double	**fpBetaMy**	*factor $\beta_{M,y}$ (used only if fpBetaMethod = sfpbm_User) [ ]*
double	**fpBetaMz**	*factor $\beta_{M,z}$ (used only if fpBetaMethod = sfpbm_User) [ ]*
double	**fpBetaLT**	*factor β_{LT} (used only if fpBetaMethod = sfpbm_User) [ ]*
double	**slsAngle**	*alfa value for checking inclined members [deg]*
ELongBoolean	**slsEyLimitDef**	*allowed deflection limit in y direction is active*
ELongBoolean	**slsEzLimitDef**	*allowed deflection limit in z direction is active*

ELongBoolean	**slsHxLimitDef**	*allowed horizontal displacement in x direction is active*
ELongBoolean	**slsHyLimitDef**	*allowed horizontal displacement in y direction is active*
ELongBoolean	**slsUyDef**	*pre-camber in y direction is active*
ELongBoolean	**slsUzDef**	*pre-camber in z direction is active*
ELongBoolean	**slsHGlob**	*horizontal displacement direction is global*
ESteelSLSHMethod	**slsHMode**	*calculation method of the height used for displacement cheks*
ESteelSLSEMethod	**slsEMode**	*deflection interpretation*
ESteelSLSLMethod	**slsLMode**	*member length interpretation*
ESteelSLSPreCamberCurve	**slsPreCamberCurve**	*type of the pre-camber curve*
double	**slsEyLimit**	*allowed deflection limit in y direction [m]*
double	**slsEzLimit**	*allowed deflection limit in z direction [m]*
double	**slsHxLimit**	*allowed horizontal displacement in x direction [m]*
double	**slsHyLimit**	*allowed horizontal displacement in y direction [m]*
double	**slsUy**	*pre-camber in y direction [m]*
double	**slsUz**	*pre-camber in z direction [m]*
double	**slsCustomLy**	*when slsLMode=sslslm_Custom,the user specified member length in y direction [m]*
double	**slsCustomLz**	*when slsLMode=sslslm_Custom,the user specified member length in z direction [m]*
double	**slsCustomH**	*h value when slsHMode=sslshm_Custom [m]*
double	**slsRatio)**	*position of the maximum value of pre-camber, given as a ratio [ ]*

RSteelDesignParameters_MSZ_STAS = (

ELongBoolean	**BreakAtElements**	*See here*
double	**nuy**	*see v_y in steel design in Axis VM manual*
double	**nuz**	*see v_z in steel design in Axis VM manual*
double	**nuw**	*see v_w in steel design in Axis VM manual*
double	**d**	*see steel design in Axis VM manual [m]*
double	**a**	*distance of trans. stiffeners [m]*
EStiffeners	**Stiffeners**)	*type of stiffeners*

Steel design parameters in accordance with Hungarian and Romanian national codes.

RSteelDesignParameters_NEN = (

ELongBoolean	**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	**lgF**	*see steel design in Axis VM manual*
double	**lgA**	*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]*
ELongBoolean	**YBraced**	*see steel design in Axis VM manual*
ELongBoolean	**ZBraced**	*see steel design in Axis VM manual*
ETorsion	**Torsion**	*type of torsion*

)

Steel design parameters in accordance with Dutch national code.

RSteelLTBSupport= (

double	**AbsPos**	*absolute position of the support*
double	**Ecc**	*eccentricity*
double	**Ry**	*lateral support stiffness in local y direction*
double	**Rxx**	*torsional stiffness about the member's axis*

double **Rzz** *torsional stiffness about the local z axis*
double **Rw** *warping stiffness*

)

RSteelDesignParameters = (

Warning! *This record has become obsolete, it was superseded by RSteelDesignParameters_V153*

RSteelDesignParametersMSZ_STAS **MSZ_STAS** *design parameters according to MSZ and STAS national design code*
RSteelDesignParameters_EC_SIA_ITA **EC_SIA_ITA** *design parameters according to EC,SIA and ITA national design code*
RSteelDesignParametersNEN **NEN** *design parameters according to NEN national design code*
)

RSteelDesignParameters_V153 = (***Warning!*** *This record has become obsolete, it was superseded by RSteelDesignParameters_V172*

RSteelDesignParametersMSZ_STAS **MSZ_STAS** *design parameters according to MSZ and STAS national design code*
RSteelDesignParameters_EC_SIA_ITA_V153 **EC_SIA_ITA** *design parameters according to EC,SIA and ITA national design code*
RSteelDesignParametersNEN **NEN** *design parameters according to NEN national design code*
)
only the field according to the current national design code will be active/taken into account

RSteelDesignParameters_V172 = (

RSteelDesignParametersMSZ_STAS **MSZ_STAS** *design parameters according to MSZ and STAS national design code*
RSteelDesignParameters_EC_SIA_ITA_V172 **EC_SIA_ITA** *design parameters according to EC,SIA and ITA national design code*
RSteelDesignParametersNEN **NEN** *design parameters according to NEN national design code*
)
only the field according to the current national design code will be active/taken into account

Functions

long **Add** ([in] SAFEARRAY(byte) **DesignParameters**, [in] SAFEARRAY(long)* **LineIds**)

Warning! *This function has become obsolete, was superseded by Add_V153*

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)*
LineIds Index array (long) with lineIDs

Defines a new steel design member.
If successful, returns the SteelDesignMember index of new SteelDesignMember otherwise an error code (errDatabaseNotReady, errIndexOutOfBounds, sdmeLineListIsEmpty, sdmeNotConnectingLines, sdmeCOMError).

---

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

---

long **Add_V153** ([i/o] RSteelDesignParameters_V153 **SteelDesignParameters**, [in] SAFEARRAY(long)* **LineIds**)

Warning! *This function has become obsolete, was superseded by Add_V172*

SteelDesignParameters *Contains all design parameters in the field corresponding to the current national design code. The other national design code fields are ignored.*
LineIds Index array (long) with lineIDs

Defines a new steel design member.
If successful, returns the SteelDesignMember index of new SteelDesignMember otherwise an error code (errDatabaseNotReady, errIndexOutOfBounds, sdmeLineListIsEmpty, sdmeNotConnectingLines, sdmeCOMError).

---

long **Add_V172** ([i/o] RSteelDesignParameters_V172 **SteelDesignParameters**,
[in] SAFEARRAY(long)* **LineIds**)

SteelDesignParameters	*Contains all design parameters in the field corresponding to the current national design code. The other national design code fields are ignored.*
LineIds	Index array (long) with lineIDs

Defines a new steel design member.
If successful, returns the SteelDesignMember index of new SteelDesignMember otherwise an error code (errDatabaseNotReady, errIndexOutOfBounds, sdmeLineListIsEmpty, sdmeNotConnectingLines, sdmeCOMError).

---

long **Add2** ([i/o] RSteelDesignParameters **SteelDesignParameters**,
[in] SAFEARRAY(long)* **LineIds**)

Warning! *This function has become obsolete, was superseded by Add_V153*

SteelDesignParameters	*Parameter which contain all design parameters from all national design codes*
LineIds	Index array (long) with lineIDs

Defines a new steel design member.
If successful, returns the SteelDesignMember index of new SteelDesignMember otherwise an error code (errDatabaseNotReady, errIndexOutOfBounds, sdmeLineListIsEmpty, sdmeNotConnectingLines, sdmeCOMError).

---

long **Add2_vb** (Visual Basic compatible function of **Add2**)

---

long **Add3** ([i/o] SAFEARRAY(byte) **SteelDesignParameters**,
[in] SAFEARRAY(long)* **LineIds**)

Warning! *This function has become obsolete, was superseded by Add_V153*

SteelDesignParameters	*Parameters which contain all design parameters of used national design code*
LineIds	Index array (long) with lineIDs

Defines a new steel design member. If successful, returns the SteelDesignMember index of new SteelDesignMember otherwise an error code (errDatabaseNotReady, errIndexOutOfBounds, sdmeLineListIsEmpty, sdmeNotConnectingLines, sdmeCOMError).

Note:Only the field corresponding to the active national design code will be taken into account from SteelDesignParameters

---

long **BulkAdd** ([i/o] SAFEARRAY(RSteelDesignParameters_V153)* **SteelDesignParameters**,
[i/o] SAFEARRAY(VARIANT)* **LineIds**)

Warning! *This function has become obsolete, was superseded by BulkAdd_V172*

SteelDesignParameters	*Array of parameters which contain all design parameters according to the active national design code*
LineIds	Array of VARIANTs, each VARIANT being a SAFEARRAY(long) with lineIDs

Defines multiple new steel design members. The length of SteelDesignParameters should be the same as the length of LineIds. If successful, returns the SteelDesignMember index of the lastly created SteelDesignMember otherwise an error code (errDatabaseNotReady, errIndexOutOfBounds, sdmeLineListIsEmpty, sdmeNotConnectingLines, sdmeCOMError, sdmeArrayLengthMismatch).

Note:Only the field corresponding to the active national design code will be taken into account from SteelDesignParameters

---

long **BulkAdd_V172** ([i/o] SAFEARRAY(RSteelDesignParameters_V172)* **SteelDesignParameters**,
[i/o] SAFEARRAY(VARIANT)* **LineIds**)

SteelDesignParameters	*Array of parameters which contain all design parameters according to the active national design code*
LineIds	Array of VARIANTs, each VARIANT being a SAFEARRAY(long) with lineIDs

Defines multiple new steel design members. The length of SteelDesignParameters should be the same as the length of LineIds. If successful, returns the SteelDesignMember index of the lastly created SteelDesignMember otherwise an error code (errDatabaseNotReady, errIndexOutOfBounds, sdmeLineListIsEmpty, sdmeNotConnectingLines, sdmeCOMError, sdmeArrayLengthMismatch).

Note:Only the field corresponding to the active national design code will be taken into account from SteelDesignParameters

---

long **Clear**

Removes all SteelDesignMembers. It returns the number of deleted SteelDesignMembers. If it returns a negative number, that is an error code (errDatabaseNotReady).

---

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

Index	*SteelDesignMember index (0 < Index ≤ IAxisVMSteelDesignMembers.Count)*

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

---

long **DeleteSelected**

Returns number of deleted SteelDesignMembers, otherwise returns an error code (errDatabaseNotReady, errNotSupportedByNationalDesignCode).

---

long **GetDesignParameters** ([in] long **Index,** [out] SAFEARRAY(byte)* **DesignParameters**)

Warning! *This function has become obsolete, was superseded by GetDesignParameters_V153*

Index	*SteelDesignMember index (0 < Index ≤ IAxisVMSteelDesignMembers.Count)*
DesignParameters	*See IAxisVMSteelDesignMembers.Add*

If successful, returns SteelDesignMember index otherwise an error code (errDatabaseNotReady, errIndexOutOfBounds, errNotSupportedByNationalDesignCode, sdmeInvalidNationalDesignCode)

---

long **GetDesignParameters_V153** ([in] long **Index,** [i/o] RSteelDesignParameters_V153 **SteelDesignParameters**)

Warning! *This function has become obsolete, was superseded by GetDesignParameters_V172*

Index	*SteelDesignMember index (0 < Index ≤ IAxisVMSteelDesignMembers.Count)*
DesignParameters	*See Add_V153*

If successful, returns SteelDesignMember index otherwise an error code (errDatabaseNotReady, errIndexOutOfBounds, errNotSupportedByNationalDesignCode, sdmeInvalidNationalDesignCode)

---

long **GetDesignParameters_V172** ([in] long **Index,** [i/o] RSteelDesignParameters_V172 **SteelDesignParameters**)

Index	*SteelDesignMember index (0 < Index ≤ IAxisVMSteelDesignMembers.Count)*
DesignParameters	*Contains all design parameters in the field corresponding to the current national design code. The other national design code fields are ignored*

If successful, returns SteelDesignMember index otherwise an error code (errDatabaseNotReady, errIndexOutOfBounds, errNotSupportedByNationalDesignCode, sdmeInvalidNationalDesignCode)

---

long **GetDesignParameters2** ([in] long **Index,**
[i/o] RSteelDesignParameters **SteelDesignParameters**)

Warning! *This function has become obsolete, was superseded by GetDesignParameters_V153*

Index	*SteelDesignMember index (0 < Index ≤ IAxisVMSteelDesignMembers.Count)*
SteelDesignParameters	*Parameters which contain all design parameters from all national design codes*

If successful, returns index otherwise an error code (errDatabaseNotReady, errIndexOutOfBounds, errNotSupportedByNationalDesignCode, sdmeInvalidNationalDesignCode)

---

long **GetDesignParameters3** ([in] long **Index,**

[out] SAFEARRAY(byte) **SteelDesignParameters**)

Warning! *This function has become obsolete, was superseded by GetDesignParameters_V153*

Index	*SteelDesignMember index (0 < Index ≤ 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)*

If successful it returns index otherwise an error code (errDatabaseNotReady, errIndexOutOfBounds, errNotSupportedByNationalDesignCode, sdmeInvalidNationalDesignCode)

Note: The SteelDesignParameters must be type casted into the appropriate load record type depending on the national design code!

---

long **GetLineIds (**[in] long **Index**, [out] SAFEARRAY(long)*** LineIds)**

Index	*SteelDesignMember index (0 < Index ≤ IAxisVMSteelDesignMembers.Count)*
LineIds	Index array (long) with lineIDs

Returns number of lines in SteelDesignMember, otherwise returns an error code (errDatabaseNotReady, errNotSupportedByNationalDesignCode).

---

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

ItemIds	*Index list of selected edge connections*

Returns number of selected SteelDesignMembers, otherwise returns an error code (errDatabaseNotReady, errNotSupportedByNationalDesignCode).

---

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

Select	*selection state*

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

NOTE: Call Refresh function afterwards if not called between functions BeginUpdate and EndUpdate

---

long **SetDesignParameters** ([in] long **Index,** [out] SAFEARRAY(byte)* **DesignParameters**)

Warning! *This function has become obsolete, was superseded by SetDesignParameters_V153*

Index	*SteelDesignMember index (0 < Index ≤ 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

If successful returns SteelDesignMember index, otherwise an error code (errDatabaseNotReady, errIndexOutOfBounds, errNotSupportedByNationalDesignCode, sdmeInvalidNationalDesignCode)

---

long **SetDesignParameters_V153** ([in] long **Index,**
[i/o] RSteelDesignParameters_V153 **SteelDesignParameters**)

Warning! *This function has become obsolete, was superseded by SetDesignParameters_V172*

Index	*SteelDesignMember index (0 < Index ≤ IAxisVMSteelDesignMembers.Count)*
SteelDesignParameters	*See Add_V153*

If successful returns index, otherwise an error code (errDatabaseNotReady, errIndexOutOfBounds, errNotSupportedByNationalDesignCode, sdmeInvalidNationalDesignCode)

---

long **SetDesignParameters_V172** ([in] long **Index,**
[i/o] RSteelDesignParameters_V172 **SteelDesignParameters**)

Index	*SteelDesignMember index (0 < Index ≤ IAxisVMSteelDesignMembers.Count)*
SteelDesignParameters	*Contains all design parameters in the field corresponding to the current national design code. The other national design code fields are ignored.*

If successful returns index, otherwise an error code (errDatabaseNotReady, errIndexOutOfBounds, errNotSupportedByNationalDesignCode, sdmeInvalidNationalDesignCode)

---

long **SetDesignParameters2** ([in] long **Index,**
[i/o] RSteelDesignParameters **SteelDesignParameters**)

Warning! *This function has become obsolete, was superseded by SetDesignParameters_V153*

Index	*SteelDesignMember index (0 < Index ≤ IAxisVMSteelDesignMembers.Count)*
SteelDesignParameters	*Parameter which contain all design parameters from all national design codes*

If successful returns index, otherwise an error code (errDatabaseNotReady, errIndexOutOfBounds, errNotSupportedByNationalDesignCode, sdmeInvalidNationalDesignCode)

---

long **SetDesignParameters3** ([in] long **Index,**
[i/o] SAFEARRAY(byte) **SteelDesignParameters**)

Warning! *This function has become obsolete, was superseded by SetDesignParameters_V153*

Index	*SteelDesignMember index (0 < Index ≤ 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)*

If successful returns index, otherwise an error code (errDatabaseNotReady, errIndexOutOfBounds, errNotSupportedByNationalDesignCode, sdmeInvalidNationalDesignCode)

Note: The SteelDesignParameters must be type casted into the appropriate load record type depending on the national design code!

---

long **GetLateralSupports (**[in] long **Index**,
[out] SAFEARRAY(RSteelLTBSupport)*** LateralSupports)**

Index	*SteelDesignMember index (0 < Index ≤ IAxisVMSteelDesignMembers.Count)*
LateralSupports	*lateral supports' settings*

Get lateral supports' parameters. If successful returns design member's index, otherwise an error code (errDatabaseNotReady, errIndexOutOfBounds, errNotSupportedByNationalDesignCode, sdmeInvalidNationalDesignCode)

---

long **SetLateralSupports (**[in] long **Index**,
[i/o] SAFEARRAY(RSteelLTBSupport)*** LateralSupports)**

Index *SteelDesignMember index (0 < Index ≤ IAxisVMSteelDesignMembers.Count)*

LateralSupports *lateral supports' settings*

Sets lateral supports' parameters. If successful returns design member's index, otherwise an error code (errDatabaseNotReady, errIndexOutOfBounds, errNotSupportedByNationalDesignCode, sdmeInvalidNationalDesignCode)

Properties

long **Count** *Get number of SteelDesignMembers.*

long **CrossSetionID** • *Get or set considered cross-section index, 0 for original (default)*

long **Length** [long **Index**] *Get length of a SteelDesignMember*

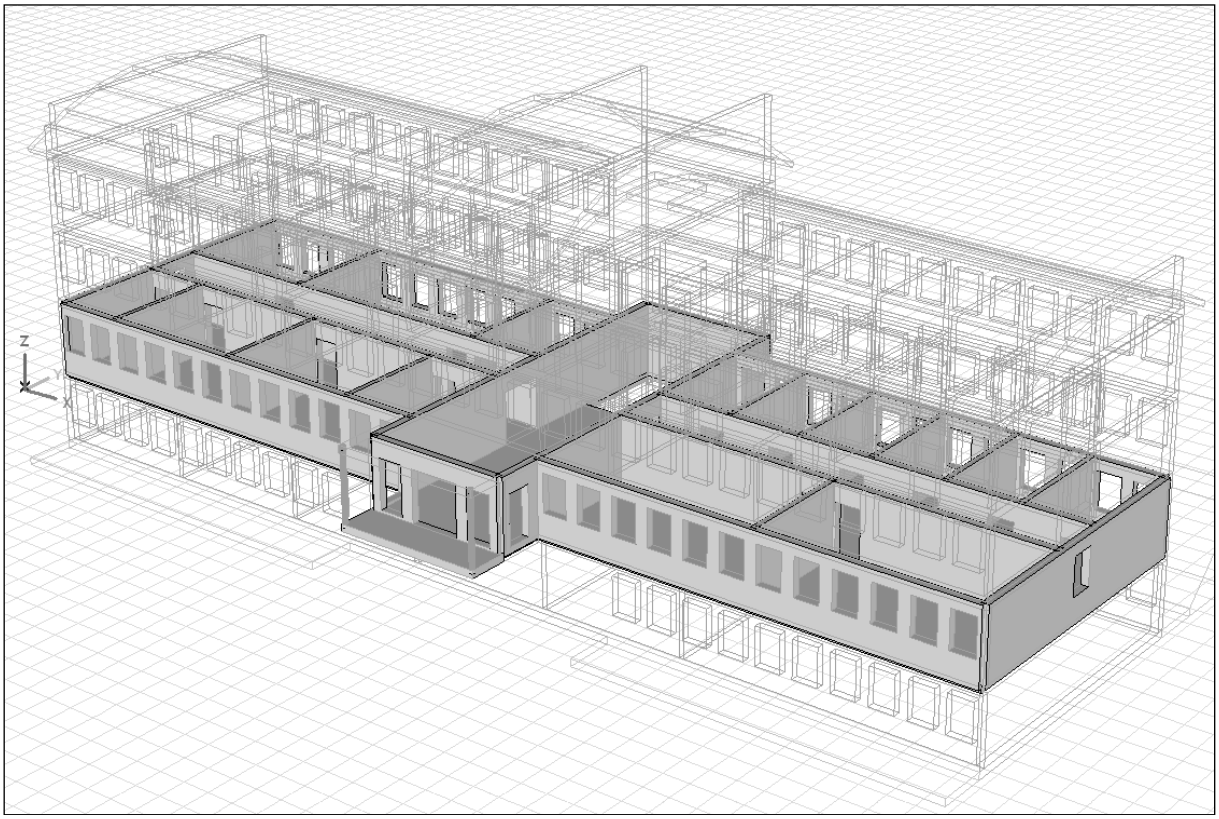
ELongBoolean **Selected** [long **Index**] • *Get or set the selection status of a SteelDesignMember*

NOTE: 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)**

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*

Get all properties of the storey. If successful, returns index of storey, otherwise an error code (errDatabaseNotReady or errIndexOutOfBounds).

---

long **IndexOfZ** ([in] double **z)**

z	*z coordinate of the storey*

If successful, returns storey index, otherwise an error code (errDatabaseNotReady or errNotFound).

---

long **SetItem** ([in] long **index,** [in] BSTR **Name,** [in] double **z)**

index	*index of the storey*
Name	*name of the storey*
z	*Z coordinate of the storey*

Set all properties of the storey. If successful, returns index of storey, otherwise an error code (errDatabaseNotReady or errIndexOutOfBounds).

---

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)**

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*

Get all properties of the seismic storey. If successful, returns index of seismic storey, otherwise an error code (errDatabaseNotReady or errIndexOutOfBounds).

---

long **IndexOfZ** ([in] double **z)**

z	*z coordinate of the seismic storey*

If successful, returns storey index, otherwise an error code (errDatabaseNotReady or errNotFound).

---

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	**AutoSearchStyle** • *Get or set the auto search style for storey search.*
long	**Count** *Get number of storeys.*
ElongBoolean	**HasEmptyStoreys** *Returns lbTrue 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**] *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).*
long	**LevelZ** [long **Index**] • *Get or set z of the storey with index Index. If successful, returns number 1,2,…, otherwise 0.*
BSTR	**Name** [long **Index**] • *Get or set name of the storey with index* **Index**.

IAxisVMStructuralGrids

Structural grids in the model

Enumerated types

enum **EShowStructuralGridLineTitle**= { — *position of the title of structural grid's line*
- **ssgltStart** = 0x00,
- **ssgltEnd** = 0x01,
- **ssgltBoth** = 0x02}

enum **EStructuralGridPlane**= { — *plane of the structural grid*
- **sgp_XY** = 0x00, — *XY plane*
- **sgp_XZ** = 0x01, — *XZ plane*
- **sgp_YZ** = 0x02 — *YZ plane*
- **sgp_WorkPlane** = 0x03 — *workplane*
- **sgp_Story** = 0x04} — *story's plane*

enum **EStructuralGridVisibility**= { — *structural grids' visibility*
- **sgvDefault** = 0x00, — *default*
- **sgvVisibleAtAllStories** = 0x01, — *visible at all stories*
- **sgvOnlyIfActive** = 0x02} — *only if structural grid is active*

enum **EStructuralGridLabelType**= { — *structural grid's label type*
- **sgltLetters**= 0x00, — *letters*
- **sgltNumbers**= 0x01} — *numbers*

Error codes

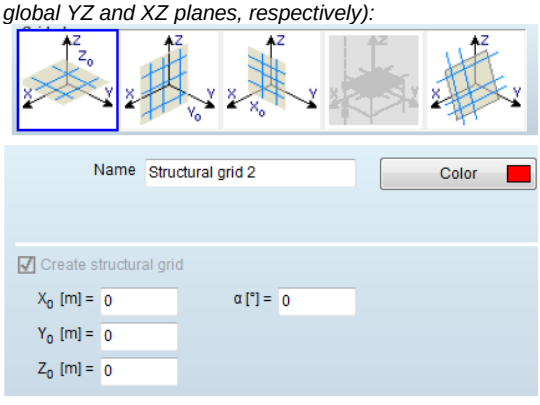
enum **EStructuralGridsError**= {

sgeInvalidName= –100001 ,	*structural grid's name is invalid*
sgeInvalidWorkPlaneindex= –100002 ,	*workplane's index is invalid*
sgeInvalidStoreyIndex= –100003 ,	*storey's index is invalid*
sgwInvalidStartCharX= -100004,	*StartCharX is invalid*
sgwInvalidStartCharY= -100005,	*StartCharY is invalid*
sqwNormalVectorVaries= -100006,	*the normal vector varies*
sqwGridLineNotInPlane= -100007}	*the grid line is not in plane*

Records / structures

RStructuralGridParams = (

EStructuralGridPlane	**Plane**	*plane of the structural grid*
long	**WorkPlaneOrStoreyIndex**	*index of workplane or storey*
double	**PlaneOffset**	*distance between the plane and the grid* *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 gird is the global XY plane, the PlaneOffset is equal to Z_0 in the software (or X_0, Y_0 related to global YZ and XZ planes, respectively):*



EStructuralGridVisibility	**Visibility**)	*visibility*
	RStructuralGridGenerationParams= (	
Rpoint2d	**Offset**	*local offset from origin*
double	**RotDeg**	*rotation angle in degrees*
long	**Colour**	*colour*
double	**Extension**	*the distance of the label from structural gird's edge*
EStructuralGridLabelType	**LabelTypeX**	*label type in X direction*
ELongBoolean	**GenerateInPositiveX**	*generate grids in positive direction*
EStructuralGridLabelType	**LabelTypeY**	*label type in Y direction*
ELongBoolean	**GenerateInPositiveY**	*generate grids in positive direction*
EShowStructuralGridLineTitle	**ShowStructuralGridLineTitle**	*show grid line's title*
	)	

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*

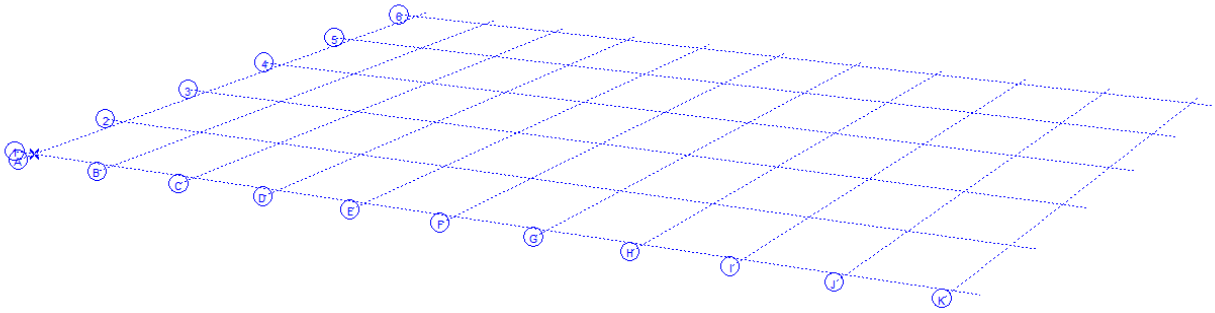
Sets structural grid's parameters. If successful, returns structural grid's index, otherwise an error code (EStructuralGridsError).

Properties

long	**Count** *Get number of structural grids*
AxisVMStructuralGrid*	**Item** [long **Index**] *Get structural grid by Index*
	Index *index of structural grid*
BSTR	**Name•** [long **Index**] *Get or set structural grid' name*
	Index *index of structural grid*

IAxisVMStructuralGrid

AxisVM StructuralGrid



Enumerated types

enum	**EStructuralGridPlane**= {	*plane of the structural grid*
	sgp_XY = 0x00,	*XY plane*
	sgp_XZ = 0x01,	*XZ plane*
	sgp_YZ = 0x02	*YZ plane*
	sgp_WorkPlane = 0x03	*workplane*
	sgp_Story = 0x04}	*story's plane*
enum	**EGridLineSpacingDirection**= {	*spacing direction of the grid line*
	glsd_X= 0x00,	*local X of the grid*
	glsd_Y= 0x01	*local Y of the grid*
	glsd_Other= 0x02}	*other*

Error codes

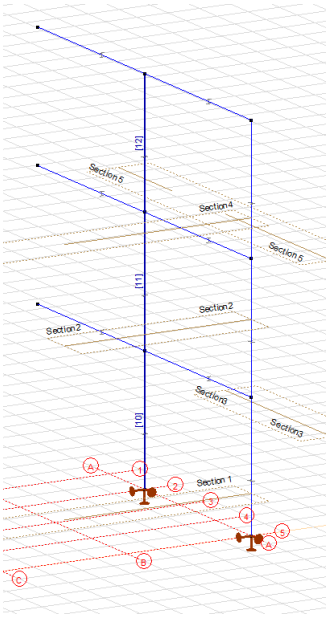
see AxisVMStructuralGrids' error codes.

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 gird's edge*
EShowStructuralGridLineTitle	**ShowTitle**	*show title*
ELongBoolean	**ToLogicalPart**	*the structural grid's line can be added to logical parts (note: see the figure below)*
double	**PlaneTolerance**	*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**)

Title *title of the new structural grid line*

StructuralGridParams *structural grid line's parameters*

Adds a new structural grid line. If successful, returns structural grid line's index, otherwise an error code (EStructuralGridsError).

---

long **GetLine** ([in] long **Index,** [out] BSTR **Title,**
[i/o] RStructuralGridLineParams **StructuralGridLineParams**)

Index *Structural grid line's index*

Title *title of the new structural grid line*

StructuralGridParams *structural grid line's parameters*

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).

---

long **SetLine** ([in] long **Index,** [in] BSTR **Title,**
[i/o] RStructuralGridLineParams **StructuralGridLineParams**)

Index *Structural grid line's index*

Title *title of the new structural grid line*

StructuralGridParams *structural grid line's parameters*

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).

---

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

Index *Structural grid line's index*

Deletes a structural grid line by Id. If successful, returns structural grid line's index, otherwise an error code (EStructuralGridsError).

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*

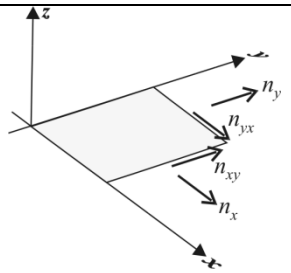
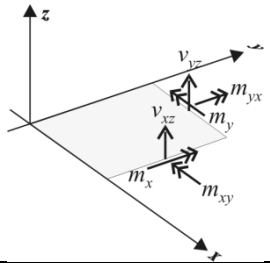
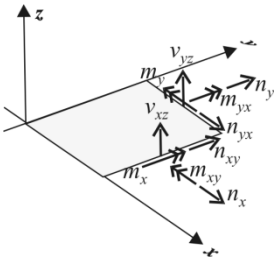
IAxisVMSurfaces

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 = 0x1,	*membrane (plane stress)*
stMembraneStrain = 0x2,	 *Membrane (plane strain)*
stPlate = 0x3,	*Plate* 
stShell = 0x4 }	*Shell* 

Types of surface elements.

enum **ESurfaceStiffnessReduction** = {

ssr_AC = 0,	*stiffness reduction factor components* *cross-section area component*
ssr_ACS = 1,	*shear cross-section area component*
ssr_IC = 2}	*bending inertia of the cross-section component*

Error codes

enum **ESurfacesError** = {

seLineCountCanBeOnly3Or4 = –100001 ,	*a surface can have 3 or 4 edges only*
seCannotModify = –100002 ,	*cannot modify surface element properties*
seCOMError = –100003 ,	*internal COM server error*
seReinforcementParametersNotExists = -100004,	*reinforcement parameters do not exists*

seConcreteIdIndexOutOfBounds = -100005,	*Concrete's material index is out of bounds*
seRebarSteelGradeIdIndexOutOfBounds = -100006,	*rebar's material index index is out of bounds*
seThicknessMustBePositive = -100007,	*domain thickness must be a positive number*
seRebarPosMustBePositive = -100008,	*rebar position must be a positive number*
sePhiMustBePositiveOrZero = -100009,	*Phi must be a positive number*
seNuMustBePositiveOrZero = -100010,	*Nu must be a positive number*
seTauaMustBePositiveOrZero = -100011,	*Tau_a must be a positive number*
seAggregateSizeMustBePositive = -100012	*aggregate size must be a positive number*
sePropertyNotValidForThisSurfaceType= -100013	*StiffnessReduction property can return this error*
sefseMustBePositive = -100014 ,	*seismic load factor fse must be a positive value*
seParametersRecordNotValidForUsedDesignCode = -100015 ,	*used parameter record type is not valid for current design code*
seConcreteCoverMustBePositive = -100016 ,	*concrete cover must be a positive value*
seRebarDiameterMustBePositive= -100017 ,	*Rebar diameter must be a positive value*
seEnvironmentClassNotValidForUsedDesignCode = -100018 ,	*EnviromentClass is not valid for selected design code*
seAlphaVRdmaxIsInvalid = - 100019 ,	*the angle of shear reinforcement is invalid*
seThetaVRdmaxIsInvalid = - 100020 ,	*he angle of shear crack is invalid*
seShrinkageEpsMustBePositive = - 100021 ,	*shrinkage strain must be positive*
seRCNonlinearSurfTypeIsInvalid = - 100022 ,	*nonlinear surface type is invalid*
seAlphaAngleIsInvalid = - 100023 ,	*the angle of ξ reinforcement direction is invalid*
seBetaAngleIsInvalid = - 100024,	*the angle between ξ and η reinf. directions is invalid*
se_k_torsionIsInvalid = - 100025 ,	*k_torsion should be <= 0.1 and < 1*
se_k_shearIsInvalid = - 100026 ,	*k_shear should be <= 0.1 and < 1*
se_k_bendingIsInvalid = - 100027,	*k_bending should be <= 0.1 and < 1*
seLimitingCrackWidthIsInvalid = - 100028,	*the value for limiting crack width is invalid*
seMaterialIndex = - 100029,	*MaterialIndex must be : 1 ≤ MaterialIndex ≤ AxisVMMaterials.Count*
seSurfaceReferenceIndexOutOfBounds = - 100030,	*reference index must be : 0 ≤ ReferenceIndex ≤ AxisVMReferences.Count*
seInvalidType = - 100031,	*an enumerated type hase violated the valid range for that type*
seElasticFoundationNegative = - 100032,	*Elastic foundation cannot be negative*
seInvalidCharacteristics = -100033,	*Characteristics is out of valid range*
seInvalidStiffnessReduction = -100034,	*stiffness reduction should be > 0.00 and <= 1.00*
seStiffnessReductionNotAllowed = -100035,	*design code doesn't allow for stiffness reduction*
seInvalidStiffnessReductionMat = -100036,	*surface's material doesn't allow for stiffness reduction*
seInvalidReinfParamForTrapezoidal = -100037 }	*reinforcement parameters are incompatible with the geometry of the trapezoidal domain*

Records / structures

RElasticFoundationXYZ = (

double	**x, y, z**	*elastic foundation stiffness in x, y, z directions [kN/m/m²]*

)

RNonLinearityXYZ = (

ELineNonLinearity	**x, y, z**	*nonlinear behaviour in x, y, z directions*

)

RResistancesXYZ = (

Double	**x, y, z**	*resistance in x, y, z directions [kN/m²]*

)

RSurface = (

long	**N**	*Number of contour lines (valid values are either 3 or 4). LineIndexes must be in the geometrically correct continuous sequence*
long	**LineIndex1**	*first contour line index*
long	**LineIndex2**	*second contour line index*
long	**LineIndex3**	*third contour line index*
long	**LineIndex4**	*fourth contour line index. Only required if N=4*
RSurfaceAttr	**Attr**	*surface attributes*
long	**DomainIndex**	*index of the domain containing this surface, 0 for none 0 ≤ DomainIndex ≤ AxisVMDomains.Count*

)

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 ≤ DomainId ≤ 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 or 4
LineIndex1 : 1 ≤ LineIndex1 ≤ AxisVMLines.Count
LineIndex2 : 1 ≤ LineIndex2 ≤ AxisVMLines.Count
LineIndex3 : 1 ≤ LineIndex3 ≤ AxisVMLines.Count
LineIndex4 : if N=3, LineIndex4=0, if N=4, 1 ≤ LineIndex4 ≤ AxisVMLines.Count
DomainIndex : domain index if the surface element is part of a mesh otherwise 0, 0 ≤ DomainId ≤ 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 ≤ Index ≤ 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 ≤ Index ≤ 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 ≤ Index ≤ 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**

If successful, returns 1. Otherwise returns error code(errDatabaseNotReady)

---

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

ItemIds *list of selected surfaces*

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

---

long **GetAllCoordinatesOfSurfaces** ([in] SAFEARRAY (long) * **ItemIds,**

[out] **SAFEARRAY(**RSurfaceCoordinates**) * SurfacesCoordinates**)

ItemIds *list of surface indexes*

SurfacesCoordinates *Surface coordinates of surfaces listed in ItemIds*

If successful, returns the number of elements in ItemIds, otherwise returns an error code(errDatabaseNotReady, errIndexOutOfBounds)

---

long **GetAllCoordinatesOfSurfaces_vb** (Visual Basic compatible function of **GetAllCoordinatesOfSurfaces**)

---

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

Select *selection state*

If Select = True, selects all surface elements.
If Select = False, deselects all surface elements.

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

NOTE: Call Refresh function afterwards if not called between functions BeginUpdate and EndUpdate

---

Properties

AxisVMAttachments* **Attachments** *Get the attachments interface*

AxisVMAttributes* **Attributes** *Get the attributes interface*

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

AxisVMSurface* **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*

ELongBoolean **Selected** [long **Index**] • *Get or set the selection status of a surface element*

Index *index of the surface*

NOTE: Call Refresh function afterwards if not called between functions BeginUpdate and EndUpdate

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

double **StiffnessReduction[**long **Index]** •

Warning! *This function was extended by StiffnessReduction_V153.Get or set stiffness reduction for the cross-section area (Ac). Default value is 1. Used only for atLinearStatic and atLinearVibration analysis types. (only valid for Eurocode, SIA, NTS standards)*

double **StiffnessReduction_V153 [**long **Index,** ESurfaceStiffnessReduction **Component]** •

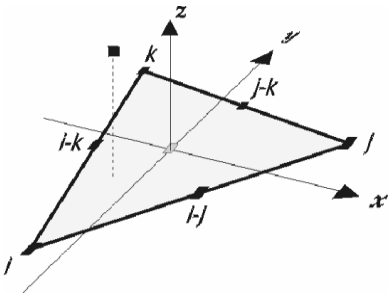
Get or set stiffness reduction for the given component. Default value is 1. Used only for atLinearStatic and atLinearVibration analysis types. (only valid for Eurocode, SIA, NTS standards)

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**

)

RNonLinearityXYZ = (

ELineNonLinearity	**x, y, z**	*nonlinear behaviour in x, y, z directions*

)

RResistancesXYZ = (

double	**x, y, z**	*resistance in x, y, z directions [kN/m²]*

)

RSurfaceAttr = (

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

)

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*
RPoint3d	**pContourPoint1**	*Coordinates of contour point 1*
RPoint3d	**pContourPoint2**	*Coordinates of contour point 2*
RPoint3d	**pContourPoint3**	*Coordinates of contour point 3*
RPoint3d	**pContourPoint4**	*Coordinates of contour point 4*
RPoint3d	**pContourLineMidPoint1**	*Coordinates of contour line mid point 1*
RPoint3d	**pContourLineMidPoint2**	*Coordinates of contour line mid point 2*
RPoint3d	**pContourLineMidPoint3**	*Coordinates of contour line mid point 3*
RPoint3d	**pContourLineMidPoint4**	*Coordinates of contour line mid point 4*

)

Functions

long **DeleteReinforcementParameters**

If successful, returns surface index. Otherwise returns error code(errIndexOutOfBounds, errDatabaseNotReady)

---

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 (errIndexOutOfBounds, errDatabaseNotReady).

---

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 (errIndexOutOfBounds, errDatabaseNotReady).

---

long **GetNormalVector** ([i/o] RPoint3d **Value**)

Get the normal vector of the surface. If successful returns surface index, otherwise returns an error code (errDatabaseNotReady).

---

long **GetReinforcementParameters** ([i/o] RReinforcementParameters **ReinforcementParameters,**
[out] SAFEARRAY(byte) **DesignCodeParameters**)

ReinforcementParameters	*reinforcement parameters*
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).*

If successful, returns the surface index, otherwise an error code (errDatabaseNotReady, errIndexOutOfBounds,seReinforcementParametersNotExists).

---

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**)

Thicknesses	*Array with thickness, with array indexes of:* *corners: 0,2,4 (rectangular and* triangular*) and 6 (rectangular only)* *contour midpoints: 1,3,5 (rectangular and* triangular*) and 7(rectangular only)* the centre: 6 (triangular) or 8 *(rectangular)*

If successful, returns length of the, otherwise an error code (errDatabaseNotReady, errIndexOutOfBounds,seReinforcementParametersNotExists).

---

long **Modify** ([in] SAFEARRAY(long) **LineIds,** [in] long **DomainId,**
[i/o] RSurfaceAttr **SurfaceAttr**)

LineIds	*indexes of coplanar lines (3 or 4) forming edges of a surface. Array length = PointCount.* *See AxisVMLines*
DomainId	*domain index if the surface element is part of a mesh otherwise 0* *0 ≤ DomainId ≤ AxisVMDomains.Count*
SurfaceAttr	*surface properties*

Redefines a new surface element.
If successful, returns the surface index, otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBounds, seLineCountCanBeOnly3Or4 [LineIds contains less than 3 or more than 4 line indexes], seCannotModify).

---

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**)

ReinforcementParameters	*reinforcement parameters*
DesignCodeParameters	*pointer to the reinforcement parameters record in SAFEARRAY format, record type depending on used design code. Typecast a corresponding record to a safearray (see* GetReinforcementParameters*), see also How to modify load data (SetLoad)*

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	**ContourColour_vb** • Visual Basic compatible property of **ContourColour**
SAFEARRAY(long)*	**ContourLines** *Get indexes of surface edges (3 or 4). Array length = PointCount.*
long	**DomainId** • *Get or set domain index if the surface element is part of a mesh otherwise 0*
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	**PointCount** *Get number of surface polygon vertices (3 or 4)*
ELongBoolean	**ReinforcementParametersExists** *True if finds reinforcement parameters (read only property)*
double	**StiffnessReduction** ***Warning!*** *This function was extended by StiffnessReduction_V153.Get or set stiffness reduction for the cross-section area (Ac). Default value is 1. Used only for atLinearStatic and atLinearVibration analysis types. (only valid for Eurocode, SIA, NTS standards)*
double	**StiffnessReduction_V153 [**ESurfaceStiffnessReduction **Component]** *Get or set stiffness reduction for the given component. Default value is 1. Used only for atLinearStatic and atLinearVibration analysis types. (only valid for Eurocode, SIA, NTS 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

RStiffnessesXYZ = (

double	**x, y, z**	*elastic foundation stiffness in local x, y, z directions of the surface [kN/m/m²]*

)

RNonLinearityXYZ = (

ELineNonLinearity	**x, y, z**	*nonlinear behaviour in local x, y, z directions of the surface*

)

RResistancesXYZ = (

double	**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*

Adds a surface support (elastic foundation) to a surface element.
If successful, returns the index of the surface support otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBounds).

---

long **AddSurfacePasternakFoundation** ([in] long **SurfaceId,**
[i/o] RStiffnessesXYZ **StiffnessesXYZ,**
[in] double **ShearStiffness,**
[i/o] RNonLinearityXYZ **NonLinearityXYZ,**
[i/o] RResistancesXYZ **ResistancesXYZ**)

SurfaceId	*surface index* *0 < SurfaceId ≤ AxisVMSurfaces.Count*
StiffnessesXYZ	*stiffnesses of the elastic foundation*
ShearStiffness	*shear stiffness of the support support (noted as R_G on the window for supports) [kN/m]*
ResistancesXYZ	*resistances of the elastic foundation*

Adds a Winkler-Pasternak surface support to a surface element.
If successful, returns the index of the surface support otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBounds).

---

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

Index	*surface support to delete*

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

---

long **DeleteSelected**

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

---

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

Select	*selection state*

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

NOTE: Call Refresh function afterwards if not called between functions BeginUpdate and EndUpdate

---

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

ItemIds *list of selected surface supports*

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

Properties

long	**Count** *Get number of surface supports in the model*
AxisVMSurfaceSupport*	**Item [**long **Index]** *Get surface support interface*
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*
long	**SelCount** *Get number of selected surface supports in the model. Negative number is an error code (errDatabaseNotReady).*

IAxisVMSurfaceSupport

An AxisVM surface support interface.

Enumerated types

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

Records / structures

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

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

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

Functions

long **GetNonLinearityXYZ** ([i/o] RNonLinearityXYZ **Value**)
Get the nonlinear behaviour of the surface support. If unsuccessful, returns an error code(errIndexOutOfBounds, errDatabaseNotReady).

long **GetPasternakStiffness** ([i/o] double **Value**)
Get the Pasternak stiffness (noted as R_G on the AxisVM window for supports) of the surface support. Its unit is [kN/m]. 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**)
Set the nonlinear behaviour of the surface support. If unsuccessful, returns an error code(errIndexOutOfBounds, errDatabaseNotReady).

long **SetPasternakStiffness** ([in] double **Value**)
Set the Pasternak stiffness (noted as R_G on the AxisVM window for supports) of the surface support. Its unit is [kN/m]. If unsuccessful, returns an error code(errIndexOutOfBounds, errDatabaseNotReady).

long **SetResistancesXYZ** ([i/o] RResistancesXYZ **Value**)
Set the resistance components of the surface support. If unsuccessful, returns an error code(errIndexOutOfBounds, errDatabaseNotReady).

long **SetStiffnessesXYZ** ([i/o] RStiffnessesXYZ **Value**)
Set the stiffness components of the surface support. If unsuccessful, returns an error code(errIndexOutOfBounds, errDatabaseNotReady).

Properties

ESurfaceSupportType **SupportType** *Get surface support type*

IAxisVMTask

Various AxisVM tasks

Enumerated types

enum **EDirectObjectDrawType** = {

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: ltNodalForce, ltBeamConcentrated, ltBeamDistributed*

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).

---

long **AddSupportModal** ([in] EPartItemType **ElementType**)

ElementType *type of support to be added, allowed: pitNode, pitLine*

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).

---

long **DrawObjectsDirectlyModal** ([in] EDirectObjectDrawType **ObjectType**)

ObjectType *type of object (element) to be added*

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).

---

long **ShowDomainMeshingFormModal** ()

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).

---

long **ShowLoadCasesAndGroupsFormModal** ()

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).

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	*grains parallel with top edge(tapered members)*
tgBottomEdge = 1	*grains patallel with bottom edge(tapered members)*

}

enum **ELoadPositionType** = {

lptUpper = 0	*load on top of the member*
lptAxis = 1	*load at centre of gravity of the member*
lptLower = 2	*load on bottom of the member*

}

enum **ETimberSLSEMethod** = {

tslsem_No = 0	*deflection based on actual displacements*
tslsem_2 = 1	*deflection based on displacement relative to both left and right*
tslsem_Left = 2	*deflection based on displacement relative to the left*
tslsem_Right = 3	*deflection based on displacement relative to the right*

}

enum **ETimberSLSLMethod** = {

tslslm_Member = 0	*design member length is the actual length of the member*
tslslm_Custom = 1	*design member length is specified by the slsCustomLy and/or slsCustomLz*
tslslm_Conn = 2	*design member length is based on connecting members and supports*

}

enum **ETimberSLSPreCamberCurve** = {

tslspcc_Quadratic = 0	*quadratic*
tslslm_Linear = 1	*linear*

}

enum **ETimberSLSDesignCreepMode** = {

tslsdcm_Code = 0	*creep as specified in code (conditioned)*
tslsdcm_User = 1	*user specified creep*
tslsdcm_CodeNotConditioned = 2	*creep as specified in code (not conditioned)*

}

enum **ETimberSLSFinalCalcMode** = {

tslsfcm_None = 0	*no checking of final deformation*
tslsfcm_DefCorrected = 1	*enhanced method for checking of final deformation*
tslsfcm_DefApprox = 2	*conservative method for checking of final deformation*

}

Error codes

enum **ETimberDesignMemberError** = {

tdmeLineListIsEmpty = -100001	*line list is empty*
tdmeNotConnectingLines = -100002	*lines are not connecting to each other*
tdmeInvalidNationalDesignCode = -100003	*design not valid for current national design code*
tdmeInvalidLines = -100004	*among the LineIDs some are not a valid target for timber design*
tdmeArrayLengthMismatch = -100005	*the length of input SafeArrays is different*

}

Records / structures

RTimberDesignParameters_EC_SIA_ITA= (***Warning!*** *This record has become obsolete, it was superseded by RTimberDesignParameters_EC_V153*

double	**Ky**	*see timber beam design in AxisVM manual*
double	**Kz**	*see timber beam design in AxisVM manual*
double	**Klt**	*see timber beam design in AxisVM manual*
ELoadPositionType	**LoadPosition**	*load position (used for lateral torsional buckling)*
ETimberGrain	**Grain**	*arrangement of grains in tapered members*
double	**LayerThickness**	*thickness of layers of glued laminated timber (Glulam) [m]*

)

RTimberDesignParameters = (***Warning!*** *This record has become obsolete, it was superseded by RTimberDesignParameters_V171*

ELongBoolean	**BreakAtElements**	*See here*
RTimberDesignParameters_EC_SIA_ITA	**EC_SIA_ITA**	*timber design parameters*

)

RTimberDesignParameters_EC_V153= (timber design parameters for EC, SIA, NTC

Warning! *This record has become obsolete, it was superseded by RTimberDesignParameters_EC_V171*

double	**Ky**	*flexural buckling factor about local y axis [ ]*
double	**Kz**	*flexural buckling factor about local z axis [ ]*
double	**Klt**	*lateral torsional buckling factor [ ]*
ELoadPositionType	**LoadPosition**	*load position (used for lateral torsional buckling)*
ETimberGrain	**Grain**	*arrangement of grains in tapered members*
double	**LayerThickness**	*thickness of layers of glued laminated timber (Glulam) [m]*
ELongBoolean	**slsEyLimitDef**	*allowed deflection limit in y direction is active*
ELongBoolean	**slsEzLimitDef**	*allowed deflection limit in z direction is active*
ELongBoolean	**slsUyDef**	*pre-camber in y direction is active*
ELongBoolean	**slsUzDef**	*pre-camber in z direction is active*
ETimberSLSEMethod	**slsEMode**	*deflection interpretation*
ETimberSLSLMethod	**slsLMode**	*member length interpretation*
ETimberSLSPreCamberCurve	**slsPreCamberCurve**	*type of the pre-camber curve*
double	**slsEyLimit**	*allowed deflection limit in y direction [m]*
double	**slsEzLimit**	*allowed deflection limit in z direction [m]*
double	**slsCustomLy**	*when slsLMode=tslslm_Custom,the user specified member length in y direction [m]*
double	**slsCustomLz**	*when slsLMode=tslslm_Custom,the user specified member length in z direction [m]*
double	**slsUy**	*pre-camber in y direction [m]*
double	**slsUz**	*pre-camber in z direction [m]*
ETimberSLSDesignCreepMode	**slsCreepMode**	*creep mode (only for SIA code)*

double	**slsPhi**	*user value for creep (only for SIA code, when slsCreepMode= tslsdcm_User)*
ELongBoolean	**FireResistDef**	*fire resistance design is active*
double	**fpKy**	*flexural buckling factor about local y axis, in case of fire [ ]*
double	**fpKz**	*flexural buckling factor about local z axis, in case of fire [ ]*
double	**fpKLT**	*lateral torsional buckling factor, in case of fire [ ]*

)

RTimberDesignParameters_EC_V171= (timber design parameters for EC, SIA, NTC

Warning! *This record has become obsolete, it was superseded by RTimberDesignParameters_EC_V172*

double	**Ky**	*flexural buckling factor about local y axis [ ]*
double	**Kz**	*flexural buckling factor about local z axis [ ]*
double	**Klt**	*lateral torsional buckling factor [ ]*
ELoadPositionType	**LoadPosition**	*load position (used for lateral torsional buckling)*
ETimberGrain	**Grain**	*arrangement of grains in tapered members*
double	**LayerThickness**	*thickness of layers of glued laminated timber (Glulam) [m]*
long	**CrossSectionIndex**	*cross-section index of the overriding cross-section from IAxisVMCrossSections. For no override, it must be set to 0, then the actual cross-section will be used.*
ELongBoolean	**slsEyLimitDef**	*allowed deflection limit in y direction is active*
ELongBoolean	**slsEzLimitDef**	*allowed deflection limit in z direction is active*
ELongBoolean	**slsUyDef**	*pre-camber in y direction is active*
ELongBoolean	**slsUzDef**	*pre-camber in z direction is active*
ETimberSLSEMethod	**slsEMode**	*deflection interpretation*
ETimberSLSLMethod	**slsLMode**	*member length interpretation*
ETimberSLSPreCamberCurve	**slsPreCamberCurve**	*type of the pre-camber curve*
double	**slsCustomLy**	*when slsLMode=tslslm_Custom,the user specified member length in y direction [m]*
double	**slsCustomLz**	*when slsLMode=tslslm_Custom,the user specified member length in z direction [m]*
double	**slsUy**	*pre-camber in y direction [m]*
double	**slsUz**	*pre-camber in z direction [m]*
ETimberSLSDesignCreepMode	**slsCreepMode**	*creep mode (only for SIA code)*
double	**slsPhi**	*user value for creep (only for SIA code, when slsCreepMode= tslsdcm_User)*
double	**xmax_Pos**	*position of the pre-camber maximum [m]*
ELongBoolean	**CheckFinalDeformation**	*final deformation checks*
ELongBoolean	**CheckInstaDeformation**	*instantaneous deformation checks*
ETimberSLSFinalCalcMode	**SLSFinalCalcMode**	*final deformation checking mode*
double	**slsEyLimit**	*allowed final deflection limit in y direction [m]*
double	**slsEzLimit**	*allowed final deflection limit in z direction [m]*

double	**slsInstaEyLimit**	*allowed instantaneous deflection limit in y direction [m]*
double	**slsInstaEzLimit**	*allowed instantaneous deflection limit in z direction [m]*
ELongBoolean	**FireResistDef**	*fire resistance design is active*
double	**fpKy**	*flexural buckling factor about local y axis, in case of fire [ ]*
double	**fpKz**	*flexural buckling factor about local z axis, in case of fire [ ]*
double	**fpKLT**	*lateral torsional buckling factor, in case of fire [ ]*

)

RTimberDesignParameters_EC_V172= (timber design parameters for EC, SIA, NTC

double	**Ky**	*flexural buckling factor about local y axis [ ]*
double	**Kz**	*flexural buckling factor about local z axis [ ]*
double	**Klt**	*lateral torsional buckling factor [ ]*
ELoadPositionType	**LoadPosition**	*load position (used for lateral torsional buckling)*
ETimberGrain	**Grain**	*arrangement of grains in tapered members*
double	**LayerThickness**	*thickness of layers of glued laminated timber (Glulam) [m]*
long	**CrossSectionIndex**	*cross-section index of the overriding cross-section from IAxisVMCrossSections. For no override, it must be set to 0, then the actual cross-section will be used.*
ELongBoolean	**slsEyLimitDef**	*allowed deflection limit in y direction is active*
ELongBoolean	**slsEzLimitDef**	*allowed deflection limit in z direction is active*
ELongBoolean	**slsUyDef**	*pre-camber in y direction is active*
ELongBoolean	**slsUzDef**	*pre-camber in z direction is active*
ETimberSLSEMethod	**slsEMode**	*deflection interpretation*
ETimberSLSLMethod	**slsLMode**	*member length interpretation*
ETimberSLSPreCamberCurve	**slsPreCamberCurve**	*type of the pre-camber curve*
double	**slsCustomLy**	*when slsLMode=tslslm_Custom,the user specified member length in y direction [m]*
double	**slsCustomLz**	*when slsLMode=tslslm_Custom,the user specified member length in z direction [m]*
double	**slsUy**	*pre-camber in y direction [m]*
double	**slsUz**	*pre-camber in z direction [m]*
ETimberSLSDesignCreepMode	**slsCreepMode**	*creep mode (only for SIA code)*
double	**slsPhi**	*user value for creep (only for SIA code, when slsCreepMode= tslsdcm_User)*
double	**xmax_Pos**	*position of the pre-camber maximum [m]*
ELongBoolean	**CheckFinalDeformation**	*final deformation checks*
ELongBoolean	**CheckInstaDeformation**	*instantaneous deformation checks*
ETimberSLSFinalCalcMode	**SLSFinalCalcMode**	*final deformation checking mode*
double	**slsEyLimit**	*allowed final deflection limit in y direction [m]*
double	**slsEzLimit**	*allowed final deflection limit in z direction [m]*

double	**slsInstaEyLimit**	*allowed instantaneous deflection limit in y direction [m]*
double	**slsInstaEzLimit**	*allowed instantaneous deflection limit in z direction [m]*
ELongBoolean	**FireResistDef**	*fire resistance design is active*
double	**fpKy**	*flexural buckling factor about local y axis, in case of fire [ ]*
double	**fpKz**	*flexural buckling factor about local z axis, in case of fire [ ]*
double	**fpKLT**	*lateral torsional buckling factor, in case of fire [ ]*
ELongBoolean	**Buckling**	*take into account buckling (intended only for X7R3 and later)*
ELongBoolean	**LTB**	*take into account lateral torsional buckling (intended only for X7R3 and later)*

)

RTimberDesignParameters_V153 = (

Warning! *This record has become obsolete, it was superseded by RTimberDesignParameters_V171*

ELongBoolean	**BreakAtElements**	*See here*
RTimberDesignParameters_EC_V153	**EC**	*timber design parameters (for EC, SIA, NTC)*

)

RTimberDesignParameters_V171 = (

Warning! *This record has become obsolete, it was superseded by RTimberDesignParameters_V172*

ELongBoolean	**BreakAtElements**	*See here*
RTimberDesignParameters_EC_V171	**EC**	*timber design parameters (for EC, SIA, NTC)*

)

RTimberDesignParameters_V172 = (

ELongBoolean	**BreakAtElements**	*See here*
RTimberDesignParameters_EC_V172	**EC**	*timber design parameters (for EC, SIA, NTC)*

)

Functions

long **Add** ([i/o] RTimberDesignParameters* **TimberDesignParameters**,
[in] SAFEARRAY(long)* **LineIds**)

Warning! *This function has become obsolete, was superseded by Add_V171*

TimberDesignParameters *timber design parameters*
LineIds *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_V153** ([i/o] RTimberDesignParameters_V153* **TimberDesignParameters**,
[in] SAFEARRAY(long)* **LineIds**)

Warning! *This function has become obsolete, was superseded by Add_V171*

TimberDesignParameters *timber design parameters*
LineIds *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_V171** ([i/o] RTimberDesignParameters_V171* **TimberDesignParameters**,
[in] SAFEARRAY(long)* **LineIds**)

Warning! *This function has become obsolete, was superseded by Add_V172*

TimberDesignParameters *timber design parameters*
LineIds *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, tdmeInvalidLines).

---

long **Add_V172** ([i/o] RTimberDesignParameters_V172* **TimberDesignParameters**,
[in] SAFEARRAY(long)* **LineIds**)

TimberDesignParameters *timber design parameters*
LineIds *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, tdmeInvalidLines).

---

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

---

long **BulkAdd** ([i/o] SAFEARRAY(RTimberDesignParameters_V171)* **TimberDesignParameters**,
[in] SAFEARRAY(VARIANT)* **LineIds**)

Warning! *This function has become obsolete, was superseded by BulkAdd_V172*

TimberDesignParameters *array of timber design parameters*
LineIds Array of VARIANTs, each VARIANT being a SAFEARRAY(long) with lineIDs

Defines multiple new timber design members. The length of TimberDesignParameters should be the same as the length of LineIds. If successful, returns the TimberDesignMember index of the lastly created TimberDesignMember otherwise an error code (errDatabaseNotReady, errIndexOutOfBounds, tdmeInvalidNationalDesignCode, tdmeNotConnectingLines, tdmeLineListIsEmpty, tdmeInvalidLines, *tdmeArrayLengthMismatch).*

---

long **BulkAdd_V172** ([i/o] SAFEARRAY(RTimberDesignParameters_V172)* **TimberDesignParameters**,
[in] SAFEARRAY(VARIANT)* **LineIds**)

TimberDesignParameters *array of timber design parameters*
LineIds Array of VARIANTs, each VARIANT being a SAFEARRAY(long) with lineIDs

Defines multiple new timber design members. The length of TimberDesignParameters should be the same as the length of LineIds. If successful, returns the TimberDesignMember index of the lastly created TimberDesignMember otherwise an error code (errDatabaseNotReady, errIndexOutOfBounds, tdmeInvalidNationalDesignCode, tdmeNotConnectingLines, tdmeLineListIsEmpty, tdmeInvalidLines, *tdmeArrayLengthMismatch).*

---

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**)

Index *timber design member index (0 < Index ≤ IAxisVMSteelDesignMembers.Count)*

Deletes a timber design member. If successful, returns the timber design member index (1 ≤ Index ≤ Count), otherwise returns an error code (errDatabaseNotReady, errIndexOutOfBounds).

---

long **DeleteSelected**

Returns number of deleted timber design member, otherwise returns an error code (errDatabaseNotReady,errNotSupportedByNationalDesignCode, tdmeInvalidNationalDesignCode).

---

long **GetDesignParameters** ([in] long **Index,**
[i/o] RTimberDesignParameters* **TimberDesignParameters**)

Warning! *This function has become obsolete, was superseded by GetDesignParameters_V171*

Index *timber design member index (0 < Index ≤ IAxisVMTimberDesignMembers.Count)*
TimberDesignParameters *timber design parameters*
If successful, returns index otherwise an error code (errDatabaseNotReady, errIndexOutOfBounds, tdmeInvalidNationalDesignCode)

---

long **GetDesignParameters_V153** ([in] long **Index,**
[i/o] RTimberDesignParameters_V153* **TimberDesignParameters**)

Warning! *This function has become obsolete, was superseded by GetDesignParameters_V171*

Index *timber design member index (0 < Index ≤ IAxisVMTimberDesignMembers.Count)*
TimberDesignParameters *timber design parameters*
If successful, returns index otherwise an error code (errDatabaseNotReady, errIndexOutOfBounds, tdmeInvalidNationalDesignCode)

---

long **GetDesignParameters_V171** ([in] long **Index,**
[i/o] RTimberDesignParameters_V171* **TimberDesignParameters**)

Warning! *This function has become obsolete, was superseded by GetDesignParameters_V172*

Index *timber design member index (0 < Index ≤ IAxisVMTimberDesignMembers.Count)*
TimberDesignParameters *timber design parameters*
If successful, returns index otherwise an error code (errDatabaseNotReady, errIndexOutOfBounds, tdmeInvalidNationalDesignCode)

---

long **GetDesignParameters_V172** ([in] long **Index,**
[i/o] RTimberDesignParameters_V172* **TimberDesignParameters**)

Index *timber design member index (0 < Index ≤ IAxisVMTimberDesignMembers.Count)*
TimberDesignParameters *timber design parameters*
If successful, returns index otherwise an error code (errDatabaseNotReady, errIndexOutOfBounds, tdmeInvalidNationalDesignCode)

---

long **GetLineIds** ([in] long **Index**, [out] SAFEARRAY(long)*** LineIds**)

Index *timber design member index (0 < Index ≤ TimberDesignMembers.Count)*
LineIds *Index array (long) with lineIDs*
Returns number of lines in timber design member, otherwise returns an error code (errDatabaseNotReady,errIndexOutOfBounds,errNotSupportedByNationalDesignCode, tdmeInvalidNationalDesignCode).

---

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

ItemIds *Index list of selected edge connections*
Returns number of selected timber design member, otherwise returns an error code (errDatabaseNotReady, errNotSupportedByNationalDesignCode, tdmeInvalidNationalDesignCode).

---

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

Select *selection state*

If Select is True, selects all timber design members.
If Select is False, deselects all timber design members.
If successful, returns the number of selected timber design member, otherwise returns an error code (errDatabaseNotReady,errNotSupportedByNationalDesignCode, tdmeInvalidNationalDesignCode).

NOTE: Call Refresh function afterwards if not called between functions BeginUpdate and EndUpdate

---

long **SetDesignParameters** ([in] long **Index,**
[i/o] RTimberDesignParameters* **TimberDesignParameters**)

Warning! *This function has become obsolete, was superseded by SetDesignParameters_V171*

Index *timber design member index (0 < Index ≤ IAxisVMTimberDesignMembers.Count)*
TimberDesignParameters *timber design parameters*
If successful returns timber design memberindex, otherwise an error code (errDatabaseNotReady, errIndexOutOfBounds, tdmeInvalidNationalDesignCode)

long **SetDesignParameters_V153** ([in] long **Index,**
[i/o] RTimberDesignParameters_V153* **TimberDesignParameters**)

Warning! *This function has become obsolete, was superseded by SetDesignParameters_V171*

Index *timber design member index (0 < Index ≤ IAxisVMTimberDesignMembers.Count)*
TimberDesignParameters *timber design parameters*
If successful returns timber design memberindex, otherwise an error code (errDatabaseNotReady, errIndexOutOfBounds, tdmeInvalidNationalDesignCode)

long **SetDesignParameters_V171** ([in] long **Index,**
[i/o] RTimberDesignParameters_V171* **TimberDesignParameters**)

Warning! *This function has become obsolete, was superseded by SetDesignParameters_V172*

Index *timber design member index (0 < Index ≤ IAxisVMTimberDesignMembers.Count)*
TimberDesignParameters *timber design parameters*
If successful returns timber design memberindex, otherwise an error code (errDatabaseNotReady, errIndexOutOfBounds, tdmeInvalidNationalDesignCode)

long **SetDesignParameters_V172** ([in] long **Index,**
[i/o] RTimberDesignParameters_V172* **TimberDesignParameters**)

Index *timber design member index (0 < Index ≤ IAxisVMTimberDesignMembers.Count)*
TimberDesignParameters *timber design parameters*
If successful returns timber design memberindex, otherwise an error code (errDatabaseNotReady, errIndexOutOfBounds, tdmeInvalidNationalDesignCode)

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*

ELongBoolean **Selected** [long **Index**] • *Get or set the selection status of a TimberDesignMember*
NOTE: Call Refresh function afterwards if not called between functions BeginUpdate and EndUpdate

long **SelCount** *Get number of TimberDesignMembers in the model*

IAxisVMTimeIncrementFunctions

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.

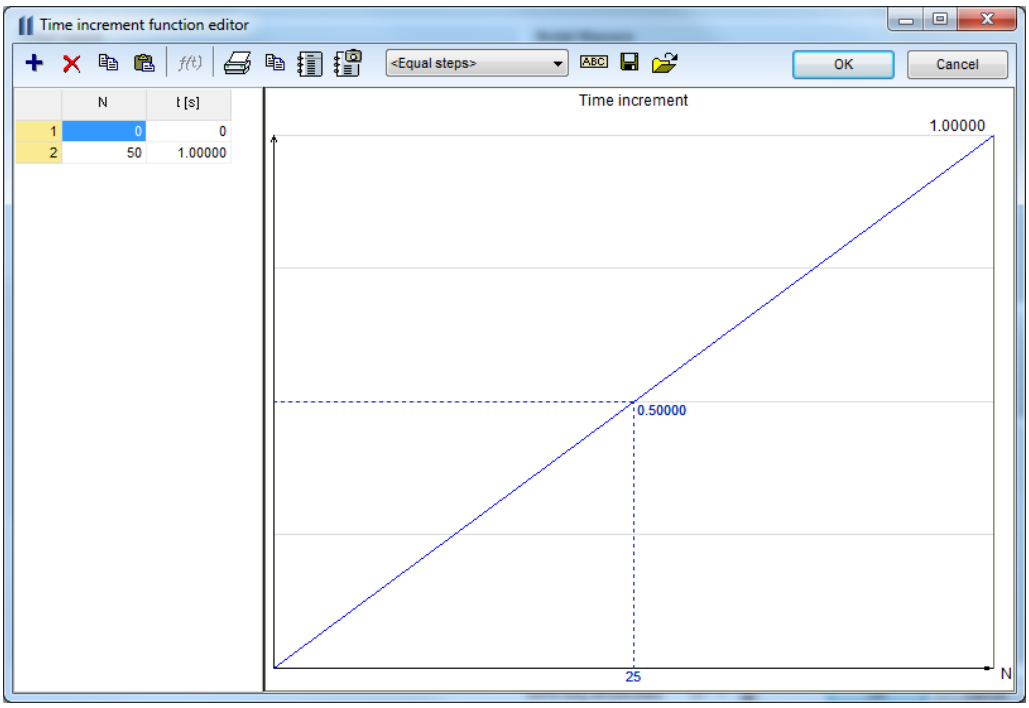
Functions

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 *time increment function index*

If successful, returns number of time increment functions after delete, otherwise an error code (errDatabaseNotReady or errIndexOutOfBounds).

---

long **DeletePoint** ([in] long **index,** [in] long **PointIndex**)

index *time increment function index*

PointIndex *index of a point in time 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 *time increment function index*

StartPointIndex *index of the start point in time increment function*

EndPointIndex *index of the end point in time increment function*

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 *time increment function index*

FunctionPoints *point array of the time increment function*

Deletes function point range starting with point index **StartPointIndex** (including) and ending with **EndPointIndex** (including).

If successful, returns number of points of the function, otherwise an error code (errDatabaseNotReady, errIndexOutOfBounds)

---

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

Name *Name of the time increment function*

If successful, returns function index, otherwise an error code (errDatabaseNotReady).

---

long **Modify** ([in] long **index,** [in] SAFEARRAY(RPoint2d)* **FunctionPoints**)

index *time increment function index*

FunctionPoints *point array of the time 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 *time increment function index*

FileName *Name of the file used for saving*

Saves function to AxisVM directory \timeinc with file extension: tnc

If successful, returns function index, otherwise an error code (errDatabaseNotReady, errIndexOutOfBounds or fueFileExists).

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,	*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*
ddmNoDuplication= 0x1,	*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 bem*
ddmDuplication= 0x2 }	*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*

enum **EVirtualBeamType** = {

vbtVirtualBeam = 0x0,	*virtual element for integration is assigned to domains (virtual beam)*
vbtVirtualStrip = 0x1 }	*virtual element for integration in the model is given by a strip (virtual strip)*

enum **EVBDefinitionType** = {

vbdtCentroid = 0x0,	*virtual beam is defined by centroids*
vbdtStraight = 0x1	*virtual beam is defined as a straight line*
vbdt1PAndV = 0x2	*virtual beam is defined with the help of a point and a vector*
vbdt2P = 0x3 }	*virtual beam is defined with the help of two points*

enum **EVSDefinitionType** = {

vsdtCentroid = 0x0,	*virtual strip is defined without eccentricity*
vsdtEccentric = 0x1 }	*virtual strip is defined with eccentricity*

Error codes

enum **EVirtualBeamError** = {

vbeDomainIndexOutOfBounds = –100001,	*domain index is out of bounds*
vbeDomainIndexIsInvalid = –100002,	*domain index is invalid (e.g. it is given twice in the list)*
vbeDomainListIsEmpty = –100003,	*domain list is empty*
vbeChainIndexOutOfBounds = –100004,	*domain index is out of bounds*
vbeDuplication = –100005,	*a given domain already belongs to another virtual beam*

	(note: only is EDomainsDuplicateMode = ddmDuplicationError)
vbeInvalidParameters = –100006,	*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*
vbeInvalidReferenceParams = –100007	*the type, the length or the parameters of the reference is invalid*
vbeVirtualBeamNoSection = -100008	*section cannot be created*
vbeInvalidSections = -100009	*cross section is invalid*
}	

Records / structures

RVirtualBeamParams= (

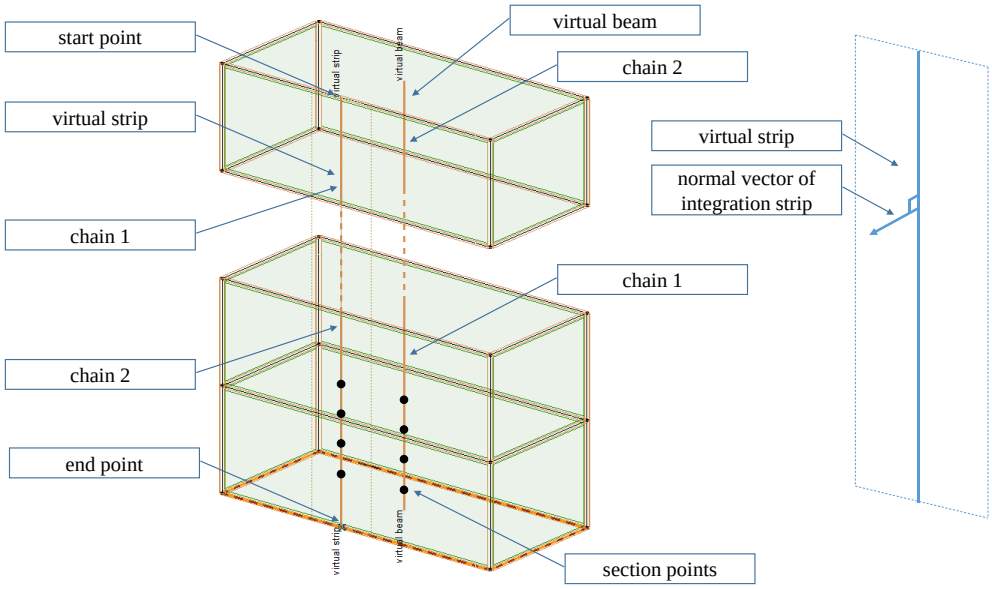
BSTR	**Name**	*name of the domain virtual beam*
long	**LocXVid**	*index of the reference vector (see IAxisVMReferences) being parallel with the local X Note: if it is equal to 0 then the reference vector is automatic*
long	**LocZVid**	*index of the reference vector (see IAxisVMReferences) being parallel with the local Z Note: if it is equal to 0 then the reference vector is automatic*
RPoint3d	**LocXV**	*local X vector (note: used only for output)*
RPoint3d	**LocZV**	*local Z vector (note: used only for output)*
long	**SectionI**	*cross section index related to ith end of virtual beam*
long	**SectionJ**	*cross section index related to jth end of virtual beam*
EVBDefinitionType	**DefinitionType**	*definition type*
RPoint3d	**P1**	*first point*
RPoint3d	**P2**	*second point*
ELongBoolean	**InnerDomains**	*consider inner domains while creating virtual beam*

)

RVirtualStripParams= (

BSTR	**Name**	*name of the virtual strip*
RPoint3d	**StartP**	*start point of the virtual strip, coordinates in metres.*
RPoint3d	**EndP**	*end point of the virtual strip, coordinates in metres.*
long	**SectionI**	
long	**SectionJ**	
RPoint3d	**NormV**	*normal vector of integration strip note: if it is not perpendicular to the vector from StartP to EndP, its perpendicular component is considered*
double	**L**	*integration width on the left hand side [m]*
double	**R**	*integration width on the right hand side [m]*
EVSDefinitionType	**DefinitionType**	*definition type*
double	**EccIY**	*local y eccentricity at start point of virtual strip [m]*
double	**EccIZ**	*local z eccentricity at start point of virtual strip [m]*
double	**EccJY**	*local y eccentricity at end point of virtual strip [m]*
double	**EccJZ**	*local z eccentricity at end point of virtual strip [m]*

)



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<Index<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 ≤ Index ≤ 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, vbeInvalidParameters, errInvalidName or vbeInvalidReferenceParams)

---

long **GetVirtualStripParams** ([in] long **Index,** [i/o] RVirtualStripParams **VirtualStripParams**)

Index	*index of the strip virtual beam*
VirtualStripParams	*virtual stip 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*

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)

---

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

Name *name of the virtual beam*

Retrieves the index of the virtual beam/strip having name equal to Name.
If successful, returns the index of the virtual beam/strip, otherwise returns an error code (errDatabaseNotReady, errNotFound).

---

long **GetDomains** ([in] long **Index,** [out] SAFEARRAY(long)* **DomainIds**)

Index *index of the domain virtual beam/strip*

DomainIds *array of domain indices considered for the virtual beam/strip*
0<Index<IAxisVMDomains.Count+1

Retrieves a list of domains connected to the virtual beam/strip identified with Index. If successful returns the number of domains of the virtual beam/strip, otherwise an error code (errDatabaseNotReady, errIndexOutOfBounds or vbeInvalidParameters).

---

long **GetExtendedDomainList** ([in] long **Index,** [out] SAFEARRAY(long)* **DomainIds**)

Index *index of the domain virtual beam/strip*

DomainIds *array of domain indices considered for the virtual beam/strip*
0<Index<IAxisVMDomains.Count+1

Retrieves a list of domains connected to the virtual beam/strip 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/strip, otherwise an error code (errDatabaseNotReady, errIndexOutOfBounds or vbeInvalidParameters).

---

long **ModifyDomains** ([in] long **Index**, [i/o] SAFEARRAY(long)* **DomainIds,** [in] EVBDomainsDuplicateMode **DuplicateMode**)

Index *index of the domain virtual beam/strip*

DomainIds *array of domain indices considered for the new virtual beam/strip*
0<Index<IAxisVMDomains.Count+1

DuplicateMode *mode of handling of duplication*

Modifies the list of domains connected to virtual beam/strip identified with Index. If successful returns the index of the virtual beam/strip, otherwise an error code (errDatabaseNotReady, errIndexOutOfBounds, vbeDomainIndexIsInvalid, vbeDomainIndexOutOfBounds, vbeDomainListIsEmpty, vbeInvalidParameters or vbeDuplication).

Properties

long **Count** *number of virtual beams and strips in the model*

long **ChainCount** [long **Index**]

Index *index of the virtual beam/strip*

Get the number of chains related to a virtual beam/strip identified with Index

long **SectionCount** [long **Index,** long **ChainIndex**]

Index *index of the virtual beam/strip*

ChainIndex *index of a specific chain of a virtual beam/strip*

Get the number of section points related to a virtual beam's/strip's chain identified with ChainIndex

double **Length** [long **Index,** long **ChainIndex**]

Index *index of the virtual beam/strip*

ChainIndex *index of a specific chain of a virtual beam/strip*

Get the length of a virtual beam's/strip's chain

BSTR **Name** [long **Index**] •

Index *index of the virtual beam/strip*

Get or set the name of a virtual beam/strip

EVirtualBeamType **VirtualBeamType** [long **Index**]

Index *index of the virtual beam/strip*

Get the type of a virtual beam/strip

long **IndexOfUID** [long **UID**] *Get index of the virtual beam/strip*

UID *unique index of the virtual beam/strip*

long **UID** [long **Index**] *Get unique index of the virtual beam/strip which remains the same while exists in the model*

Index *index of the virtual beam/strip*

IAxisVMWindLoad

Wind load management

Records / structures

RWindLoadParams_V161 = (

double	**A**	*Altitude above sea level*
double	**v_b0**	*Basic wind velocity*
double	**c_dir_xp**	*directional factors in +X direction*
double	**c_dir_xm**	*directional factors in -X direction*
double	**c_dir_yp**	*directional factors in +Y direction*
double	**c_dir_ym**	*directional factors in -Y direction*
double	**c_season**	*Season factor*
ELongBoolean	**TerrainCategoryDifferent**	*Terrain category different in directions*
ELongBoolean	**CustomDirectionalFactors**	*Custom directional factors 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*
double	**c_o**	*Orography factor*
double	**Iw**	*Importance factor (ndcEuroCode_RO only)*
long	**Zone**	*Number of supported zones depened on national design code where first zone name is 1, second 2, etc..* *Used only in listed design codes:* ***ndcEuroCode_PL:*** *Strefa 1, Strefa 2, Strefa 3,* ***ndcEuroCode_GER:*** *Zone 1, Zone 2, Zone 3, Zone 4* ***ndcItalian:*** *Zona 1, Zona 2, Zona 3, Zona 4, Zona 5, Zona 6, Zona 7, Zona 8, Zona 9* ***ndcEuroCode_RO:*** *Zona 1, Zona 2, Zona 3, Zona 4, Zona 5* ***ndcEuroCode_CZ:*** *Zone 1, Zone 2, Zone 3, Zone 4, Zone 5* ***ndcEuroCode_NL****: Zone I, Zone II, Zone III* ***ndcEuroCode_B:*** *Zone 1, Zone 2, Zone 3, Zone 4*
double	**AltitudeFactor**	*Altitude factor (only for code EC_NO)*
double	**TurbulenceFactor**	*Turbulence factor (only for code EC_NO)*

) parameters of the wind load. Not all fields will be active at the same time.

RWindLoadParams_V171 = (

double	**A**	*Altitude above sea level*
double	**v_b0**	*Basic wind velocity*
double	**c_dir_xp**	*directional factors in +X direction*
double	**c_dir_xm**	*directional factors in -X direction*
double	**c_dir_yp**	*directional factors in +Y direction*
double	**c_dir_ym**	*directional factors in -Y direction*
double	**c_season**	*Season factor*
ELongBoolean	**TerrainCategoryDifferent**	*Terrain category different in directions*
ELongBoolean	**CustomDirectionalFactors**	*Custom directional factors 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*
double	**c_o**	*Orography factor*
double	**Iw**	*Importance factor (ndcEuroCode_RO only)*
long	**Zone**	*Number of supported zones depened on national design code where first zone name is 1, second 2, etc..* *Used only in listed design codes:* ***ndcEuroCode_PL:*** *Strefa 1, Strefa 2, Strefa 3,* ***ndcEuroCode_GER:*** *Zone 1, Zone 2, Zone 3, Zone 4* ***ndcItalian:*** *Zona 1, Zona 2, Zona 3, Zona 4, Zona 5, Zona 6, Zona 7, Zona 8, Zona 9* ***ndcEuroCode_RO:*** *Zona 1, Zona 2, Zona 3, Zona 4, Zona 5* ***ndcEuroCode_CZ:*** *Zone 1, Zone 2, Zone 3, Zone 4, Zone 5* ***ndcEuroCode_NL****: Zone I, Zone II, Zone III* ***ndcEuroCode_B:*** *Zone 1, Zone 2, Zone 3, Zone 4*
double	**AltitudeFactor**	*Altitude factor (only for code EC_NO)*
double	**TurbulenceFactor**	*Turbulence factor (only for code EC_NO)*
ELongBoolean	**LoadCasePerSubStruct**	*separate load cases for each substructure*

) parameters of the wind load. Not all fields will be active at the same time.

Functions

long **Add** ([in] RWindSubStructParams **Params,** [in] SAFEARRAY(long)* **LoadPanels**)

Params *parameters of the wind substructure*

LoadPanels *list of the load panel indexes*
0<Index<=IAxisVMLoadPanel.Count

Adds a new wind substructure. If successful returns the index of the substructure, otherwise an error code (errDatabaseNotReady or wleNoWindLoadDefined*).*

---

long **CreateWindCases** ()

Creates the wind load cases. The wind load already has to be defined (through the Params property), with at least one substructure. If successful returns the count of wind load cases, otherwise an error code (errDatabaseNotReady or wleNoWindLoadDefined*).*

---

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

Index *Index of the substructure*
0<Index<=IAxisVMWindLoad.Count

Deletes a substructure. If successful returns 1, otherwise an error code (errDatabaseNotReady or errIndexOutOfBounds).

---

long **DeleteWindCases** ()

Deletes the wind load cases, and sets Defined to lbFalse. If successful returns 1, otherwise an error code (errDatabaseNotReady).

Properties

long	**Count**• *Substructure count*
RWindLoadParams_V161	**Params** • *Get or set the parameters of the wind load*
RWindLoadParams_V171	**Params_V171** • *Get or set the parameters of the wind load*
AxisVMWindSubStructure	**Item**• *Substructure (valid indexes are 1..Count)*
ELongBoolean	**CodeAllowed**• *Wind load can be defined for this national code*
ELongBoolean	**Defined**• *Is wind load defined. Setting the params will define it.*

AxisVMWindSubStructure

One element of the wind load substructures.

Enumerated types

enum **EWindStructureRoofType** = {

wsrtUndefined = 0,	*not used*
wsrtFlat = 1,	*flat roof*
wsrtMonopitch = 2,	*monopitch roof*
wsrtDuopitch = 3,	*duopitch roof*
wsrtHip = 4,	*hip roof*
wsrtBarrel = 5 ,	*barrel roof*
wsrtMonopitchCanopy = 6 ,	*monopitch canopy*
wsrtDuopitchCanopy = 7 ,	*duopitch canopy*
wsrtFreestandingWall = 8,	*free-standing wall*
wsrtSignBoard = 9 }	*signboard*

roof type of the substructure. See section **4.10.14. Wind load – SWG module** *in AxisVM user manual for more explanation.*

enum **ERoofInternalPressure** = *{*

ripUndefined = 0,	*not used*
ripAuto = 1,	*based on Eurocode. Called Approximative on the user interface*
ripOpeningAreaBased = 2,	*based on area of openings*
ripInternalCoeff = 3,	*coefficients provided directly by the user*
ripOff = 4 }	*no internal pressure*

internal pressure type. See section **4.10.14. Wind load – SWG module** *in AxisVM user manual for more explanation.*

enum **ERoofFrictionEffect** = *{*

rfeSmooth = 1,	*smooth surface*
rfeRough = 2,	*rough surface*
rfeVeryRough = 3,	*very rough surface*
rfeCustom = 4 }	*c_fr directly controlled by the user*

friction effect type. See section **4.10.14. Wind load – SWG module** *in AxisVM user manual for more explanation.*

enum **ERoofMultiSpanPos** = *{*

rmspFirst = 1,	*first or last*
rmspSecond = 2,	*second or penultimate*
rmspInner = 3 }	*inner span*

location of the roof in the sequence. See section **4.10.14. Wind load – SWG module** *in AxisVM user manual for more explanation.*

enum **ERoofMultiSpanDir** = *{*

rmsdX = 1,	*roof axis in the x direction*
rmsdY = 2 }	*roof axis in the y direction*

direction of the roof axis. See section **4.10.14. Wind load – SWG module** *in AxisVM user manual for more explanation.*

Records / structures

RWindSubStructParams = (

BSTR	**Name**	*substructure name (should be concise)*
EWindStructureRoofType	**RoofType**	*type of the roof*
ELongBoolean	**TorsionalEffect**	*only for rooftypes : wsrtFlat, wsrtMonopitch, wsrtDuopitch, wsrtHip, wsrtBarrel. Controls whether there should be additional load cases created for torsional winds*
ERoofInternalPressure	**InternalPressure**	*how should be the internal pressure handled*
double	**Mu_Xp**	*only for rooftypes : wsrtFlat, wsrtMonopitch, wsrtDuopitch, wsrtHip, wsrtBarrel and InternalPressure type ripOpeningAreaBased : the value of μ_{X+} [ ]*
double	**Mu_Xm**	*only for rooftypes : wsrtFlat, wsrtMonopitch, wsrtDuopitch, wsrtHip, wsrtBarrel and InternalPressure type ripOpeningAreaBased : the value of μ_{X-} [ ]*
double	**Mu_Yp**	*only for rooftypes : wsrtFlat, wsrtMonopitch, wsrtDuopitch, wsrtHip, wsrtBarrel and InternalPressure type ripOpeningAreaBased : the value of μ_{Y+} [ ]*
double	**Mu_Ym**	*only for rooftypes : wsrtFlat, wsrtMonopitch, wsrtDuopitch, wsrtHip, wsrtBarrel and InternalPressure type ripOpeningAreaBased : the value of μ_{Y-} [ ]*
double	**cpi_Xp**	*only for rooftypes : wsrtFlat, wsrtMonopitch, wsrtDuopitch, wsrtHip, wsrtBarrel and InternalPressure type ripInternalCoeff : the value of c_{pX+} [ ]*
double	**cpi_Xm**	*only for rooftypes : wsrtFlat, wsrtMonopitch, wsrtDuopitch, wsrtHip, wsrtBarrel and InternalPressure type ripInternalCoeff : the value of c_{pX-} [ ]*
double	**cpi_Yp**	*only for rooftypes : wsrtFlat, wsrtMonopitch, wsrtDuopitch, wsrtHip, wsrtBarrel and InternalPressure type ripInternalCoeff : the value of c_{pY+} [ ]*
double	**cpi_Ym**	*only for rooftypes : wsrtFlat, wsrtMonopitch, wsrtDuopitch, wsrtHip, wsrtBarrel and InternalPressure type ripInternalCoeff : the value of c_{pY-} [ ]*
ERoofFrictionEffect	**FricionEffect**	*only for rooftypes : wsrtFlat, wsrtMonopitch, wsrtDuopitch, wsrtHip, wsrtBarrel, wsrtMonopitchCanopy, wsrtDuopitchCanopy, wsrtFreestandingWall. It controls the handling of c_{fr} : rfeSmooth, rfeRough, rfeVeryRough for preset values and rfeCustom for custom values*
double	**CustomFriction**	*only for rooftypes : wsrtFlat, wsrtMonopitch, wsrtDuopitch, wsrtHip, wsrtBarrel, wsrtMonopitchCanopy, wsrtDuopitchCanopy, wsrtFreestandingWall and FrictionEffect rfeCustom. The custom value of c_{fr} [ ]*
ELongBoolean	**IsRelativeElevation**	*only for rooftypes : wsrtFlat, wsrtMonopitch, wsrtDuopitch, wsrtHip, wsrtBarrel, wsrtMonopitchCanopy, wsrtDuopitchCanopy,*

		wsrtFreestandingWall. Controls whether elevation is relative to terrain level
double	**RelativeElevation**	*only for rooftypes : wsrtFlat, wsrtMonopitch, wsrtDuopitch, wsrtHip, wsrtBarrel, wsrtMonopitchCanopy, wsrtDuopitchCanopy, wsrtFreestandingWall and IsRelativeElevation lbTrue. Also always for roof type wsrtSignBoard. The relative elevation Δ_Z [m]*
ELongBoolean	**IsMultiSpan**	*only for rooftypes : wsrtMonopitchCanopy, wsrtDuopitchCanopy. Controls whether it is part of a multispan setup*
ERoofMultiSpanPos	**MultiSpanPos**	*only for rooftypes : wsrtMonopitchCanopy, wsrtDuopitchCanopy and IsMultiSpan set for lbTrue : the position of the roof in a multispan setup*
ERoofMultiSpanDir	**MultiSpanDir**	*only for rooftypes : wsrtMonopitchCanopy, wsrtDuopitchCanopy and IsMultiSpan set for lbTrue : the direction of the roof axis*
EFlatRoofEdgeType	**FlatRoofEdgeType**	*only for rooftype wsrtFlat : the type of the roof edge*
double	**FlatRoofEdgeParam**	*only for rooftype wsrtFlat. Depending on FlatRoofEdgeType : retWithParapetWall : parapet height h_p [m] retRoundEaves : rounding radius r [m] retMansardEaves : pitch angle α [°]*
double	**Blockage_Xp**	*only for rooftypes : wsrtMonopitchCanopy, wsrtDuopitchCanopy. The blockage factor φ_{X+} [ ]*
double	**Blockage_Xm**	*only for rooftypes : wsrtMonopitchCanopy, wsrtDuopitchCanopy. The blockage factor φ_{X-} [ ]*
double	**Blockage_Yp**	*only for rooftypes : wsrtMonopitchCanopy, wsrtDuopitchCanopy. The blockage factor φ_{Y+} [ ]*
double	**Blockage_Ym**	*only for rooftypes : wsrtMonopitchCanopy, wsrtDuopitchCanopy. The blockage factor φ_{Y-} [ ]*
double	**Solidity**	*only for rooftype wsrtFreestandingWall. The solidity factor φ [ ]*

) parameters of a wind substructure. Not all fields will be active at the same time.

Remark : while return corner for free standing walls can be specified per structure, that is actually a global property of the load panel, and can be set at the load panel.

Functions

long **GetPanels** ([out] SAFEARRAY(long)* **LoadPanels**)

LoadPanels *list of the load panel indexes*
0<Index<=IAxisVMLoadPanel.Count

Queries the load panels associated to the substructure. If successful returns the length of LoadPanels, otherwise an error code (errDatabaseNotReady or errIndexOutOfBounds).

---

long **SetPanels** ([in] SAFEARRAY(long)* **LoadPanels**)

LoadPanels *list of the load panel indexes*
0<Index<=IAxisVMLoadPanel.Count

Sets the load panels associated to the substructure. If successful returns the index of the substructure, otherwise an error code (errDatabaseNotReady or errIndexOutOfBounds).

Properties

RWindSubStructParams **Params** • *Get or set the parameters of the wind substructure*

IAxisVMWindows

Used for changing display settings of all AxisVM windows

Enumerated types

enum **EDisplayMode** = {

dmDiagram = 0x00,	*Display diagram*
dmDiagramFilled = 0x07,	*Display filled diagram*
dmDiagramAvgVals = 0x01,	*Display diagram+ average values if possible*
dmSectionLine = 0x02,	*Display section lines*
dmSectionFilled = 0x08,	*Display filled section lines*
dmIsolines = 0x03 ,	*Display Isolines*
dmIsosurfaces2D = 0x04 ,	*Display Isosurfaces 2D*
dmIsosurfaces3D = 0x05 ,	*Display Isosurfaces 3D*
dmIsosurfacesAvgVals = 0x09,	*Display Isosurfaces with average values*
dmSolidModel = 0x0,	*Display values as colored rendered solids*
dmNone= 0x06 }	*Display none*

Type of display mode.

enum **EDisplayAnalysisType** = {

datLinear = 0x00,	*Display results of linear analysis*
datNonLinear = 0x01 }	*Display results of nonlinear analysis*

Analysis type of displayed results

enum **EDisplayShape** = {

dsUndeformed = 0x00,	*Model is not deformed*
dsDeformed = 0x01 }	*Model is deformed*

Show deformation or not.

enum **EDisplayedEnvelopes** = {

de_Current = 0x00,	*Only the selected envelope*
de_Custom = 0x01,	*Only custom envelopes*
de_All = 0x02 }	*All envelopes*

Displayed envelopes

enum **EActualReinforcementLabelType** = {

arltRebarsAndReinfValues = 0x00,	*Rebars + Reinforcement values*
arltRebarsAndQuantity = 0x01 }	*Rebars + Quantity x (Length)*

Type of actual reinforcement labels

enum **ECriticalCombinationFormula** = {

ccfAuto = 0x00,	*Automatically selected combination type*
ccfCustom = 0x01,	*A custom combination type is enforced. The actual value is specified in a per window basis.*
ccfSemiAuto = 0x02 }	*Automatically selected ULS combination type, the SLS combination type is enforced. The actual value is specified in a per window basis.*

enum **ESmoothing** = {

smthNone = 0x00,	*None*
smthSelective = 0x01 ,	*Selective*
smthAllSurf = 0x02 }	*All surfaces*

Display smoothing of results

enum **EIntensityReferenceValue** = {

irvAbsMaxModel = 0x00,	*Absolute maximum of the entire model*
irvAbsMaxParts = 0x01 ,	*Absolute maximum of the current parts*
irvCustomVal = 0x02 }	*Custom value*

Intensity reference value

enum **EWindowSplit** = {

wsHorizontal = 0x00,	*Split windows Horizontally*
wsVertical = 0x01 }	*Split windows vertically*

Type of window splitting

enum **ECombinationMethod** = {

cmULS = 0x00,	*Conservative ULS combination formula*
cmULSab = 0x01 }	*Newer ULS combination formulas see EN 1990:6.10(a) and (b)*

Combination method in persistent and transient deign situations

enum **EWindowColourMode** = {

wcmColour = 0x00,	*Colour*

wcmGreyScale = 0x01, *Greyscale*
wcmBlackAndWhite = 0x02 } *Black and white*
Colour mode of the window

Error codes

enum **EWindowsError** = {

weLoadCaseIdOutOfBounds = –100001 *Load case index is not valid*
weLoadCombinationIdOutOfBounds = –100002 *Load combination index is not valid*
weEnvelopeIdOutOfBounds = –100003 *Envelope index is not valid*
}

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 valus : 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 = (

ELongBoolean	**Nodes**	*Show results for nodes*
ELongBoolean	**Lines**	*Show results for lines*
ELongBoolean	**Surfaces**	*Show results for surfaces*
ELongBoolean	**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*

EResultComponent	**ResultComponent**	*Result component*
double	**Scale**	*Scale of value on display*
EDisplayMode	**DisplayMode**	*Display mode*
EDisplayShape	**DisplayShape**	*Display shape*
RWriteValuesTo	**WriteValuesTo**	*Write values tothese types of elements*

) Basic display parameters

RBasicDisplayParameters_V153 = (

EResultComponent	**ResultComponent**	*Result component*
double	**Scale**	*Scale of value on display*
EDisplayMode	**DisplayMode**	*Display mode*
EDisplayShape	**DisplayShape**	*Display shape*
RWriteValuesTo	**WriteValuesTo**	*Write values tothese types of elements*
ELongBoolean	**AutoScale**	*Adjusts the diagram scale automaticaly*

) Basic display parameters

RExtendedDisplayParameters = (
Warning! *This record has become obsolete, it was superseded by RExtendedDisplayParameters_V153*

RBasicDisplayParameters	**BasicDispParams**	*Basic display parameters*
EDisplayAnalysisType	**DisplayAnalysisType**	*Analysis type of displayed results*
EResultType	**ResultType**	*Type of result to display*
EMinMaxType	**MinMaxType**	*Min or max value*
ECombinationType	**CriticalResCombinationType**	*For ResultType = rtEnvelope and rtCritical set combination type*
ELongBoolean	**SectPlaneContour**	*Contour of section plane is displayed if lbTrue*
EDisplayedEnvelopes	**DisplayedEnvelopes**	*Displayed envelopes*

) Extended display parameters

RExtendedDisplayParameters_V153 = (

RBasicDisplayParameters_V153	**BasicDispParams**	*Basic display parameters*
EDisplayAnalysisType	**DisplayAnalysisType**	*Analysis type of displayed results*
EResultType	**ResultType**	*Type of result to display*
EMinMaxType	**MinMaxType**	*Min or max value*
ECombinationType	**CriticalResCombinationType**	*For ResultType = rtEnvelope and rtCritical set combination type*
ELongBoolean	**SectPlaneContour**	*Contour of section plane is displayed if lbTrue*
EDisplayedEnvelopes	**DisplayedEnvelopes**	*Displayed envelopes*

) Extended display parameters

RShowGraphicSymbols = (

ELongBoolean	**Mesh**	*Show mesh*
ELongBoolean	**Node**	*Show node*
ELongBoolean	**SurfaceCentre**	*Show surface centre*
ELongBoolean	**CentreOfCircle**	*Show centre of circle*
ELongBoolean	**Domain**	*Show domains*
ELongBoolean	**NodalSupport**	*Show nodal supports*
ELongBoolean	**LineSupport**	*Show line supports*
ELongBoolean	**SurfaceSupport**	*Show surface support*
ELongBoolean	**Foundation**	*Show foundations*
ELongBoolean	**AutoFoundationDimension**	*Show foundation dimensions automatically*
ELongBoolean	**Links**	*Show links*
ELongBoolean	**Rigids**	*Show rigids*

ELongBoolean	**Diaphragm**	*Show diaphragms*
ELongBoolean	**Reference**	*Show references*
ELongBoolean	**CrossSectionShape**	*Show cross section shapes*
ELongBoolean	**EndReleases**	*Show end releases*
ELongBoolean	**StructuralMembers**	*Show structural members*
ELongBoolean	**ReinfParams**	*Show reinforcement parameters*
ELongBoolean	**ReinfDomain**	*Show reinforcement of domains*
ELongBoolean	**Mass**	*Show mass*
ELongBoolean	**StoreyCentGrav**	*Show storey's centre of gravity*
ELongBoolean	**StoreyShearCent**	*Show storey's centre of shear*
ELongBoolean	**ARBO_CRETelems**	*Show ARBO/CRET elements*
ELongBoolean	**COBIAXelems**	*Show COBIAX elements*
ELongBoolean	**Trusses**	*Show truss elements*
ELongBoolean	**Beams**	*Show beam elements*
ELongBoolean	**Ribs**	*Show rib elements*
ELongBoolean	**Springs**	*Show spring elements*
ELongBoolean	**IsolineLabels**	*Show labels on isolines*
ELongBoolean	**RoundIsoValues**	*Round values of isolines and isoareas*
ELongBoolean	**Gaps**	*Show gap elements*
ELongBoolean	**StructuralGrids**	*Show structural grids*

) Shown graphical symbol settings

RShowLocalSystems = (

ELongBoolean	**Beam**	*Show local coordinate systems of beams*
ELongBoolean	**Rib**	*Show local coordinate systems of ribs*
ELongBoolean	**Surface**	*Show local coordinate systems of surfaces*
ELongBoolean	**Domain**	*Show local coordinate systems of domains*
ELongBoolean	**Support**	*Show local coordinate systems of supports*
ELongBoolean	**Spring**	*Show local coordinate systems of springs*
ELongBoolean	**Gap**	*Show local coordinate systems of gaps*
ELongBoolean	**Link**	*Show local coordinate systems of links*
ELongBoolean	**EdgeHinge**	*Show local coordinate systems of edge hinges*

)

Show local coordination systems by element types

RShowLoads = (

ELongBoolean	**Concentrated**	*Show concentrated loads*
ELongBoolean	**Line**	*Show line loads*
ELongBoolean	**Surface**	*Show surface loads*
ELongBoolean	**Temperature**	*Show temperature loads*
ELongBoolean	**SelfWeight**	*Show self weight*
ELongBoolean	**Miscel**	*Show miscelaneous (other) loads*
ELongBoolean	**LoadDistrScheme**	*Show load distribution scheme*
ELongBoolean	**DerivedBeamLoad**	*Show derived beam loads*
ELongBoolean	**MovingLoadPhases**	*Show moving load phases*

)Show loads by types

RShowSymbols = (

RShowGraphicSymbols	**ShowGraphicSymbols**	*Show these graphic symbols*
RShowLocalSystems	**ShowLocalSystems**	*Show these local coordination systems*
RShowLoads	**ShowLoads**	*Show these loads*
ELongBoolean	**ObjectContours3D**	*Show 3D object contours*

)Shown symbols settings

RShowNumbering = (

ELongBoolean	**Node**	*Show node numbers*
ELongBoolean	**Truss**	*Show truss numbers*
ELongBoolean	**Beam**	*Show beam numbers*
ELongBoolean	**Rib**	*Show rib numbers*
ELongBoolean	**Surface**	*Show surface numbers*
ELongBoolean	**Domain**	*Show domain numbers*
ELongBoolean	**Support**	*Show support numbers*
ELongBoolean	**Links**	*Show link numbers*
ELongBoolean	**Rigid**	*Show rigid numbers*
ELongBoolean	**Diaphragm**	*Show diaphragm numbers*
ELongBoolean	**Spring**	*Show spring numbers*
ELongBoolean	**Gap**	*Show gap numbers*
ELongBoolean	**Material**	*Show material numbers*
ELongBoolean	**CrossSection**	*Show cross section numbers*
ELongBoolean	**Reference**	*Show reference numbers*
ELongBoolean	**ARBO_CRETelems**	*Show arbo/cret element numbers*
ELongBoolean	**DesignGroup**	*Show design group numbers*
ELongBoolean	**OptimisationGroup**	*Show optimisation group numbers*

)Shown numbers by element types

RShowProperties = (

ELongBoolean	**MaterialName**	*Show material name*
ELongBoolean	**CrossSectName**	*Show cross section name*
ELongBoolean	**BoltedJoint**	*Show bolted joints*
ELongBoolean	**ColumnReinf**	*Show column reinforcement*
ELongBoolean	**BeamLength**	*Show beam lengths*
ELongBoolean	**Thickness**	*Show domain(surface) thickness*
ELongBoolean	**DomainArea**	*Show domain area*
ELongBoolean	**COBIAXlabels**	*Show COBIAX labels*
ELongBoolean	**LoadValue**	*Show load value*
ELongBoolean	**MassValue**	*Show mass value*
ELongBoolean	**Units**	*Show units*
ELongBoolean	**ConcentratedLoadValue**	*Show value of concentrated loads*
ELongBoolean	**LineLoadValue**	*Show value of line loads*
ELongBoolean	**SurfaceLoadValue**	*Show value of surface loads*
ELongBoolean	**TemperatureLoadValue**	*Show value of temperature loads*
ELongBoolean	**SelfWeightValue**	*Show value of self weight*
ELongBoolean	**OtherLoadValue**	*Show value of all other types of loads*

)

Shown properties

RShowActualReinforcement = (

ELongBoolean	**Symbol_axb**	*Show Symbol of actual bottom reinforcement in x direction*
ELongBoolean	**Symbol_ayb**	*Show Symbol of actual bottom reinforcement in y direction*
ELongBoolean	**Symbol_axt**	*Show Symbol of actual top reinforcement in x direction*
ELongBoolean	**Symbol_ayt**	*Show Symbol of actual top reinforcement in y direction*
ELongBoolean	**Label_axb**	*Show Symbol of actual bottom reinforcement in x direction*
ELongBoolean	**Label_ayb**	*Show Symbol of actual bottom reinforcement in y direction*
ELongBoolean	**Label_axt**	*Show Symbol of actual top reinforcement in x direction*
ELongBoolean	**Label_ayt**	*Show Symbol of actual top reinforcement in y direction*
EActualReinforcementLabelType	**ActReinfLabelType**	*Type of actual reinforcement label*
ELongBoolean	**AccordResComponent**	*According to displayed result component*

)

Show symbol/label of actual reinforcement

RShowLabels = (

RShowNumbering	**ShowNumbering**	*Shown numbers by element types*
RShowProperties	**ShowProperties**	*Shown properties*
RShowActualReinforcement	**ShowActualReinf**	*Show symbol/label of actual reinforcement*
ELongBoolean	**UseFiniteElements**	*Use finite element numbers. More info here.*
ELongBoolean	**LabelsOnLines**	*Labels on lines seen from axis direction*

)

Show numbering and properties of various elements

RInfoWindowSwitch = (

ELongBoolean	**Coordinates**	*Show coordinates window*
ELongBoolean	**Info**	*Show info window*
ELongBoolean	**ColourCoding**	*Show colour coding window*
ELongBoolean	**ColourLegend**	*Show colour legend window*

)

Show or hide windows

RDisplaySwitch = (

ELongBoolean	**Parts**	*Show Parts*
ELongBoolean	**NonVisiblePartsGreyed**	*Show NonVisiblePartsGreyed*
ELongBoolean	**Guidlines**	*Show Guidlines*
ELongBoolean	**StructuralGrid**	*Show StructuralGrid*

)

Display switch

RShowSwitches = (

RInfoWindowSwitch	**InfoWindowSwitch**	*Show or hide windows*
RDisplaySwitch	**DisplaySwitch**	*Display switch*

)

Show windows and display switch

RCommonCriticalResultsSettings = (

ELongBoolean	**InvestigateAllCombos**	*Investigate all combinations*
ECriticalCombinationFormula	**CritComboFormula**	*Crittical combination formula*

ECombinationMethod	**InPersistentAndTransientDesign Situations**	*Combination method in persistent and transient des. situations*

)

Display setting for critical results

RCommonCriticalResultsSettings_V161 = (

ELongBoolean	**InvestigateAllCombos**	*Investigate all combinations*
ECriticalCombinationFormula	**CritComboFormula**	*Crittical combination formula*
ECombinationMethod	**InPersistentAndTransientDesign Situations**	*Combination method in persistent and transient des. situations*
ECombinationType	**SemiAutoSLS**	*type of semi-auto SLS override (must be one of ctSemiAutoSLS1, ctSemiAutoSLS2, ctSemiAutoSLS3)*

)

Display setting for critical results

RMiscellaneousSettings = (

ESmoothing	**Smoothing**	*Smoothing*
double	**MaxAngleLocZ**	*Maximum angle allowed between local z axes in degrees*
double	**MaxAngleLocX**	*Maximum angle allowed between local x axes in degrees*
EIntensityReferenceValue	**IntensityRefVal**	*Intensity reference value*
double	**CustomVal**	*Used when IntensityRefVal= irvCustomVal*
ELongBoolean	**SupportsAvgResultsPerMember**	*Show average results of supports per member*

)

Miscellaneous settings

RCommonDisplayParameters = (

RCommonCriticalResultsSettings	**CriticalResSettings**	Critical results settings when *ExtendedDisplayParameters.ResultType = rtCritical*
ELongBoolean	**CutMomentPeaks**	*Cut moment peaks*
ELongBoolean	**DrawInPlane**	*Draw in plane*
RMiscellaneousSettings	**MiscelSettings**	*Miscelsettings*

)

Display parameters, which are common for all AxisVM tabs.

RCommonDisplayParameters_V161 = (

RCommonCriticalResultsSettings_V161	**CriticalResSettings**	Critical results settings when *ExtendedDisplayParameters.ResultType = rtCritical*
ELongBoolean	**CutMomentPeaks**	*Cut moment peaks*
ELongBoolean	**DrawInPlane**	*Draw in plane*
RMiscellaneousSettings	**MiscelSettings**	*Miscelsettings*

)

Display parameters, which are common for all AxisVM tabs.

RWorldRectangle = (

double	**Left**	*distance of left side of a rectangle from the origin*
double	**Right**	*distance of right side of a rectangle from the origin*
double	**Top**	*distance of top side of a rectangle from the origin*
double	**Bottom**	*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	*Index ot the window* *0<Index<IAxisVMWindows.Count*
WindowSplit	*Type of window splitting*
SplitPos	*Split position (ratio) of the window, valid number: 0,1 - 0,9*

Splits window to two. If successful returns IAxisVMWindows.Count*, otherwise an error code (errDatabaseNotReady or errIndexOutOfBounds).*

---

long **GetCommonDisplayParameters** ([i/o] RCommonDisplayParameters **CommonDisplayParameters**)

CommonDisplayParameters	*Display parameters which are common for all AxisVM tabs.*

Get common display parameters. If successful returns 1, otherwise an error code (errDatabaseNotReady).

---

long **GetCommonDisplayParameters_V161** ([i/o] RCommonDisplayParameters_V161 **CommonDisplayParameters**)

CommonDisplayParameters	*Display parameters which are common for all AxisVM tabs.*

Get common display parameters. If successful returns 1, otherwise an error code (errDatabaseNotReady).

---

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	*Index ot 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 ot the mode shape*
SectionIds	*Index of displayed sections*

Get display parameters of buckling results. If successful returns 1, otherwise an error code (errDatabaseNotReady or errIndexOutOfBounds).

---

long **GetBucklingDisplayParameters_V153** ([in] long **Index,** [i/o] RExtendedDisplayParameters_V153 **ExtendedDisplayParameters,** [out] long **LoadCaseOrCombinationId,** [out] long **ModeShapeId,** [out] SAFEARRAY(long)* **SectionIds**)

Index	*Index ot 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 ot the mode shape*
SectionIds	*Index of displayed sections*

Get display parameters of buckling results. If successful returns 1, otherwise an error code (errDatabaseNotReady or errIndexOutOfBounds).

---

long **GetDisplayOptions** ([in] long **Index,** [i/o] RShowSymbols **ShowSymbols,**
[i/o] RShowLabels **ShowLabels,** [i/o] RShowSwitches **ShowSwitches**)

Index	*Index ot the window* *0<Index<IAxisVMWindows.Count*
ShowSymbols	*Shown symbols settings*
ShowLabels	*Show numbering and properties of various elements*
ShowSwitches	*Show windows and display switch*

Get display options. If successful returns 1, otherwise an error code (errDatabaseNotReady or errIndexOutOfBounds). For a full low level access, or querrying directly only a few switches, use the more universal Switch property instead.

---

long **GetDefaultDisplayOptions** ([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*

Get default display options of AxisVM. If successful returns 1, otherwise an error code (errDatabaseNotReady).

---

long **GetDetectedLayerIDs** ([in] long **Index,** [out] SAFEARRAY(long)* **DetectedLayerIDs**)

Index	*Index ot the window* *0<Index<IAxisVMWindows.Count*
DetectedLayerIDs	*Indexes of layers which can be detected with mouse*

Get indexes of detected layers. If successful returns Index, otherwise an error code (errDatabaseNotReady or errIndexOutOfBounds).

---

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	*Index ot the window* *0<Index<IAxisVMWindows.Count*
ExtendedDisplayParameters	*Extended display parameters*
DynLoadCaseOrEnvelopeId	*Dynamic load case index or envelope index depending on the ExtendedDisplayParameters.ResultType*
TimeStepId	*Index ot the time step*
SectionIds	*Index of displayed sections*

Get display parameters of dynamic results. If successful returns 1, otherwise an error code (errDatabaseNotReady or errIndexOutOfBounds).

---

long **GetDynamicDisplayParameters_V153** ([in] long **Index,** [i/o] RExtendedDisplayParameters_V153 **ExtendedDisplayParameters,** [out] long **DynLoadCaseOrEnvelopeId,** [out] long **TimeStepId,**
[out] SAFEARRAY(long)* **SectionIds**)

Index	*Index ot the window* *0<Index<IAxisVMWindows.Count*
ExtendedDisplayParameters	*Extended display parameters*
DynLoadCaseOrEnvelopeId	*Dynamic load case index or envelope index depending on the ExtendedDisplayParameters.ResultType*
TimeStepId	*Index ot the time step*
SectionIds	*Index of displayed sections*

Get display parameters of dynamic results. If successful returns 1, otherwise an error code (errDatabaseNotReady or errIndexOutOfBounds).

---

long **GetLockedLayerIDs** ([in] long **Index,** [out] SAFEARRAY(long)* **LockedLayerIDs**)

Index	*Index ot the window* *0<Index<IAxisVMWindows.Count*
LockedLayerIDs	*Indexes of layers which are locked*

Get indexes of locked layers. If successful returns Index, otherwise an error code (errDatabaseNotReady or errIndexOutOfBounds).

---

long **GetRCDesignDisplayParameters** ([in] long **Index,** [i/o] RExtendedDisplayParameters **ExtendedDisplayParameters,** [out] long L**oadCaseOrCombinationOrEnvelopeId,**
[out] SAFEARRAY(long)* **SectionIds**)

Warning! *This function has become obsolete, was superseded by GetRCDesignDisplayParameters_V153*

Index	*Index ot the window* *0<Index<IAxisVMWindows.Count*
ExtendedDisplayParameters	*Extended display parameters*
LoadCaseOrCombinationOrEnvelopeId	*Load case,load combination or envelope index depending on the ExtendedDisplayParameters.ResultType*
SectionIds	*Index of displayed sections*
ExtendedDisplayParameters	*Extended display parameters*

Get display parameters of RC design results. If successful returns 1, otherwise an error code (errDatabaseNotReady or errIndexOutOfBounds).

---

long **GetRCDesignDisplayParameters_V153** ([in] long **Index,** [i/o] RExtendedDisplayParameters_V153 **ExtendedDisplayParameters,** [out] long L**oadCaseOrCombinationOrEnvelopeId,**
[out] SAFEARRAY(long)* **SectionIds**)

Index	*Index ot the window* *0<Index<IAxisVMWindows.Count*
ExtendedDisplayParameters	*Extended display parameters*
LoadCaseOrCombinationOrEnvelopeId	*Load case,load combination or envelope index depending on the ExtendedDisplayParameters.ResultType*
SectionIds	*Index of displayed sections*
ExtendedDisplayParameters	*Extended display parameters*

Get display parameters of RC design results. If successful returns 1, otherwise an error code (errDatabaseNotReady or errIndexOutOfBounds).

---

long **GetResultCase** ([in] long **Index,** [i/o] RResultCase **ResultCase**)

Index	*Index ot the window* *0<Index<IAxisVMWindows.Count*
ResultCase	*Result case descriptor*

Gets the result case record, according to the one in the result case selector drop down menu. If it is called on a tab that doesn't have a result case drop down menu, the returned ResultCase is unspecified. If successful returns the Index, otherwise an error code (errDatabaseNotReady or errIndexOutOfBounds).

---

long **GetStaticDisplayParameters** ([in] long **Index,** [i/o] RExtendedDisplayParameters **ExtendedDisplayParameters,** [out] long L**oadCaseOrCombinationOrEnvelopeId,**
[out] SAFEARRAY(long)* **SectionIds**)

Warning! *This function has become obsolete, was superseded by GetRCDesignDisplayParameters_V153*

Index	*Index ot the window* *0<Index<IAxisVMWindows.Count*
ExtendedDisplayParameters	*Extended display parameters*
LoadCaseOrCombinationOrEnvelopeId	*Load case,load combination or envelope index depending on the ExtendedDisplayParameters.ResultType*
SectionIds	*Index of displayed sections*

Get display parameters of (static) analysis results. If successful returns 1, otherwise an error code (errDatabaseNotReady or errIndexOutOfBounds).

---

long **GetStaticDisplayParameters_V153** ([in] long **Index,** [i/o] RExtendedDisplayParameters_V153 **ExtendedDisplayParameters,** [out] long L**oadCaseOrCombinationOrEnvelopeId,**
[out] SAFEARRAY(long)* **SectionIds**)

Index	*Index ot the window* *0<Index<IAxisVMWindows.Count*
ExtendedDisplayParameters	*Extended display parameters*
LoadCaseOrCombinationOrEnvelopeId	*Load case,load combination or envelope index depending on the ExtendedDisplayParameters.ResultType*
SectionIds	*Index of displayed sections*

Get display parameters of (static) analysis results. If successful returns 1, otherwise an error code (errDatabaseNotReady or errIndexOutOfBounds).

---

long **GetSteelDesignDisplayParameters** ([in] long **Index,** [i/o] RExtendedDisplayParameters **ExtendedDisplayParameters,** [out] long L**oadCaseOrCombinationOrEnvelopeId,**
[out] SAFEARRAY(long)* **SectionIds**)

Warning! *This function has become obsolete, was superseded by GetSteelDesignDisplayParameters_V153*

Index	*Index ot the window* *0<Index<IAxisVMWindows.Count*
ExtendedDisplayParameters	*Extended display parameters*
LoadCaseOrCombinationOrEnvelopeId	*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 L**oadCaseOrCombinationOrEnvelopeId,**
[out] SAFEARRAY(long)* **SectionIds**)

Index	*Index ot the window* *0<Index<IAxisVMWindows.Count*
ExtendedDisplayParameters	*Extended display parameters*
LoadCaseOrCombinationOrEnvelopeId	*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 **LoadCaseOrCombinationOrEnvelopeId,**
[out] SAFEARRAY(long)* **SectionIds**)

Warning! *This function has become obsolete, was superseded by GetTimberDesignDisplayParameters_V153*

Index	*Index ot the window* *0<Index<IAxisVMWindows.Count*
ExtendedDisplayParameters	*Extended display parameters*
LoadCaseOrCombinationOrEnvelopeId	*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 **LoadCaseOrCombinationOrEnvelopeId,**
[out] SAFEARRAY(long)* **SectionIds**)

Index	*Index ot the window* *0<Index<IAxisVMWindows.Count*
ExtendedDisplayParameters	*Extended display parameters*
LoadCaseOrCombinationOrEnvelopeId	*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 ot the window* *0<Index<IAxisVMWindows.Count*
ExtendedDisplayParameters	*Extended display parameters*
LoadCaseOrCombinationId	*Load case or combination index depending on the ExtendedDisplayParameters.ResultType*
ModeShapeId	*Index ot 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 ot the window* *0<Index<IAxisVMWindows.Count*
ExtendedDisplayParameters	*Extended display parameters*
LoadCaseOrCombinationId	*Load case or combination index depending on the ExtendedDisplayParameters.ResultType*
ModeShapeId	*Index ot 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 ot 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 ot 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 ot 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 ot 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 *Index ot the window. 0<Index<IAxisVMWindows.Count*

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 **ReDraw**

Redraws all windows in AxisVM. If successful returns number of windows, otherwise an error code (errDatabaseNotReady or errIndexOutOfBounds).

---

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

Index *Index ot the window*
0<Index<IAxisVMWindows.Count

Deletes window. If successful returns Index, otherwise an error code (errDatabaseNotReady or errIndexOutOfBounds).

---

long **SaveWindowToBitmap** ([in] long **Index,** [in] EWindowColourMode **WindowColourMode,** [in] BSTR **FileName**)

Index *Index ot the window. 0<Index<IAxisVMWindows.Count*

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 **SaveWindowsToBitmap** ([in] EWindowColourMode **WindowColourMode,** [in] BSTR **FileName**)

WindowColourMode *Colour mode*

FileName *file name with full path including extension*

Save the whole desktop of AxisVM to a bitmap file. If successful returns 1.

---

long **SaveWindowToClipboard** ([in] long **Index,** [in] EWindowColourMode **WindowColourMode**)

Index *Index ot the window*
0<Index<IAxisVMWindows.Count

WindowColourMode *Colour mode*

Save window to the clipboard. If successful returns Index, otherwise an error code (errIndexOutOfBounds).

---

long **SaveWindowsToClipboard** ([in] EWindowColourMode **WindowColourMode**)

WindowColourMode *Colour mode*

Save the whole desktop of AxisVM to the clipboard. If successful returns 1.

---

long **SaveWindowToMetafile** ([in] long **Index,** [in] BSTR **FileName**)

Index *Index ot the window*
0<Index<IAxisVMWindows.Count

FileName *file name with full path including extension*

Save window to a meta file. If successful returns Index, otherwise an error code (errIndexOutOfBounds).

---

long **SaveWindowsToMetafile** ([in] BSTR **FileName**)

FileName *file name with full path including extension*

Save the whole desktop of AxisVM to a meta file. If successful returns 1.

---

long **SetCommonDisplayParameters** ([i/o] RCommonDisplayParameters **CommonDisplayParameters**)

CommonDisplayParameters *Display parameters which are common for all AxisVM tabs.*

Set common display parameters. If successful returns 1, otherwise an error code (errDatabaseNotReady).

---

long **SetCommonDisplayParameters_V161** ([i/o] RCommonDisplayParameters_V161 **CommonDisplayParameters**)

CommonDisplayParameters *Display parameters which are common for all AxisVM tabs.*

Set common display parameters. If successful returns 1, otherwise an error code (errDatabaseNotReady).

---

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 ot 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 ot 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 ot 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 ot 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 ot 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	*Index ot the window* *0<Index<IAxisVMWindows.Count*
ShowSymbols	*Shown symbols settings*
ShowLabels	*Show numbering and properties of various elements*
ShowSwitches	*Show windows and display switch*

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.

---

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	*Index ot the window* *0<Index<IAxisVMWindows.Count*
ExtendedDisplayParameters	*Extended display parameters*
DynLoadCaseOrEnvelopeId	*Dynamic load case index or envelope index depending on the ExtendedDisplayParameters.ResultType (only rtLoadCase and rtEnvelope are valid types)*
TimeStepId	*Index ot the time step*
SectionIds	*Index of displayed sections*

Set display parameters of dynamic results. If successful returns 1, otherwise an error code (errDatabaseNotReady, weLoadCaseIdOutOfBounds, weEnvelopeIdOutOfBounds or errIndexOutOfBounds).

---

long **SetDynamicDisplayParameters_V153** ([in] long **Index,** [i/o] RExtendedDisplayParameters_V153 **ExtendedDisplayParameters,** [in] long **DynLoadCaseOrEnvelopeId,** [in] long **TimeStepId,** [in] SAFEARRAY(long)* **SectionIds**)

Index	*Index ot the window* *0<Index<IAxisVMWindows.Count*
ExtendedDisplayParameters	*Extended display parameters*
DynLoadCaseOrEnvelopeId	*Dynamic load case index or envelope index depending on the ExtendedDisplayParameters.ResultType (only rtLoadCase and rtEnvelope are valid types)*
TimeStepId	*Index ot the time step*
SectionIds	*Index of displayed sections*

Set display parameters of dynamic results. If successful returns 1, otherwise an error code (errDatabaseNotReady, weLoadCaseIdOutOfBounds, weEnvelopeIdOutOfBounds or errIndexOutOfBounds).

---

long **SetLockedLayerIDs** ([in] long **Index,** [i/o] SAFEARRAY(long)* **LockedLayerIDs**)

Index	*Index ot the window* *0<Index<IAxisVMWindows.Count*
LockedLayerIDs	*Indexes of layers which are locked*

Set indexes of locked layers. If successful returns Index, otherwise an error code (errDatabaseNotReady or errIndexOutOfBounds).

---

long **SetRCDesignDisplayParameters** ([in] long **Index,** [i/o] RExtendedDisplayParameters **ExtendedDisplayParameters,** [in] long **LoadCaseOrCombinationOrEnvelopeId,** [in] SAFEARRAY(long)* **SectionIds**)

Warning! *This function has become obsolete, was superseded by SetRCDesignDisplayParameters_V153*

Index	*Index ot the window* *0<Index<IAxisVMWindows.Count*
ExtendedDisplayParameters	*Extended display parameters*
LoadCaseOrCombinationOrEnvelopeId	*Load case,load combination or envelope index depending on the ExtendedDisplayParameters.ResultType* *0 for rtCritical*
SectionIds	*Index of displayed sections*

Set display parameters of RC design results. If successful returns 1, otherwise an error code (errDatabaseNotReady, weLoadCaseIdOutOfBounds, weLoadCombinationIdOutOfBounds, weEnvelopeIdOutOfBounds or errIndexOutOfBounds).

---

long **SetRCDesignDisplayParameters_V153** ([in] long **Index,** [i/o] RExtendedDisplayParameters_V153 **ExtendedDisplayParameters,** [in] long **LoadCaseOrCombinationOrEnvelopeId,** [in] SAFEARRAY(long)* **SectionIds**)

Index	*Index ot the window* *0<Index<IAxisVMWindows.Count*
ExtendedDisplayParameters	*Extended display parameters*
LoadCaseOrCombinationOrEnvelopeId	*Load case,load combination or envelope index depending on the ExtendedDisplayParameters.ResultType* *0 for rtCritical*
SectionIds	*Index of displayed sections*

Set display parameters of RC design results. If successful returns 1, otherwise an error code (errDatabaseNotReady, weLoadCaseIdOutOfBounds, weLoadCombinationIdOutOfBounds, weEnvelopeIdOutOfBounds or errIndexOutOfBounds).

---

long **SetStaticDisplayParameters** ([in] long **Index,** [i/o] RExtendedDisplayParameters **ExtendedDisplayParameters,** [in] long **LoadCaseOrCombinationOrEnvelopeId,**
[in] SAFEARRAY(long)* **SectionIds**)

Warning! *This function has become obsolete, was superseded by SetStaticDisplayParameters_V153*

Index	*Index ot the window* *0<Index<IAxisVMWindows.Count*
ExtendedDisplayParameters	*Extended display parameters*
LoadCaseOrCombinationOrEnvelopeId	*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, weLoadCaseIdOutOfBounds, weLoadCombinationIdOutOfBounds, weEnvelopeIdOutOfBounds or errIndexOutOfBounds).

---

long **SetStaticDisplayParameters_V153** ([in] long **Index,** [i/o] RExtendedDisplayParameters_V153 **ExtendedDisplayParameters,** [in] long **LoadCaseOrCombinationOrEnvelopeId,**
[in] SAFEARRAY(long)* **SectionIds**)

Index	*Index ot the window* *0<Index<IAxisVMWindows.Count*
ExtendedDisplayParameters	*Extended display parameters*
LoadCaseOrCombinationOrEnvelopeId	*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, weLoadCaseIdOutOfBounds, weLoadCombinationIdOutOfBounds, weEnvelopeIdOutOfBounds or errIndexOutOfBounds).

---

long **SetSteelDesignDisplayParameters** ([in] long **Index,** [i/o] RExtendedDisplayParameters **ExtendedDisplayParameters,** [in] long **LoadCaseOrCombinationOrEnvelopeId ,**
[in] SAFEARRAY(long)* **SectionIds**)

Warning! *This function has become obsolete, was superseded by SetSteelDesignDisplayParameters_V153*

Index	*Index ot the window* *0<Index<IAxisVMWindows.Count*
ExtendedDisplayParameters	*Extended display parameters*
LoadCaseOrCombinationOrEnvelopeId	*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, weLoadCaseIdOutOfBounds, weLoadCombinationIdOutOfBounds, weEnvelopeIdOutOfBounds or errIndexOutOfBounds).

---

long **SetSteelDesignDisplayParameters_V153** ([in] long **Index,** [i/o] RExtendedDisplayParameters_V153 **ExtendedDisplayParameters,** [in] long **LoadCaseOrCombinationOrEnvelopeId ,**
[in] SAFEARRAY(long)* **SectionIds**)

Index	*Index ot the window* *0<Index<IAxisVMWindows.Count*
ExtendedDisplayParameters	*Extended display parameters*
LoadCaseOrCombinationOrEnvelopeId	*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, weLoadCaseIdOutOfBounds, weLoadCombinationIdOutOfBounds, weEnvelopeIdOutOfBounds or errIndexOutOfBounds).

---

long **SetTimberDesignDisplayParameters** ([in] long **Index,** [i/o] RExtendedDisplayParameters **ExtendedDisplayParameters,** [in] long **LoadCaseOrCombinationOrEnvelopeId ,**
[in] SAFEARRAY(long)* **SectionIds**)

Warning! *This function has become obsolete, was superseded by SetTimberDesignDisplayParameters_V153*

Index	*Index ot the window* *0<Index<IAxisVMWindows.Count*
ExtendedDisplayParameters	*Extended display parameters*
LoadCaseOrCombinationOrEnvelopeId	*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, weLoadCaseIdOutOfBounds, weLoadCombinationIdOutOfBounds, weEnvelopeIdOutOfBounds or errIndexOutOfBounds).

---

long **SetTimberDesignDisplayParameters_V153** ([in] long **Index,** [i/o] RExtendedDisplayParameters_V153 **ExtendedDisplayParameters,** [in] long **LoadCaseOrCombinationOrEnvelopeId ,**
[in] SAFEARRAY(long)* **SectionIds**)

Index	*Index ot the window* *0<Index<IAxisVMWindows.Count*
ExtendedDisplayParameters	*Extended display parameters*
LoadCaseOrCombinationOrEnvelopeId	*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, weLoadCaseIdOutOfBounds, weLoadCombinationIdOutOfBounds, weEnvelopeIdOutOfBounds or errIndexOutOfBounds).

---

long **SetVibrationDisplayParameters** ([in] long **Index,** [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*

Index	*Index ot the window* *0<Index<IAxisVMWindows.Count*
ExtendedDisplayParameters	*Extended display parameters*
LoadCaseOrCombinationId	*Load case or combination index depending on the ExtendedDisplayParameters.ResultType (only rtLoadCase and rtLoadCombination are valid)*
ModeShapeId	*Index ot the mode shape*
SectionIds	*Index of displayed sections*

Set display parameters of vibration results. If successful returns 1, otherwise an error code (errDatabaseNotReady, weLoadCaseIdOutOfBounds, weLoadCombinationIdOutOfBounds or errIndexOutOfBounds).

---

long **SetVibrationDisplayParameters_V153** ([in] long **Index,** [i/o] RExtendedDisplayParameters_V153 **ExtendedDisplayParameters,** [in] long **LoadCaseOrCombinationId,** [in] long **ModeShapeId,** [in] SAFEARRAY(long)* **SectionIds**)

Index	*Index ot the window* *0<Index<IAxisVMWindows.Count*
ExtendedDisplayParameters	*Extended display parameters*
LoadCaseOrCombinationId	*Load case or combination index depending on the ExtendedDisplayParameters.ResultType (only rtLoadCase and rtLoadCombination are valid)*
ModeShapeId	*Index ot the mode shape*
SectionIds	*Index of displayed sections*

Set display parameters of vibration results. If successful returns 1, otherwise an error code (errDatabaseNotReady, weLoadCaseIdOutOfBounds, weLoadCombinationIdOutOfBounds or errIndexOutOfBounds).

---

long **SetVisibleLayerIDs** ([in] long **Index,** [i/o] SAFEARRAY(long)* **VisibleLayerIDs**)

Index	*Index ot the window* *0<Index<IAxisVMWindows.Count*
VisibleLayerIDs	*Indexes of visible layers*

Set indexes of visible layers. If successful returns Index, otherwise an error code (errDatabaseNotReady or errIndexOutOfBounds).

---

long **SetVisibleSectionIDs** ([in] long **Index,** [i/o] SAFEARRAY(long)* **SectionIDs**)

Index	*Index ot 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 ot 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 ot 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 ot 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.*
ELongBoolean	**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 **DynLoadCaseOrEnvelopeId,** [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 **DynLoadCaseOrEnvelopeId,** [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 **LoadCaseOrCombinationOrEnvelopeId,** [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 **LoadCaseOrCombinationOrEnvelopeId,** [out] SAFEARRAY(long)* **SectionIds**)

See GetRCDesignDisplayParameters of IAxisVMWindows

---

long **GetResultCase** ([i/o] RResultCase **ResultCase**)

ResultCase *Result case descriptor*

Gets the result case record, according to the one in the result case selector drop down menu. If it is called on a tab that doesn't have a result case drop down menu, the returned ResultCase is unspecified. If successful returns the index of the window, otherwise an error code (errDatabaseNotReady or errIndexOutOfBounds).

---

long **GetStaticDisplayParameters (**[i/o] RExtendedDisplayParameters **ExtendedDisplayParameters,**
[out] long L**oadCaseOrCombinationOrEnvelopeId,** [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 L**oadCaseOrCombinationOrEnvelopeId,** [out] SAFEARRAY(long)* **SectionIds**)

See GetStaticDisplayParameters of IAxisVMWindows

---

long **GetSteelDesignDisplayParameters** ([i/o] RExtendedDisplayParameters **ExtendedDisplayParameters,** [out] long L**oadCaseOrCombinationOrEnvelopeId,**
[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 L**oadCaseOrCombinationOrEnvelopeId,**
[out] SAFEARRAY(long)* **SectionIds**)

See GetSteelDesignDisplayParameters of IAxisVMWindows

---

long **GetTimberDesignDisplayParameters** ([i/o] RExtendedDisplayParameters **ExtendedDisplayParameters,** [out] long **LoadCaseOrCombinationOrEnvelopeId,**
[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 **LoadCaseOrCombinationOrEnvelopeId,**
[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 *Indexes of layers which can be detected by mouse*
Set indexes of detected layers. If successful returns window index, otherwise an error code (errDatabaseNotReady).

---

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 **DynLoadCaseOrEnvelopeId,** [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 **DynLoadCaseOrEnvelopeId,** [in] long **TimeStepId,** [in] SAFEARRAY(long)* **SectionIds**)
See SetDynamicDisplayParameters of IAxisVMWindows

---

long **SetLockedLayerIDs** ([i/o] SAFEARRAY(long)* **LockedLayerIDs**)
LockedLayerIDs *Indexes of locked layers*
Set indexes of locked layers. If successful returns window index, otherwise an error code (errDatabaseNotReady).

---

long **SetRCDesignDisplayParameters** ([i/o] RExtendedDisplayParameters **ExtendedDisplayParameters,**
[in] long L**oadCaseOrCombinationOrEnvelopeId,** [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 L**oadCaseOrCombinationOrEnvelopeId,** [in] SAFEARRAY(long)* **SectionIds**)
See SetRCDesignDisplayParameters of IAxisVMWindows

---

long **SetStaticDisplayParameters** ([i/o] RExtendedDisplayParameters **ExtendedDisplayParameters,**
[in] long L**oadCaseOrCombinationOrEnvelopeId,** [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 L**oadCaseOrCombinationOrEnvelopeId,** [in] SAFEARRAY(long)* **SectionIds**)
See SetStaticDisplayParameters of IAxisVMWindows

---

long **SetSteelDesignDisplayParameters** ([i/o] RExtendedDisplayParameters **ExtendedDisplayParameters,**
[in] long **LoadCaseOrCombinationOrEnvelopeId,** [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 **LoadCaseOrCombinationOrEnvelopeId,** [in] SAFEARRAY(long)* **SectionIds**)
See SetSteelDesignDisplayParameters of IAxisVMWindows

---

long **SetTimberDesignDisplayParameters** ([i/o] RExtendedDisplayParameters **ExtendedDisplayParameters,**
[in] long **LoadCaseOrCombinationOrEnvelopeId,** [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 **LoadCaseOrCombinationOrEnvelopeId,** [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,	*Global XY plane*
gwptXZ = 0x01,	*Global XZ plane*
gwptYZ = 0x02 }	*Global YZ plane*

Global plane of the workplane

enum **ESmartWorkplaneElementType** = {

swetMember = 0x00,	*XY plane made of member's local x and y axis*
swetSurface = 0x01,	*XY plane made of surface's local x and y axis*
swetDomain = 0x02 }	*XY plane made of domain's local x and y axis*

Plane of an existing element (member, surface, domain)

enum **EWorkplaneType** = {

wptGlobal = 0x00,	*Global work plane*
wptSmart= 0x01,	*Smart work plane*
wptGeneral = 0x02 }	*General work plane*

Type of the work plane

Error codes

enum **EWorkplanesError** = {

wpeInvalidName = –100001	*Name of the work plane is not valid*
wpeNameAlreadyExists = –100002	*Work plane with the same name already exists in the model*
wpeInvalidWorkPlaneParameters = –100003	*One or more parameters of the work plane are not valid*
wpeWorkplaneIsNotGlobal = –100004	*Type of the subject worklpane is not global*
wpeWorkplaneIsNotSmart = –100005	*Type of the subject worklpane is not smart*
wpeWorkplaneIsNotGeneral = –100006	*Type of the subject worklpane is not general*

}

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	*Name of the new work plane*
Origin	*Coordinates of the origin*
LocalX	*Vector for work plane's x axis*
LocalY	*Vector for work plane's y axis*
HideNotInPlane	*Hide elements which are not in the work plane*
ShowGrayedNotInPlane	*Show elements which are not in the work plane grayed*

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	*Name of the new workplane*
GlobalWorkplaneType	*Global plane of the new work plane*
PlaneOffset	*Distance of the work planes's origin from global plane*
HideNotInPlane	*Hide elements which are not in the work plane*
ShowGrayedNotInPlane	*Show elements which are not in the work plane grayed*

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	*Name of the new work plane*
SmartWorkplaneElementType	*Type of the element used for the work plane*
ElementIndex	*index of the element*
HideNotInPlane	*Hide elements which are not in the work plane*
ShowGrayedNotInPlane	*Show elements which are not in the work plane grayed*

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	*Index of the general work plane*
Origin	*Coordinates of the origin*
LocalX	*Vector for work plane's x axis*
LocalY	*Vector for work plane's y axis*

Get parameters of the general work plane. If successful, returns the work plane index otherwise an error code (errDatabaseNotReady, errIndexOutOfBounds, wpeWorkplaneIsNotGeneral)

---

long **GetGlobalParameters** ([in] long **Index,** [out] EGlobalWorkplaneType **GlobalWorkplaneType,**
[out] double **PlaneOffset**)

Index	*Index of the global work plane*
GlobalWorkplaneType	*Global plane of the new work plane*
PlaneOffset	*Distance of the work planes's origin from global plane*

Get parameters of the global work plane. If successful, returns the work plane index otherwise an error code (errDatabaseNotReady, errIndexOutOfBounds, wpeWorkplaneIsNotGlobal)

---

long **GetSmartParameters** ([in] long **Index,** [out] ESmartWorkplaneElementType **SmartWorkplaneElementType,** [out] long **ElementIndex**)

Index *Index of the global work plane*
SmartWorkplaneElementType *Type of the element used for the work plane*
ElementIndex *index of the element*

Get parameters of the smart work plane. If successful, returns the work plane index otherwise an error code (errDatabaseNotReady, errIndexOutOfBounds, wpeWorkplaneIsNotSmart)

---

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

Index *Index of the global work plane*

Delete the work plane. If successful, returns 1, otherwise an error code (errDatabaseNotReady, errIndexOutOfBounds)

---

long **SetGeneralParameters** ([in] long **Index,** [i/o] Rpoint3D **Origin,** [i/o] Rpoint3D **LocalX,** [i/o] Rpoint3D **LocalY**)

Index *Index of the general work plane*
Origin *Coordinates of the origin*
LocalX *Vector for work plane's x axis*
LocalY *Vector for work plane's y axis*

Set parameters of the general work plane. If successful, returns the work plane index otherwise an error code (errDatabaseNotReady, errIndexOutOfBounds, wpeWorkplaneIsNotGeneral)

---

long **SetGlobalParameters** ([in] long **Index,** [in] EGlobalWorkplaneType **GlobalWorkplaneType,** [in] double **PlaneOffset**)

Index *Index of the global work plane*
GlobalWorkplaneType *Global plane of the new work plane*
PlaneOffset *Distance of the work planes's origin from global plane*

Get parameters of the global work plane. If successful, returns the work plane index otherwise an error code (errDatabaseNotReady, errIndexOutOfBounds, wpeInvalidWorkPlaneParameters, wpeWorkplaneIsNotGlobal)

---

long **SetSmartParameters** ([in] long **Index,** [in] ESmartWorkplaneElementType **SmartWorkplaneElementType,** [in] long **ElementIndex**)

Index *Index of the global work plane*
SmartWorkplaneElementType *Type of the element used for the work plane*
ElementIndex *index of the element*

Set parameters of the smart work plane. If successful, returns the work plane index otherwise an error code (errDatabaseNotReady, errIndexOutOfBounds, wpeWorkplaneIsNotSmart, wpeInvalidWorkPlaneParameters)

Properties

long **Count** *Get number of work planes*

ELongBoolean **HideNotInPlane** • [long **Index**]
Index *index of the work plane*
Get or set whether elements which are not in the work plane are hidden or not

long **Name** [long **Index**]
Index *index of the work plane*
Get name of the work plane.

ELongBoolean **ShowGrayedNotInPlane•** [long **Index**]
Index *index of the work plane*
Get or set whether elements which are not in the work plane are shown greyed or not

EWorkplaneType **WorkplaneType** [long **Index**]
Index *index of the work plane*
Get type of the work plane.

IAxisVMXLAMpanels

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 thicknessess must 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** = {

ltStraightLine = 0x00,	*straight line*
ltCircleArc = 0x01 }	*arc*

Geometry.

Records / structures

RPoint2d = (

double	**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.*

Obtains startpoint or endpoint coordinates.

---

void **SetCircleArcCenter** ([i/o] RPoint2d **Value**)

Value *circle arc center coordinates in metres.*

Set circle arc center coordinates.

---

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.*

Set line endpoints

---

void **SetPoint** ([in] ELine2dPointIndex **Index,** [i/o] RPoint2d **Value**)

Index *specifies the startpoint or the endpoint*

EndPoint *point coordinates in metres.*

Set startpoint or endpoint coordinates.

Properties

EArcAngleOrientation **CircleArcOrientation** • *(if LineType = ltCircleArc) 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	**LineType**	*line geometry*
RPoint3d	**P1**	*startpoint*
RPoint3d	**P2**	*endpoint*
RPoint3d	**ArcCenter**	*(if LineType = ltCircleArc3d) center of the circle*
EArcAngleOrientation	**ArcOrientation**	*(if LineType = ltCircleArc3d) orientation of the arc. There are 2 arcs on a circle between points P1, P2. oCounterClockwise will select the one starting from P1 and going according to right-hand rule based on NormVect. oClockwise will select the other arc.*
RPoint3d	**NormVect**	*(if LineType = ltCircleArc3d) arc plane normal (the normal for a plane can have 2 orientation, ArcOrientation will interpreted based on its direction)*

)

RPoint3d = (

double	**x, y, z**	*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.

ELongBoolean **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 < Index ≤ 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 < Index ≤ Count)*

Set a line by index. If unsuccessful returns an error code (errIndexOutOfBounds)

---

long **StraightLines** (([in] SAFEARRAY(RPoint3d) **Coords**)

Coords *list of vertex coordinates*

Clears the existing lines and creates a straight linelist with the vertexes passed in Coords. If it is intended for a polygon, the first and last coordinate must be

the same (i.e. a closed line). Result is the number of vertexes passed in Coords.

Properties

long **Count**

Get *number of 3D lines in the list*

IAxisVMMovingLoadOnBeam

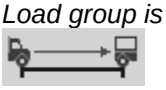
Moving loads on beams. This interface can be created with ObjectCreator then added with IAxisVMMovingLoads interface.

Enumerated types

enum **ERunningMode** = {

rmOneWay = 0x00,	*Moving one way*
rmRoundTrip = 0x01 }	*Moving around*

enum **EStructureMode** = {

smCrainrunway = 0x00,	*Load group is applied at start and end* 
smBridge = 0x01 }	*Load group is not applied at start and end* 

Records / structures

RConcentratedMovingLoadOnBeam = (

ESystem	**SystemGL**	*coordinate system of load components (global or local only)*
double	**Position**	*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*
double	**Fx, Fy, Fz**	*x, y, z force components [kN]*
double	**Mx, My, Mz**	*moment components about the x, y, z axis [kNm]*

)

RDistributedMovingLoadOnBeam = (

ESystem	**SystemGL**	*coordinate system of load components (global or local only)*
EBeamRibDistributionType	**DistributionType**	*distributed by length or projected*
double	**Position1**	*position of the start point where the distributed load is applied, , see AxisVM manual Moving loads for more info*
double	**Fx1, Fy1, Fz1**	*x, y, z force components at the start point of the distributed load [kN]*
double	**Position2**	*position of the end point where the distributed load is applied, , see AxisVM manual Moving loads for more info*
double	**Fx2, Fy2, Fz2**	*x, y, z force components at the end point of the distributed load [kN]*

)

RMovingLoadOnBeamItem = (

ELoadType	**ItemType**	*ltBeamConcentrated or ltBeamDistributed*
RConcentratedMovingLoadOnBeam	**Concentrated**	*Parameters of concentrated moving load on beam*
RDistributedMovingLoadOnBeam	**Distributed**	*Parameters of distributed moving load on beam*

) *Record is used for defining moving load on beam item*

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

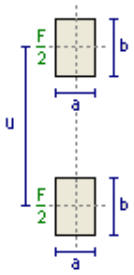
Moving loads on beams. This interface can be created with ObjectCreator then added with IAxisVMMovingLoads interface.

Records / structures

RMovingLoadOnDomainItem = (

ESystem	**SystemGL**	*coordinate system of load components*
double	**Position**	*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*
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 then 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 *Index of the moving load on a domain*

MovingLoadOnDomainItem *Moving load on a domain parameters*

Set moving load parameters of a load item with index ItemIndex. If successful, returns load item index, otherwise an error code (errIndexOutOfBounds, errDatabaseNotReady, mleInvalidSystemValue).

---

long **SetPath** ([in] SAFEARRAY (RPoint3D) *** Path,** [in] SAFEARRAY (RPoint3D) *** NormV**)

Path *Global coordinates of the path defined by points, (has always one more items then NormV array)*

NormV *Normal vectors of each path segment, path segment is the line between two points of the path*

Set path of the moving load acting at two points. If successful, returns 1, otherwise an error code (errDatabaseNotReady, mleInvalidPathOrNormV, mleInvalidNormVLength).

---

long **SetPath_vb** (Visual Basic compatible function of **SetPath**)

Properties

ELongBoolean	**ConcentratedLoad** *Get or set number of moving load is acting as point load at centre of wheel (No by default)*
long	**ItemCount** *Get number of moving load items*
long	**LoadCaseId** *Get or set loadcase index of moving load on domain*
ERunningMode	**RunningMode** *Get or set mode of run of moving load on domain*
long	**Steps** *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*
EStructureMode	**StructureMode** *Get or set mode of structure of moving load on domain*

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 < Index ≤ 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 < Index ≤ 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.

Most events will merely contain an Error function, called when an error has happened in the related entity. For example IAxisVMLines.Delete(133) is called in a model with only 100 lines. In this case IAxisVMLinesEvents.Error will be called with an Index of 133, and the errorcode errIndexOutOfBounds. In the case of IAxisVMLines.Delete the result will also be errIndexOutOfBounds. However, there are many cases when the return value is used for a different purpose. For example IAxisVMLoads.AddLineSelfWeights returns the number of successfully created loads. If AddLineSelfWeights receives an array of records where one record contains a LineId of 133, it will not create a self weight for that record (because there are only 100 lines in the model). It will trigger however the IAxisVMLinesEvents.Error with the Index = 133 and ErrorCode = errIndexOutOfBounds. Because of this specific record the return value will be one less.than the length of the input array.

The list of AxisVM COM event interfaces:

IAxisVMAccelerationEvents

Events:

void **Error** ([in] long **Index**, [in] long **ErrorCode**)

Index *Not used*

ErrorCode *the error code*

Called in case of an error.

IAxisVMActualReinforcementEvents

Events:

void **Error** ([in] long **Index**, [in] long **ErrorCode**)

Index *Not used*

ErrorCode *the error code*

Called in case of an error.

IAxisVMApplicationEvents

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

Events:

void **Error** ([in] long **Index**, [in] long **ErrorCode**)

Index *Not used*

ErrorCode *the error code*

Called in case of an error.

IAxisVMAttributesEvents

Events:

void **Error** ([in] long **Index**, [in] long **ErrorCode**)

Index *Not used*

ErrorCode *the error code*

Called in case of an error.

IAxisVMColumnRebarsEvents

Events:

void **Error** ([in] long **Index**, [in] long **ErrorCode**)

Index *Not used*

ErrorCode *the error code*

Called in case of an error.

IAxisVMCalculatedReinforcementEvents

Events:

void **Error** ([in] long **Index**, [in] long **ErrorCode**)

Index *Not used*

ErrorCode *the error code*

Called in case of an error.

IAxisVMCalculationEvents

Enumerated types:

enum **ECalculatioFinishedType**= {

cft_OK= 0	*calculation finished without problems*
cft_Canceled= 1	*calculation finished due to cancellation*
cft_Warning= 2	*calculation finished with warning(s)*
cft_Error= 3}	*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 occured (string array)*

LoadCombinationIds *load combinationor load case indexes where error/ warning occured (long array)*

Called after the calculation (analysis) has finished.

IAxisVMCrackWidthEvents

Events:

void **Error** ([in] long **Index**, [in] long **ErrorCode**)

Index *Not used*

ErrorCode *the error code*

Called in case of an error.

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

Events:

void **Error** ([in] long **Index**, [in] long **ErrorCode**)

Index *Not used*

ErrorCode *the error code*

Called in case of an error.

IAxisVMCustomPartFolderEvents

Events:

void **Error** ([in] long **Index**, [in] long **ErrorCode**)

Index *Not used*

ErrorCode *the error code*

Called in case of an error.

IAxisVMDisplacementsEvents

Events:

void **Error** ([in] long **Index**, [in] long **ErrorCode**)

Index *Not used*

ErrorCode *the error code*

Called in case of an error.

IAxisVMDiaphragmEvents

Events:

void **Error** ([in] long **Index**, [in] long **ErrorCode**)

Index *Not used*

ErrorCode *the error code*

Called in case of an error.

IAxisVMDimensionsEvents

Events:

void **Error** ([in] long **Index**, [in] long **ErrorCode**)

Index *Not used*

ErrorCode *the error code*

Called in case of an error.

IAxisVMDomainsEvents

Events:

void **Error** ([in] long **Index**, [in] long **ErrorCode**)

Index *Not used*

ErrorCode *the error code*

Called in case of an error.

IAxisVMDomainSupportsEvents

Events:

void **Error** ([in] long **Index**, [in] long **ErrorCode**)

Index *Not used*

ErrorCode *the error code*

Called in case of an error.

IAxisVMDrawingsLibraryEvents

Events:

void **Error** ([in] long **Index**, [in] long **ErrorCode**)

Index *Not used*

ErrorCode *the error code*

Called in case of an error.

IAxisVMDynamicLoadFunctionsEvents

Events:

void **Error** ([in] long **Index**, [in] long **ErrorCode**)

Index *Not used*

ErrorCode *the error code*

Called in case of an error.

IAxisVMEdgeConnectionsEvents

Events:

void **Error** ([in] long **Index**, [in] long **ErrorCode**)

Index *Not used*

ErrorCode *the error code*

Called in case of an error.

IAxisVMEnvelopesEvents

Events:

void **Error** ([in] long **Index**, [in] long **ErrorCode**)

Index *Not used*

ErrorCode *the error code*

Called in case of an error.

IAxisVMForcesEvents

Events:

void **Error** ([in] long **Index**, [in] long **ErrorCode**)

Index *Not used*

ErrorCode *the error code*

Called in case of an error.

IAxisVMIncrementFunctionsEvents

Events:

void **Error** ([in] long **Index**, [in] long **ErrorCode**)

Index *Not used*

ErrorCode *the error code*

Called in case of an error.

IAxisVMLayersEvents

Events:

void **Error** ([in] long **Index**, [in] long **ErrorCode**)

Index *Not used*

ErrorCode *the error code*

Called in case of an error.

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

Events:

void **Error** ([in] long **Index**, [in] long **ErrorCode**)

Index *Not used*

ErrorCode *the error code*

Called in case of an error.

IAxisVMLinkElementsEvents

Events:

void **Error** ([in] long **Index**, [in] long **ErrorCode**)

Index *Not used*

ErrorCode *the error code*

Called in case of an error.

IAxisVMLoadsEvents

Events:

void **Error** ([in] long **Index**, [in] long **ErrorCode**)

Index *Not used*

ErrorCode *the error code*

Called in case of an error.

IAxisVMLoadCasesEvents

Events:

void **Error** ([in] long **Index**, [in] long **ErrorCode**)

Index *Not used*

ErrorCode *the error code*

Called in case of an error.

IAxisVMLoadCombinationsEvents

Events:

void **Error** ([in] long **Index**, [in] long **ErrorCode**)

Index *Not used*

ErrorCode *the error code*

Called in case of an error.

IAxisVMLoadGroupsEvents

Events:

void **Error** ([in] long **Index**, [in] long **ErrorCode**)

Index *Not used*

ErrorCode *the error code*

Called in case of an error.

IAxisVMLogicalPartsEvents

Events:

void **Error** ([in] long **Index**, [in] long **ErrorCode**)

Index *Not used*

ErrorCode *the error code*

Called in case of an error.

IAxisVMMaterialsEvents

Events:

void **Error** ([in] long **Index**, [in] long **ErrorCode**)

Index *Not used*

ErrorCode *the error code*

Called in case of an error.

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.

IAxisVMMembersSupportsEvents

Events:

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**

All elements of the list have been cleared.

---

void **Cleared**

Index	*index of the list element to delete*

All elements of the list have been cleared.

---

void **DisplayedErrors** ([in] BSTR **Title,** in] BSTR **Description,** in] BSTR **Errors**)

Title	*Title of the error*
Description	*Description of the error*
Errors	*Error messages(s), contain a multiline strings (separated with CR+LF)*

Called before showing a form with errors

---

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 progress as a number between 0 and 1*
Abort	*if set to lbTrue, the progres is aborted (does not work all the time)*

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.

---

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,** [in] ELongBoolean **NewStatus**)

Index *index of the model (always set to 1)*

NewStatus *True if new model*

Called when model is loaded

---

void **ModelChanged** ([in] long **Index**)

Index *index of the model (always set to 1)*

Called after model has been changed, not called when only the view of the model has changed

The FileChanged event is not invoked when this event is invoked.

---

void **FileChanged** ([in] long **Index**)

Index *index of the model (always set to 1)*

Called after the file has changed, also when view has changed. Invoked after all operation affecting the model fiile, e.g.: If new cross-section or material is created, but not associated to any element, the file is changed but not the model.

The ModelChanged event is not invoked when this event is invoked.

---

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.

---

void **SelectionProcessingChanged** ([in] long **Index,** [in] ELongBoolean **NewStatus**)

Index *index of the model (always set to 1)*

NewStatus *if True, the selection task has started*
if False, the selection task has finished

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).

IAxisVMMovingLoadsEvents

Events:

void **Error** ([in] long **Index**, [in] long **ErrorCode**)

Index *Not used*

ErrorCode *the error code*

Called in case of an error.

IAxisVMMovingLoadOnBeamEvents

Events:

void **Error** ([in] long **Index**, [in] long **ErrorCode**)

Index *Not used*

ErrorCode *the error code*

Called in case of an error.

IAxisVMMovingLoadOnDomainEvents

Events:

void **Error** ([in] long **Index**, [in] long **ErrorCode**)

Index *Not used*

ErrorCode *the error code*

Called in case of an error.

IAxisVMNodalSupportsEvents

Events:

void **Error** ([in] long **Index**, [in] long **ErrorCode**)

Index *Not used*

ErrorCode *the error code*

Called in case of an error.

IAxisVMNodesEvents

Events:

void **Error** ([in] long **Index**, [in] long **ErrorCode**)

Index *Not used*

ErrorCode *the error code*

Called in case of an error.

IAxisVMPushoverHingeFunctionsEvents

Events:

void **Error** ([in] long **Index**, [in] long **ErrorCode**)

Index *Not used*

ErrorCode *the error code*

Called in case of an error.

IAxisVMRCBeamDesignEvents

Events:

void **Error** ([in] long **Index**, [in] long **ErrorCode**)

Index *Not used*

ErrorCode *the error code*

Called in case of an error.

IAxisVMRCColumnCheckingEvents

Events:

void **Error** ([in] long **Index**, [in] long **ErrorCode**)

Index *Not used*

ErrorCode *the error code*

Called in case of an error.

IAxisVMRigidBodiesEvents

Events:

void **Error** ([in] long **Index**, [in] long **ErrorCode**)

Index *Not used*

ErrorCode *the error code*

Called in case of an error.

IAxisVMRebarSteelGradesEvents

Events:

void **Error** ([in] long **Index**, [in] long **ErrorCode**)

Index *Not used*

ErrorCode *the error code*

Called in case of an error.

IAxisVMReferencesEvents

Events:

void **Error** ([in] long **Index**, [in] long **ErrorCode**)

Index *Not used*

ErrorCode *the error code*

Called in case of an error.

IAxisVMReportsEvents

Events:

void **Error** ([in] long **Index**, [in] long **ErrorCode**)

Index *Not used*

ErrorCode *the error code*

Called in case of an error.

IAxisVMReinforcementCheckEvents

Events:

void **Error** ([in] long **Index**, [in] long **ErrorCode**)

Index *Not used*

ErrorCode *the error code*

Called in case of an error.

IAxisVMResultsEvents

Events:

void **Error** ([in] long **Index**, [in] long **ErrorCode**)

Index *Not used*

ErrorCode *the error code*

Called in case of an error.

IAxisVMSectionsEvents

Events:

void **Error** ([in] long **Index**, [in] long **ErrorCode**)

Index *Not used*

ErrorCode *the error code*

Called in case of an error.

IAxisVMSeismicStoreysEvents

Events:

void **Error** ([in] long **Index**, [in] long **ErrorCode**)

Index *Not used*

ErrorCode *the error code*

Called in case of an error.

IAxisVMSettingsEvents

Events:

void **Error** ([in] long **ErrorCode**)

ErrorCode *the error code*

Called in case of an erroneous setting.

IAxisVMShearCapacityEvents

Events:

void **Error** ([in] long **Index**, [in] long **ErrorCode**)

Index *Not used*

ErrorCode *the error code*

Called in case of an error.

IAxisVMSpectrumEvents

Events:

void **Error** ([in] long **Index**, [in] long **ErrorCode**)

Index *Not used*

ErrorCode *the error code*

Called in case of an error.

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

Events:

void **Error** ([in] long **Index**, [in] long **ErrorCode**)

Index *Not used*

ErrorCode *the error code*

Called in case of an error.

IAxisVMSteelDesignResultsEvents

Events:

void **Error** ([in] long **Index**, [in] long **ErrorCode**)

Index *Not used*

ErrorCode *the error code*

Called in case of an error.

IAxisVMStoreysEvents

Events:

void **Error** ([in] long **Index**, [in] long **ErrorCode**)

Index *Not used*

ErrorCode *the error code*

Called in case of an error.

IAxisVMStressesEvents

Events:

void **Error** ([in] long **Index**, [in] long **ErrorCode**)

Index *Not used*

ErrorCode *the error code*

Called in case of an error.

IAxisVMStructuralGridsEvents

Events:

void **Error** ([in] long **Index**, [in] long **ErrorCode**)

Index *Not used*

ErrorCode *the error code*

Called in case of an error.

IAxisVMSurfacesEvents

Events:

void **Error** ([in] long **Index**, [in] long **ErrorCode**)

Index *Not used*

ErrorCode *the error code*

Called in case of an error.

IAxisVMSurfaceSupportsEvents

Events:

void **Error** ([in] long **Index**, [in] long **ErrorCode**)

Index *Not used*

ErrorCode *the error code*

Called in case of an error.

IAxisVMTimberDesignMembersEvents

Events:

void **Error** ([in] long **Index**, [in] long **ErrorCode**)

Index *Not used*

ErrorCode *the error code*

Called in case of an error.

IAxisVMTimberDesignResultsEvents

Events:

void **Error** ([in] long **Index**, [in] long **ErrorCode**)

Index *Not used*

ErrorCode *the error code*

Called in case of an error.

IAxisVMTimeIncrementFunctionsEvents

Events:

void **Error** ([in] long **Index**, [in] long **ErrorCode**)

Index *Not used*

ErrorCode *the error code*

Called in case of an error.

IAxisVMVelocityEvents

Events:

void **Error** ([in] long **Index**, [in] long **ErrorCode**)

Index *Not used*

ErrorCode *the error code*

Called in case of an error.

IAxisVMVirtualBeamsEvents

Events:

void **Error** ([in] long **Index**, [in] long **ErrorCode**)

Index *Not used*

ErrorCode *the error code*

Called in case of an error.

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.

IAxisVMXLAMpanelsEvents

Events:

void **Error** ([in] long **Index**, [in] long **ErrorCode**)

Index *Not used*

ErrorCode *the error code*

Called in case of an error.

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 functions 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.

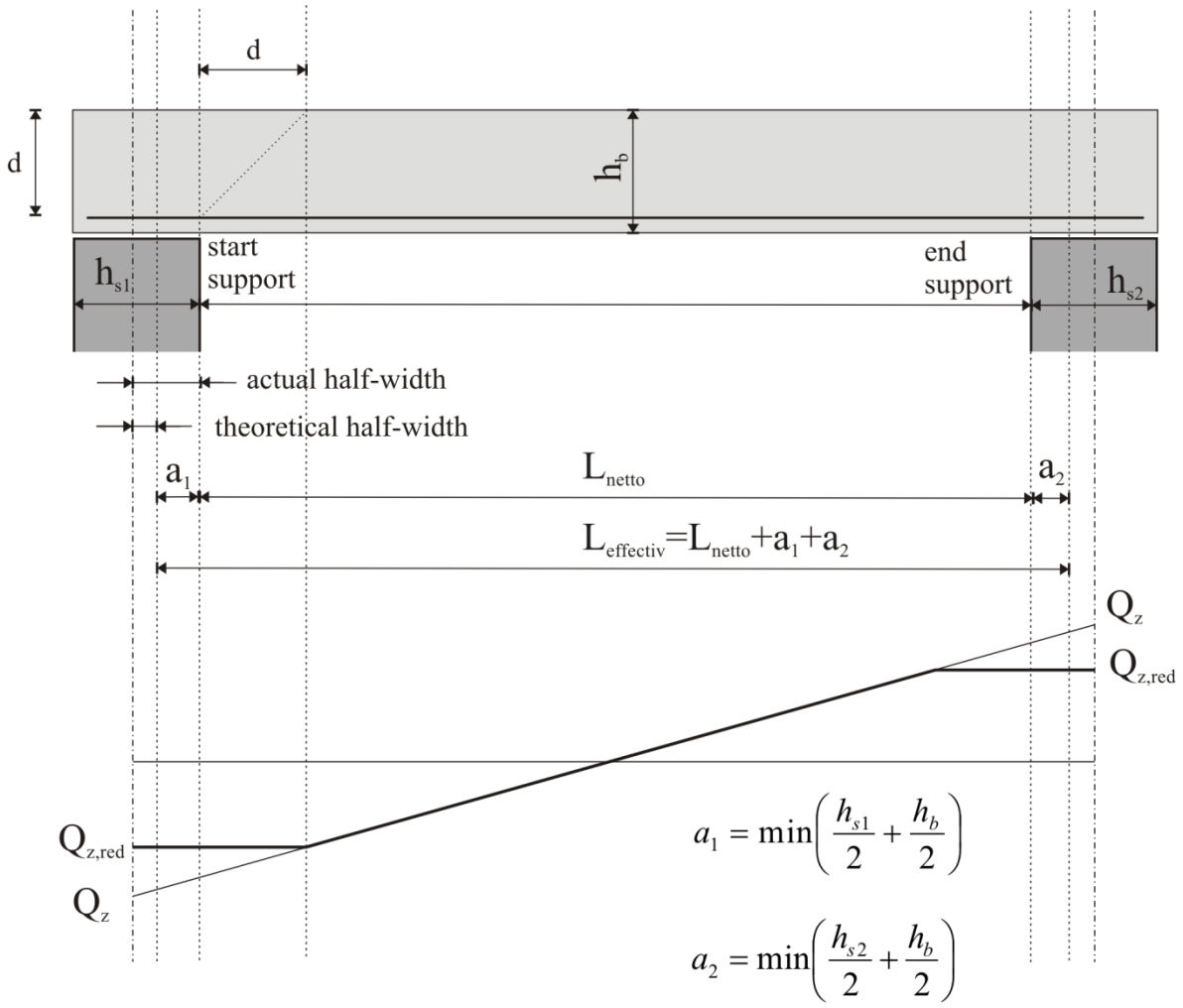
The following interface and function names are handled in the latest release of AxisVM x4:

InterfaceName	FunctionName
IAxisVMApplication	**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 asociated loads, like loads applied on load panels etc.*

Shear force reduction

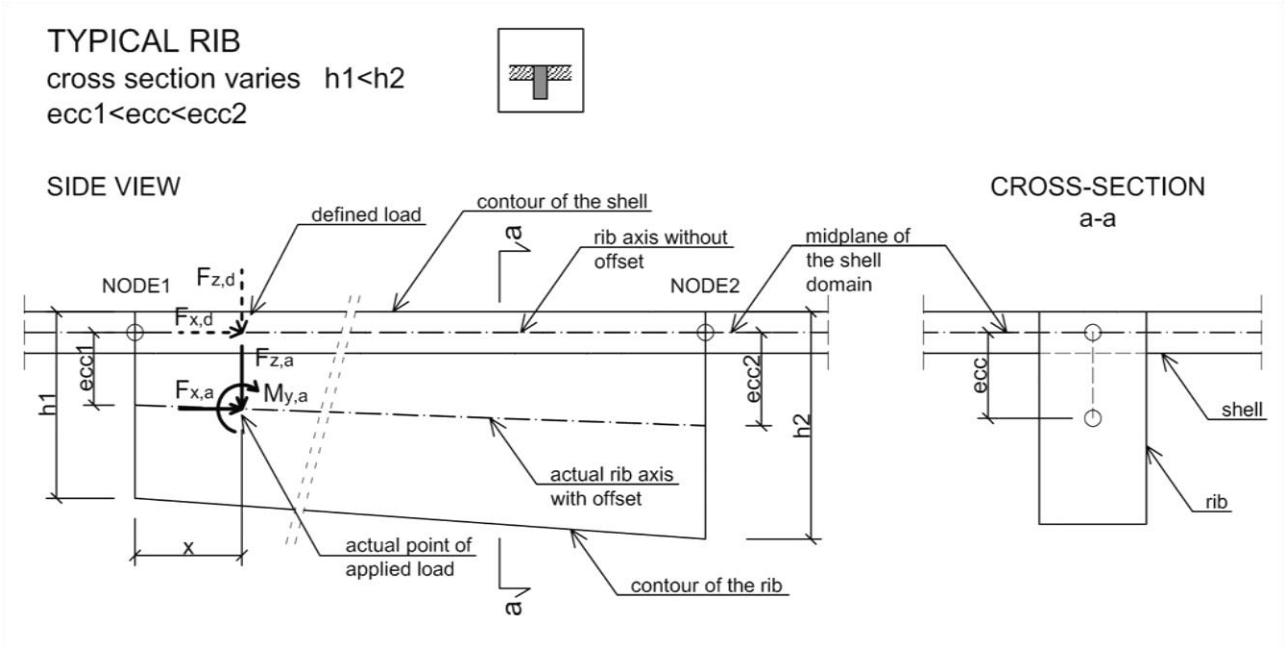
The Qz,red is the shear force Qz at distance d from face of the support. If the load is applied at the top of the beam, then part of the load between axis of the support and d distance from the face of the support is directly distributed to the support.



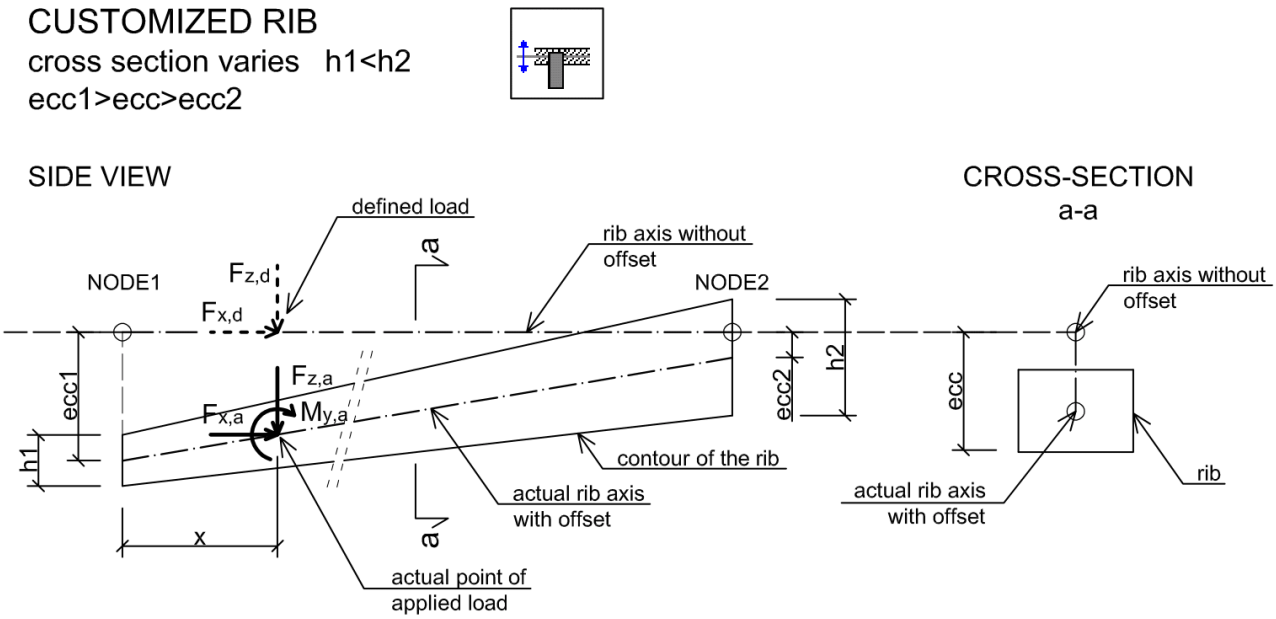
Eccentricity and loads on ribs

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 *Fx,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 a 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 a 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).



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 IAxisVMApplication interface.

[5] Just like in previous versions COM clients must start AxisVM COM server by creating a COM object with the IAxisVMApplication 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 IAxisVMApplication
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
- IAxisVMIncrementFunctions
- IAxisVMVelocity
- IAxisVMAcceleration
- IAxisVMRigidBodies
- IAxisVMDiaphragm
- IAxisVMObjectCreator

New functions:

- IAxisVMApplication
  - 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:

- IAxisVMModel
  - 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
- IAxisVMApplication:
  - 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 conrete 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 conrete cover or rebar position depending on used design code!!!!*

Property renamed:
IAxisVMMember
LineType -> MemberType

Enum value renamed:
lngGreek -> lngBrasilianPortuguese

New functions:

- IAxisVMMember
  - DefineAsTruss
  - DefineAsTimberTruss
  - GetTrussData
  - GetTimberTrussData

- IAxisVMMembers
  - DeleteNameOfAllTrussses
  - 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

- AxisVMApplication
  - 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)
  - fi -> ds
  - d -> 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

- **IAxisVMModel**:
    - 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

- **IAxisVMModel**
    - Sections

Changes between versions 6.2 and 6.3

New types in enums:

- ENationalDesignCode
    - ndcEuroCode_PL
    - ndcEuroCode_DK
    - ndcEuroCode_S

- Elanguage
    - lngPolish
    - lngBulgarian

- EReleaseType
    - rtPushover

New fields in records

- RRCBeamDesignParameters
    - ShortTime

- RReinforcementParameters
    - ConcreteTensilStrengthCheck

- 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:

- IAxisVMModel
  - 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, IAxisVMEnvelopes, IAxisVMWindows, IAxisVMWindow, IAxisVMCustomParts, IAxisVMCustomPartFolder, IAxisVMLogicalParts

New properties

- IAxisVMStresses , IAxisVMForces, IAxisVMDisplacements, IAxisVMSteelDesignResults, IAxisVMTimberDesignResults, IAxisVMCalculatedReinforcement, IAxisVMReinforcementCheck, IAxisVMShearCapacity, IAxisVMCrackOpening, IAxisVMVelocity, IAxisVMAcceleration, IAxisVMSections
  - EnvelopeUID

- IAxisVMModel
  - 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 IAxisVMLogicalParts return partUID.

New functions:

- IAxisVMWindow
  - GetWindowDisplayPartUIDs
  - GetPixelPosition
  - SetWindowDisplayPartUIDs
  - SaveWindowToBitmap
  - SaveWindowToClipboard
  - SaveWindowToMetafile

- IAxisVMWindows
  - GetWindowDisplayPartUIDs
  - SetWindowDisplayPartUIDs
  - SaveWindowToBitmap
  - SaveWindowsToBitmap
  - SaveWindowToClipboard
  - SaveWindowsToClipboard
  - SaveWindowToMetafile
  - SaveWindowsToMetafile

- IAxisVMLogicalParts
  - GetEnabledLogicalParts
  - SetEnabledLogicalParts
  - SaveDefaultEnabledLogicalParts

- IAxisVMLoads
  - GetDomainPolyLineItems

New properties:

- IAxisVMLogicalParts
  - IsLogicalPart

- IAxisVMCustomParts
  - IsCustomPart

- IAxisVMApplication
  - 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
- Ialfa -> Ialpha

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
- IAxisVMAttributes
- IAxisVMWorkplanes

New event:

- IAxisVMCalculationEvents
  - Error([in] long Index, [in] long ErrorCode)

New properties:

- IAxisVMLines
  - UID
  - IndexOfUID
- IAxisVMDomain
  - ContourPolygon
- IAxisVMNodes, IAxisVMLines, IAxisVMMembers, IAxisVMSurfacess, IAxisVMDomains
  - Attributes
- IAxisVMWindow
  - View
  - WorkPlaneIndex
  - StoryIndex
  - ActiveStoryIndex
- IAxisVMWindows
  - LoadCaseIndex
  - View

- WorkPlaneIndex
- StoryIndex
- ActiveStoryIndex

- IAxisVMModel
  - 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 IAxisVMmodel for compatibility)*
  - SetSeismicParams (*relocated but also retained in IAxisVMmodel for compatibility*)

- IAxisVMApplication
  - 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

- IAxisVMModel
    - SetAPIGlobalData
    - GetAPIGlobalData
    - GetAPIGlobalDataSize
    - DeleteAPIGlobalData
    - EnableMainForm
    - DisableMainForm

New events:

- IAxisVMModelsEvents
    - ModelChanged

Changes between versions 7.2 and 7.3

New functions:

- IAxisVMAttributes
  - 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

- IAxisVMEnvelopes
  - Update

- IAxisVMModel
  - ImportDXF
  - ImportPDF
  - ImportIFC
  - GetIFCExportReinfParams

- IAxisVMResults
  - GetCapacityCurvePushOver

New interfaces

- IAxisVMAttachments, IAxisVMSteelCrossSectionOptimization

New properties:

- IAxisVMNodes, IAxisVMLines, IAxisVMMembers, IAxisVMSurfacess, IAxisVMDomains
  - Attachments

- IAxisVMLines, IAxisVMMembers
  - LocalX_is_ij
  - Name

- IAxisVMEnvelopes
  - Group

- IAxisVMModel
  - CallImportProgress
  - Platform
  - SpectrumPushOver

Function parameters changed:

- IAxisVMSurface, IAxisVMDomain
  - GetReinforcementParameters
  - SetReinforcementParameters

- IAxisVMRCbeamDesign
  - SetDesignParameters
  - SetDesignParameters_vb
  - GetDesignParameters

New events:

- IAxisVMModelsEvents
  - MainProgress

Changes between versions 7.3 and 7.4

New events:

- IAxisVMModelsEvents
  - BeforeMessageDisplay
  - AfterMessageDisplay
  - DisplayedErrors

New properties:

- IAxisVMWindow
  - ShowOnlySelected

Changes between versions 7.4 and 7.5

Renamed properties:

- IAxisVMModel
  - 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
  - GetXLAMParameters
  - SetXLAMParameters
- 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

        - lpeStructuralGridLineUIDOutOfBounds

- IAxisVMStructuralGrids
    - EStructuralGidsError
- IAxisVMDimensions
    - EDimensionsError

Deleted error codes:

- IAxisVMCalculation
    - ECalculationError
        - ceCalculationWasCanceled
        - ceCalculationEndsWithError

New enums:

- IAxisVMSteelDesignMembers
    - ESteelBucklingLengthMode
    - ESteelCantileverFixedEnd
    - ESteelLateralSupports
- IAxisVMModel
    - ECompanyLogoPosition
    - ECompanyLogoSizeOption
    - EGeneralAlignment
- IAxisVMCrossSections
    - ECrossSectionImageExportOptions
- IAxisVMStructuralGrid
    - EGridLineSpacingDirection
- IAxisVMStructuralGrids
    - EShowStructuralGridLineTitle
    - EStructuralGridPlane
    - EStructuralGridVisibility
    - EStructuralGridLabelType
- IAxisVMStresses
    - EXLAMSurfaceEfficiency
- IAxisVMDimensions
    - EDimensionType
    - EDimensionLabelOrientation
    - EDimensionStyle
- IAxisVMSections
    - ESectionSegmentChainIntegratedResultant
- IAxisVMCalculationEvents
    - ECalculatioFinishedType

Modified enums:

- IAxisVMDomain
    - EEnvironmentClass
- IAxisVMSteelDesignMembers
    - EMcrMethod
        - mcrmUser
- IAxisVMStresses
    - EXLAMSurfaceStress
        - xssSrx_max
        - xssSry_max

New records:

- IAxisVMModel
    - RCompanyLogoParameters
- IAxisVMStructuralGrids
    - RStructuralGridLineParams
    - RStructuralGridParams
    - RStructuralGridGenerationParams
- IAxisVMStresses

    - 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
- IAxisVMModel
  - GetCompanyLogoParameters
  - SetCompanyLogoParameters
  - SelectAll

Modified functions:

- IAxisVMStresses
  - GetEnvelopeXLAMSurfaceStress
  - GetEnvelopeXLAMSurfaceStresses
- IAxisVMApplication
  - Platform → AxisVMPlatform

New properties:

- IAxisVMLineSupports
  - LineID

- IAxisVMCustomParts
  - IndexOfUID
  - FullName

- IAxisVMLogicalParts
  - Name
  - FullName

- IAxisVMModel
  - 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
- IAxisVMModel
  - EModelError
    - meRevitModuleNotAvailable
    - meRevitImportTessDegreeOutOfRange
- IAxisVMMembers
  - EMembersError
    - mbeInvalidFunctionIDofRelease
    - mbeReleaseInitAndLimitMustBe0

        - mbeFunctionIdMustBe0
        - mbeMembersNotContinuous

- IAxisVMVirtualBeams
    - EVirtualBeamError
        - vbeDomainIndexOutOfBounds
        - vbeDomainIndexIsInvalid
        - vbeDomainListIsEmpty
        - vbeChainIndexIsInvalid

New enums:

- IAxisVMForces
    - EVirtualBeamForce
    - EConnectedToNodeType
- IAxisVMVirtualBeams
    - RVirtualBeamParams
    - RVirtualStripParams
- IAxisVMModel
    - EGeneralAlignmentHorizontal
    - EGeneralAlignmentVertical
- IAxisVMResults
    - EXYchartFillType
    - EXYchartLabelingStyle

Modified enums:

- IAxisVMModel
    - EGeneralAlignment -> EGeneralAlignmentHorizontal

New records:

- IAxisVMForces
    - RVirtualBeamForceValues
- IAxisVMVirtualBeams
    - EVBDomainsDuplicateMode
    - EVirtualBeamType

Modified records:

- IAxisVMModel
    - RCompanyLogoParameters

New functions:

- IAxisVMForces
    - LineForcesByLoadCaseIdConnectedToNode
    - LineForcesByLoadCombinationIdConnectedToNode
    - EnvelopeLineForcesConnectedToNode
    - CrtiticalLineForcesConnectedToNode
    - VirtualBeamOrStripForcesByLoadCaseId
    - VirtualBeamOrStripForcesByLoadCombinationId
    - EnvelopeVirtualBeamOrStripForces
    - EnvelopeVirtualBeamOrStripForces2
    - CriticalVirtualBeamOrStripForce
    - CriticalVirtualBeamOrStripForce2
    - CriticalVirtualBeamOrStripForces
- IAxisVMModel
    - 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:

- IAxisVMModel
  - 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
    - lopeNodeIndexListEmpty
    - lopeDomainIndexListEmpty
- 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
    - vbeInvalidSections
- IAxisVMCrossSections
  - ECrossSectionError
    - cseTooHigh_e

New enums:

- IAxisVMForces
  - ESeismicComponentSumType
- IAxisVMVirtualBeams
  - EVBDefinitionType
  - EVSDefinitionType
- IAxisVMWindows
  - EDisplayMode
    - dmDiagramFilled
    - dmSectionFilled
- IAxisVMLoadCases
  - ELoadCaseType

        - lctfire
- IAxisVMLoadGroups
    - ELoadGroupType
        - lgtfire

New functions:

- IAxisVMForces
    - LineForceByLoadCombinationIdEQ
    - EnvelopeLineForceEQ
    - CrtiticalLineForceEQ
    - SaveVirtualBeamOrStripForcesToMetaFileByLoadCaseID
    - SaveVirtualBeamOrStripForcesToMetaFileByLoadCombinationID
    - SaveEnvelopeVirtualBeamOrStripForcesToMetaFile
    - SaveCriticalVirtualBeamOrStripForcesToMetaFile
- IAxisVMLoads
    - ConvertDerivedSurfaceLoad
    - ConvertSelectedDerivedSurfaceLoad
- IAxisVMWindow
    - ReDraw
- IAxisVMWindows
    - ReDraw
- IAxisVMApplication
    - Quit
- IAxisVMVirtualBeams
    - GetReductionPointCoordinates
    - GetExtendedDomainList

New events:

- IAxisVMLinesEvents
    - AfterDeleted
- IAxisVMMembersEvents
    - AfterDeleted
- IAxisVMModelsEvents
    - FileChanged

New properties:

- IAxisVMForces
    - SeismicComponentSumType

Modified functions:

- IAxisVMForces
    - LineForcesByLoadCombinationIdConnectedToNode
    - EnvelopeLineForcesConnectedToNode
    - CrtiticalLineForcesConnectedToNode
- IAxisVMRCColumnChecking
    - CheckByParameters
- IAxisVMVirtualBeams
    - AddNewVirtualBeam

Modified records:

- IAxisVMCollumnRebars
    - 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:

- IAxisVMTask

New properties:

- IAxisVMSettings
  - StiffnessReductionColumns_A
  - StiffnessReductionColumns_I
  - StiffnessReductionBeams_A
  - StiffnessReductionBeams_I
  - StiffnessReductionOtherMembers_A
  - StiffnessReductionOtherMembers_I
  - StiffnessReductionWalls
  - StiffnessReductionSlabs
  - StiffnessReductionOtherDomains
- IAxisVMStresses, 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
- ERCColumnCheckingError
  - 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
  - CrossSetionID

- 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
- EXLAMSurfaceEfficiency
- 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
- IAxisVMApplication
    - HandleMessages

New properties:

- IAxisVMCrossSection
    - CrossSectionShapeEx
- IAxisVMCrossSectionOptimization
    - GroupCrossSectionShapeEx
- IAxisVMMembersSupports
    - ReferenceID

Changes between versions 15.0 and 15.1

New Errors:

- ELineError
    - lneInvalidSteelMaterialId
- 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
- ESteelFireParBetaMethod
- ESteelSLSHMethod
- ESteelSLSEMethod
- ESteelSLSLMethod
- ESteelSLSPreCamberCurve
- ETimberSLSEMethod
- ETimberSLSLMethod
- ETimberSLSPreCamberCurve
- ETimberSLSDesignCreepMode

Modified enum:
- EResultComponent
- ELineError
- EMembersError
- ESurfacesError
- EDomainsError
- ESeismicComponentSumType
- ENodesSupportsError
- ELineSupportsError
- EMembersSupportsError

New records:
- RRectangularFootingSpec
- RRectanularFootingCalced
- RCircularFootingSpec
- RCircularFootingCalced
- RPadFootingParams_V153
- RLinearFootingSpec
- RLinearFootingCalced
- RLinearFootingParams
- RBasicDisplayParameters_V153
- RExtendedDisplayParameters_V153
- RSpringParam
- RSpringParamIndexes
- RNodalSupportSpringParams
- RSeismicParams_V153
- RSpectrumData_ECHU
- RSpectrumData_ECNL
- RSpectrumData_V153
- RSteelDesignParameters_V153
- RSteelDesignParameters_EC_SIA_ITA_V153
- RTimberDesignParameters_V153
- RTimberDesignParameters_EC_V153

Obsolete records:
- RPadFootingDimensions
- RPadFootingParams
- RBasicDisplayParameters
- RExtendedDisplayParameters
- RSeismicParams
- RSpectrumData
- RSteelDesignParameters
- RSteelDesignParameters_EC_SIA_ITA
- RTimberDesignParameters
- RTimberDesignParameters_EC_SIA_ITA

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
- IAxisVMForces
    - NodalSupportForceByLoadCombinationIdEQ
    - EnvelopeNodalSupportForceEQ
    - CriticalNodalSupportForceEQ
    - LineSupportForceByLoadCombinationIdEQ
    - EnvelopeLineSupportForceEQ
    - CriticalLineSupportForceEQ
    - SurfaceForceByLoadCombinationIdEQ
    - EnvelopeSurfaceForceEQ
    - CriticalSurfaceForceEQ
- IAxisVMSteelDesignMembers
    - Add_V153
    - GetDesignParameters_V153
    - SetDesignParameters_V153
- IAxisVMTimberDesignMembers
    - Add_V153
    - GetDesignParameters_V153
    - SetDesignParameters_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
- IAxisVMSteelDesignMembers
    - Add
    - Add2
    - Add3
    - GetDesignParameters
    - GetDesignParameters2
    - GetDesignParameters3
    - SetDesignParameters
    - SetDesignParameters2
    - SetDesignParameters3
- IAxisVMTimberDesignMembers
    - Add
    - GetDesignParameters
    - SetDesignParameters

Obsolete properties:

- IAxisVMLines
    - StiffnessReduction_A
    - StiffnessReduction_I
- IAxisVMLine
    - StiffnessReduction_A
    - StiffnessReduction_I
- IAxisVMMembers
    - StiffnessReduction_A
    - StiffnessReduction_I
- IAxisVMMember
    - StiffnessReduction_A
    - StiffnessReduction_I
- IAxisVMSurfaces
    - StiffnessReduction
- IAxisVMSurface
    - StiffnessReduction
- IAxisVMDomains
    - StiffnessReduction
- IAxisVMDomain
    - StiffnessReduction

New properties:

- IAxisVMLines
    - StiffnessReduction
- IAxisVMLine
    - StiffnessReduction
- IAxisVMMembers
    - StiffnessReduction
- IAxisVMMember
    - StiffnessReduction

- IAxisVMNodalSupport
  - SpringParams
- IAxisVMNodesSupports
  - SpringParams
- IAxisVMSpectrum
  - Disabled
- IAxisVMLoadCases
  - SeismicGroupID
- IAxisVMSurfaces
  - StiffnessReduction_V153
- IAxisVMSurface
  - StiffnessReduction_V153
- IAxisVMDomains
  - StiffnessReduction_V153
- IAxisVMDomain
  - StiffnessReduction_V153
- IAxisVMForces
  - SupportSeismicSumType

Changes between versions 15.3 and 15.4

New enum:

- ECrossSectionRegion

Modified enum:

- ECrossSectionShapeEx

New records:

- RCrossSectionNameRec
- RCrossSectionTableRec
- RLoadNodalForce
- RLoadLineDistributed
- RLoadLineSelfWeigth
- RLoadSurfaceDistributed
- RLoadSurfaceSelfWeigth
- RLoadDomainLinear
- RLoadDomainConstant_V154
- RLoadDomainSelfWeight
- RBulkMemberSupport
- RBulkLineSupport
- RDoubleWedgedI

Obsolete records:

- RLoadDomainConstant

New interface:

- IAxisVMCrossSectionTables
- IAxisVMCrossSectionTable

New functions:

- IAxisVMNodes
  - BulkDelete
  - BulkSelect
- IAxisVMCatalog
  - GetTableCrossSections
  - GetAllTables
  - GetCrossSection_V154
- IAxisVMLoads
  - AddDomainConstants

    - AddDomainConstant_V154
    - AddDomainLinears
    - AddDomainSelfWeights
    - AddLineDistributeds
    - AddLineSelfWeights
    - AddNodalForces
    - AddSurfaceDistributeds
    - AddSurfaceSelfWeights
- IAxisVMMemberSupports
    - BulkAdd
- IAxisVMLineSupports
    - BulkAdd
- IAxisVMCrossSections
    - AddDoubleWedgedI

Obsolete functions:

- IAxisVMLoads
    - AddDomainConstant

New properties:

- IAxisVMSettings
    - NL_ConsequenceClass
- IAxisVMCrossSection
    - DoubleWedgedI

Changes between versions 15.4 and 16.1

New Errors:

- EDomainsError
    - deInvalidReinfParamForTrapezoidal
    - deInvalidHollowCoreDirection
    - deInvalidHollowCoreHoletype
    - deHollowCore*
    - deTrapezoidal*
    - deCompositeRib*
    - deNotARibbedDomain
- ESurfacesError
    - seInvalidReinfParamForTrapezoidal
- EActualReinforcementError
    - areReinfParamMissing
- EWindLoadError

Modified enum:

- ECombinationType
- ECriticalCombinationFormula
- ECrackWidth
- ECrossSectionShapeEx
- ELineError
- EMembersError
- ESpringParError

New enum:

- EDomainCompRibEccType
- EHollowHoleType
- ELEEccAlignementPoint
- ELEEccType
- EMemberLocXOrientation
- EReleasePosType
- ERoofFrictionEffect
- ERoofInternalPressure

- ERoofMultiSpanDir
- ERoofMultiSpanPos
- EVerticalDisplacement
- EWindStructureRoofType
- EXLAMServiceClass
- EXYDirection

Modified records:

- RReinforcementParameters_EC
- RReinforcementParameters_ITA
- RReinforcementParameters_SIA

New records:

- RBulkMemberWSSupport
- RBulkWSLineSupport
- RCommonCriticalResultsSettings_V161
- RCommonDisplayParameters_V161
- RCrossSectionHSQ
- RCrossSectionHSQA
- RCrossSection2IX
- RCrossSectionSFB
- RCrossSectionIFB
- RCrossSectionComposite
- RDomainCompositeRib
- RDomainHollowCore
- RDoubleLClosed
- REccReleases
- RLineAttr_V161
- RLoadDomainPolyAssoc_V161
- RRelease_V161
- RReleases_V161
- RSpringParam_V161
- RVerticalDisplacementValues
- RWindLoadParams_V161
- RWindSubStructParams
- RXLAMParams

Obsolete records:

- RLoadDomainConstant
- RLoadDomainPolyAssoc
- RWindLoadParameters

New functions:

- IAxisVMWindows
  - GetCommonDisplayParameters_V161
  - SetCommonDisplayParameters_V161
- IAxisVMCrossSections
  - Add2IX
  - AddComposite2IX
  - AddDoubleLClosed
  - AddFromCatalogTable
  - AddHSQ
  - AddHSQA
  - AddIFB
  - AddSFB
  - ReplaceWith2IX
  - ReplaceWithComposite2IX
  - ReplaceWithDoubleLClosed
  - ReplaceWithDoubleWedgedI
  - ReplaceWithHSQ
  - ReplaceWithHSQA

    - ReplaceWithIFB
    - ReplaceWithSFB
    - AddFromCatalog_V161
- IAxisVMLines
    - BulkGetAttr_V161
    - BulkSetAttr_V161
- IAxisVMLoads
    - AddDomainPolyAssoc_V161
    - AddNodalMass
    - DeleteNodalMass
    - ModifyNodalMass
- IAxisVMSurfaceSupport
    - GetPasternakStiffness
    - SetPasternakStiffness
- IAxisVMDomains
    - AddCompositeRib
    - AddHollowCore
    - AddTrapezoidal
    - AddXLAM
    - GetCompositeRibParameters
    - GetHollowCoreParameters
    - GetTrapezoidalParameters
    - GetXLAMParameters_V161
    - SetCompositeRibParameters
    - SetHollowCoreParameters
    - SetRibbedDomainActual
    - SetTrapezoidalParameters
    - SetXLAMParameters_V161
- IAxisVMDomainsSupports
    - AddDomainPasternakSupport
    - GetPasternakStiffness
    - SetPasternakStiffness
- IAxisVMDomainSupport
    - GetPasternakStiffness
    - SetPasternakStiffness
- IAxisVMSurfaceSupports
    - AddSurfacePasternakFoundation
- IAxisVMDomainSupports
    - AddDomainPasternakFoundation
- IAxisVMMember
    - ClearEccentricity
- IAxisVMMembers
    - BulkGetMembers_V161
    - BulkSetMembers_V161
    - ClearEccentricities
- IAxisVMMembersSupports
    - AddDomainEdgePasternakSupport
    - AddPasternakSupport
    - AddPasternakSupport
    - GetPasternakStiffness
    - SetPasternakStiffness
- IAxisVMLineSupports
    - AddBeamPasternakFoundation
    - AddEdgeGlobalPasternak
    - AddEdgeRelativePasternak
    - AddRibPasternakFoundation
    - BulkAddPasternak
- IAxisVMLineSupport
    - GetPasternakStiffness
    - SetPasternakStiffness
- IAxisVMLoads
    - GetSelectedItemIds

- IAxisVMSpringParams
  - Add_V161

Obsolete functions:
- IAxisVMLoadCases
  - CreateWindCases
  - GetWindLoadParams
  - SetWindLoadParams
- IAxisVMLoads
  - AddDomainConstant
  - AddDomainPolyAssoc
  - CreateWindLoadOnLoadPanels
  - DeleteWindLoadFromAllLoadPanels
  - DeleteWindLoadFromLoadPanels
  - GetLoadPanelsOfWindLoad

New properties:
- IAxisVMDomainsSupports
  - Item
- IAxisVMCrossSection
  - CrossSection2IX
  - CrossSectionHSQ
  - CrossSectionHSQA
  - CrossSectionIFB
  - CrossSectionSFB
  - DoubleLClosed
- IAxisVMDomains
  - IsCompositeRib
  - IsHollowCore
  - IsTrapezoidal
- IAxisVMResults
  - CalcCrackWidth
  - VerticalDisplacements
- IAxisVMSpringParam
  - FullRec_V161

New interfaces:
- IAxisVMCalcCrackWidth
- IAxisVMVerticalDisplacements
- IAxisVMWindLoad
- IAxisVMWindSubStructure

Changes between versions 16.1 and 16.2

Modified enum:
- EAnalysisType
- ECalculationError
- EDomainsError
- ELoadsError
- EMembersError
- EselectionType

New enum:
- EAnalysisCaseType

New Errors:
- ERCColumnCheckingError
  - rccceMixedDesignSituation
  - rccceErrorInFireDesignCalculation
- ELineError

    - lneSpringTypeIncompatible
    - lneSpringIndexOutOfBounds
- EMembersError
    - mbeSpringTypeIncompatible
    - mbeSpringIndexOutOfBounds

New records:

- RBuckling_V162
- RDynamicAnalysis_V162
- RNonLinearAnalysis_V162
- RPushOverAnalysis_V162
- RResultCase
- RVibration_V162

Obsolete records:

- RBuckling
- RDynamicAnalysis
- RNonLinearAnalysis
- RPushOverAnalysis
- RVibration

New functions:

- IAxisVMCalculation
    - Buckling_V162
    - Buckling2_V162
    - DynamicAnalysis_V162
    - NonLinearAnalysis_V162
    - NonLinearAnalysis2_V162
    - NonLinearVibration_V162
    - NonLinearVibration2_V162
    - PushOverAnalysis_V162
    - Vibration_V162
    - Vibration2_V162
- AxisVMLines
    - PointOnLine
- IAxisVMLines3d
    - StraightLines
- IAxisVMLoads
    - AddLoadPanelLinear_V162
- IAxisVMMembers
    - AssembleMembers
    - SelectArray
- IAxisVMModel
    - PickCoordinate
- IAxisVMResults
    - GetBucklingParameters_V162
    - GetDynamicParameters
    - GetNonLinearAnalysisParameters_V162
    - GetNonLinearVibrationParameters_V162
    - GetPushOverParameters
    - GetVibrationParameters_V162
- IAxisVMWindow
    - GetResultCase
- IAxisVMWindows
    - GetResultCase

Obsolete functions:

- IAxisVMCalculation
    - Buckling
    - Buckling2
    - DynamicAnalysis
    - NonLinearAnalysis

    - NonLinearAnalysis2
    - NonLinearVibration
    - NonLinearVibration2
    - PushOverAnalysis
    - Vibration
    - Vibration2
- IAxisVMResults
    - GetBucklingParameters
    - GetNonLinearAnalysisParameters
    - GetNonLinearVibrationParameter
    - GetVibrationParameters

Changes between versions 16.2 and 17.1

New enum:

- ELargeRoofModeSIA

New records:

- RCSOptimizationChecks_V171
- RCSParametricOptimizationParams_V171
- RDomainMeshParameters_V171
- RSeismicSensitivityResults_V171
- RSnowLoadParams_V171
- RTimberDesignParameters_EC_V171
- RTimberDesignParameters_V171

Obsolete records:

- RCSOptimizationChecks
- RCSParametricOptimizationParams
- RDomainMeshParameters
- RSeismicSensitivityResults
- RSnowLoadParams
- RTimberDesignParameters_EC_V153
- RTimberDesignParameters_V153

New properties:

- IAxisVMSettings
    - NodeBreaksUnmeshedMember

New functions:

- IAxisVMCrossSectionOptimization
    - GetOptimizationChecks_V171
    - GetParametersForParametricOptimization_V171
    - SetOptimizationChecks_V171
    - SetParametersForParametricOptimization_V171
- IAxisVMDomain
    - GenerateMesh_V171
    - GetMeshParameters_V171
    - SetMeshParameters_V171
- IAxisVMDomains
    - GenerateMeshOnSelectedDomains_V171
- IAxisVMLoadCases
    - CreateSnowCases_V171
    - DeleteSnowLoadCases
    - GetSnowLoadParams_V171
    - SetSnowLoadParams_V171
- IAxisVMSeismicStoreys
    - GetSeismicSensitivityResults_V171
- IAxisVMSteelDesignMembers
    - BulkAdd

- IAxisVMTimberDesignMembers
    - Add_V171
    - BulkAdd
    - GetDesignParameters_V171
    - SetDesignParameters_V171

Obsolete functions:

- IAxisVMCrossSectionOptimization
    - GetOptimizationChecks
    - GetParametersForParametricOptimization
    - SetOptimizationChecks
    - SetParametersForParametricOptimization
- IAxisVMDomain
    - GenerateMesh
    - GetMeshParameters
    - SetMeshParameters
- IAxisVMDomains
    - GenerateMeshOnSelectedDomains
- IAxisVMLoadCases
    - CreateSnowCases
    - GetSnowLoadParams
    - SetSnowLoadParams
- IAxisVMSeismicStoreys
    - GetSeismicSensitivityResults
- IAxisVMTimberDesignMembers
    - Add_V153
    - GetDesignParameters_V153
    - SetDesignParameters_V153

Changes between versions 17.1 and 17.2

Modified enum:

- ECalculationError
- EGeneralError
- ELoadType
- ESpringParType
- ESteelDesignMemberError
- ETimberSLSDesignCreepMode

New enum:

- EMemberLoadEcc
- ESteelBucklingCurves
- ESteelLoadPosition
- ESteelLTBucklingCurves

New records:

- REdgeConnectionRec _V172
- RLoadMemberConcentrated
- RLoadMemberDistributed
- RSteelDesignParameters_EC_SIA_ITA_V172
- RSteelDesignParameters_V172
- RTimberDesignParameters_V172

Obsolete records:

- REdgeConnectionRec
- RLoadBeamMemberConcentrated
- RLoadBeamMemberDistributed
- RLoadRibMemberConcentrated
- RLoadRibMemberDistributed

- RRCBeamDesignParameters_DIN
- RRCBeamDesignParameters_MSZ
- RRCBeamDesignParameters_STAS
- RSteelDesignParameters_EC_SIA_ITA_V153
- RSteelDesignParameters_V153
- RTimberDesignParameters_V171

New properties:
- I AxisVMStresses
    - IndependentStressPoints

New functions:
- IAxisVMEdgeConnections
    - Add _V172
    - GetRec_V172
    - SetRec_V172
- IAxisVMLoads
    - AddMemberConcentrated
    - AddMemberDistributed
- IAxisVMSteelDesignMembers
    - Add_V172
    - BulkAdd_V172
    - GetDesignParameters_V172
    - SetDesignParameters_V172
- IAxisVMTimberDesignMembers
    - Add_V172
    - BulkAdd_V172
    - GetDesignParameters_V172
    - SetDesignParameters_V172

Obsolete functions:
- IAxisVMEdgeConnections
    - Add
    - GetRec
    - SetRec
- IAxisVMLoads
    - AddBeamMemberConcentrated
    - AddBeamMemberDistributed
    - AddRibMemberConcentrated
    - AddRibMemberDistributed
- IAxisVMSteelDesignMembers
    - Add_V153
    - BulkAdd
    - GetDesignParameters_V153
    - SetDesignParameters_V153
- IAxisVMTimberDesignMembers
    - Add_V171
    - BulkAdd
    - GetDesignParameters_V171
    - SetDesignParameters_V171

Changes between versions 17.2 and 17.3

Modified enum:
- ELoadCaseType
- ELoadGroupType

New enum:
- EColorLegendDirection
- EColorLegendType

New records:

- RRCBeamDesignParameters_V173

Obsolete records:

- RRCBeamDesignParameters

New functions:

- IAxisVMRCBeamDesign
  - GetDesignParameters_V173
  - SetDesignParameters_V173

Obsolete functions:

- IAxisVMRCBeamDesign
  - GetDesignParameters
  - SetDesignParameters

New interfaces:

- IAxisVMColorLegend

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::IAxisVMApplication* AxApp);
unsigned int32 IsMenuItemEnabled(AxisVM::IAxisVMApplication* AxApp);
unsigned int32 IsMenuItemVisible(AxisVM::IAxisVMApplication* AxApp);

// v2.1
unsigned int32 IsPluginModalWindow(void);

// v2.2
__stdcall void InitPlugin(AxisVM::IAxisVMApplication* 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: IAxisVMApplication); stdcall;
function IsMenuItemEnabled(AxApp: IAxisVMApplication) : DWORD; stdcall;
function IsMenuItemVisible(AxApp: IAxisVMApplication) : DWORD; stdcall;
```

```
// v2.1
function IsPluginModalWindow : DWORD; stdcall;

// v2.2
procedure InitPlugin (AxApp: IAxisVMApplication); 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::IAxisVMApplication* **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 IAxisVMApplication 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 **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.

---

unsigned long **IsMenuItemEnabled** ([in] AxisVM::IAxisVMApplication* **AxApp**)

AxApp *AxisVM COM object implementing IAxisVMApplication 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::IAxisVMApplication* **AxApp**)

AxApp *AxisVM COM object implementing IAxisVMApplication interface*

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.

unsigned long **IsPluginModalWindow**

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.

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.

void **InitPlugin** ([in] AxisVM::IAxisVMApplication* **AxApp**)

AxApp *AxisVM COM object implementing IAxisVMApplication interface*

This function is called by AxisVM when AxisVM is fully loaded (IAxisVMApplication 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.

void **DeinitPlugin**

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.

.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::IAxisVMApplication* **AxApp**)

AxApp *AxisVM COM object of the application in which this plugin resides*

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.

---

long **IsMenuItemVisible** ([in] AxisVM::IAxisVMApplication* **AxApp**)

AxApp *AxisVM COM object of the application in which this plugin resides*

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.

---

long **MenuItemClick_v2** ([in] AxisVM::IAxisVMApplication* **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. 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.

---

long **IsPluginModalWindow**

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.

---

void **InitPlugin** ([in] AxisVM::IAxisVMApplication* **AxApp**)

AxApp *AxisVM COM object of the application in which this plugin resides*

This function is called by AxisVM when AxisVM is fully loaded (IAxisVMApplication Loaded event is fired) and COM server is available.

---

void **DeinitPlugin**

This function is called before AxisVM unloads plugin dlls when AxisVM application is shutting down.

String **GetTranslatedMenuItemText** ([in] ELanguage **PrgLang**)

PrgLang *the current program language of AxisVM*

This function is called by AxisVM each time when the program language changed. The plugin can specify language specific plugin menu item text.

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 AxisVMApplicationHandle { get; set; }
      int IsMenuItemEnabled(AxisVMApplicationClass iAxisVMApp);
      int IsMenuItemVisible(AxisVMApplicationClass iAxisVMApp);
      int MenuItemClick_v2(AxisVMApplicationClass iAxisVMApp);
   }
```

VB

```
Public Interface IAxisVMDotNetPluginInterface_v2
    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
End Interface
```

VC++

```
public interface class IAxisVMDotNetPluginInterface_v2
{
       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);
};
```

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::IAxisVMApplication * iAxApp);

// version 2.0: functions below are available only from AxisVM 12 release 2
unsigned int __stdcall IsItemEnabled (AxisVM::IAxisVMApplication * iAxApp);
unsigned int __stdcall IsItemVisible (AxisVM::IAxisVMApplication * iAxApp);
void __stdcall InitAddOn (AxisVM::IAxisVMApplication * 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 SetAxisVMApplicationHandle(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: IAxisVMApplication); stdcall;

// version 2.0: functions below are available only from AxisVM 12 release 2
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;
procedure SetAxisVMMainFormHandle(const FormHandle: THandle); stdcall;
procedure SetAxisVMApplicationHandle(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 IAxisVMApplication 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::IAxisVMApplication * **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 IAxisVMApplication argument otherwise (because of the single instance mode of the COM server) it will start another instance of AxisVM.***

---

unsigned int **IsItemEnabled**([in] AxisVM::IAxisVMApplication * **AxApp**)

AxApp *AxisVM COM object pointer implementing IAxisVMApplication 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::IAxisVMApplication * **AxApp**)

AxApp *AxisVM COM object pointer implementing IAxisVMApplication 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::IAxisVMApplication* **AxApp**)

AxApp *AxisVM COM object implementing IAxisVMApplication interface*

This function is called by AxisVM when AxisVM is fully loaded (IAxisVMApplication 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 SetAxisVMApplicationHandle(const unsigned int32 ApplicationHandle);
void OnMenuItemClick_v2(AxisVM::IAxisVMApplication* AxApp);
unsigned int32 IsMenuItemEnabled(AxisVM::IAxisVMApplication* AxApp);
unsigned int32 IsMenuItemVisible(AxisVM::IAxisVMApplication* AxApp);
unsigned int32 IsPluginModalWindow(void);
__stdcall void InitPlugin(AxisVM::IAxisVMApplication* 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::IAxisVMApplication * iAxApp);
unsigned int __stdcall IsItemEnabled (AxisVM::IAxisVMApplication * iAxApp);
unsigned int __stdcall IsItemVisible (AxisVM::IAxisVMApplication * iAxApp);
void __stdcall InitAddOn (AxisVM::IAxisVMApplication * 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 SetAxisVMApplicationHandle(const ApplicationHandle: THandle); stdcall;
procedure OnMenuItemClick_v2(AxApp: IAxisVMApplication); stdcall;
function IsMenuItemEnabled(AxApp: IAxisVMApplication) : DWORD; stdcall;
function IsMenuItemVisible(AxApp: IAxisVMApplication) : DWORD; stdcall;
function IsPluginModalWindow : DWORD; stdcall;
procedure InitPlugin (AxApp: IAxisVMApplication); 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 AxisVMApplicationHandle { get; set; }

        // plugin
        int Plugin_Execute (AxisVMApplicationClass iAxisVMApp);
        string Plugin_GetTranslatedMenuItemText(int PrgLang);
        void Plugin_Deinit(AxisVMApplicationClass iAxisVMApp);
        void Plugin_Init(AxisVMApplicationClass iAxisVMApp);
        int Plugin_IsMenuItemEnabled(AxisVMApplicationClass iAxisVMApp);
        int Plugin_IsMenuItemVisible(AxisVMApplicationClass iAxisVMApp);
        int Plugin_IsModalWindow { get; }
        String Plugin_MenuItemText { get; }

        // addon
        int Addon_ButtonPlaceFormId { get; }
        int Addon_ButtonPlaceToolbarId { get; }
        int Addon_Execute(AxisVMApplicationClass iAxisVMApp);
        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(AxisVMApplicationClass iAxisVMApp);
        void Addon_Init(AxisVMApplicationClass iAxisVMApp);
        int Addon_IsItemEnabled(AxisVMApplicationClass iAxisVMApp);
        int Addon_IsItemVisible(AxisVMApplicationClass iAxisVMApp);

   }
```

VB

```
Public Interface IAxisVMDotNetAddonPluginInterface_v1_0
        ' common
        ReadOnly Property Version () As Integer
        ReadOnly Property SubVersion () As Integer
        Property AxisVMMainFormHandle () As Integer
        Property AxisVMApplicationHandle () As Integer

        ' plugin
        Function Plugin_Execute (ByVal iAxisVMApp As AxisVMApplicationClass) As Integer
        ReadOnly Property Plugin_GetTranslatedMenuItemText (PrgLang as Integer) As String
        sub Plugin_Deinit (ByVal iAxisVMApp As AxisVMApplicationClass)
        sub Plugin_Init (ByVal iAxisVMApp As AxisVMApplicationClass)
        Function Plugin_IsMenuItemEnabled(ByVal iAxisVMApp As AxisVMApplicationClass) As Integer
        Function Plugin_IsMenuItemVisible(ByVal iAxisVMApp As AxisVMApplicationClass) 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 iAxisVMApp As AxisVMApplicationClass) 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 iAxisVMApp As AxisVMApplicationClass)
        sub Addon_Init (ByVal iAxisVMApp As AxisVMApplicationClass)
        Function Addon_IsItemEnabled(iAxisVMApp As AxisVMApplicationClass) As Integer
        Function Addon_ IsItemVisible (iAxisVMApp As AxisVMApplicationClass) 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**

Must return 1

---

long **SubVersion**

Must return 0

---

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 **Plugin_Execute** ([in] AxisVM::IAxisVMApplication* **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. 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.

String **Plugin_GetTranslatedMenuItemText** ([in] ELanguage **PrgLang**)

PrgLang *the current program language of AxisVM*

This function is called by AxisVM each time when the program language changed. The AddonPlugin can specify language specific menu item text.

---

void **Plugin_Init** ([in] AxisVM::IAxisVMApplication* **AxApp**)

AxApp *AxisVM COM object of the application in which this AddonPlugin resides*

This function is called by AxisVM when AxisVM is fully loaded (IAxisVMApplication Loaded event is fired) and COM server is available.

---

void **Plugin_Deinit** ([in] AxisVM::IAxisVMApplication* **AxApp**)

AxApp *AxisVM COM object of the application in which this AddonPlugin resides*

This function is called before AxisVM unloads plugin dlls when AxisVM application is shutting down.

---

long **Plugin_IsMenuItemEnabled** ([in] AxisVM::IAxisVMApplication* **AxApp**)

AxApp *AxisVM COM object of the application in which this AddonPlugin resides*

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.

---

long **Plugin_IsMenuItemVisible** ([in] AxisVM::IAxisVMApplication* **AxApp**)

AxApp *AxisVM COM object of the application in which this AddonPlugin resides*

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.

---

long **Plugin_IsModalWindow**

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 IAxisVMApplication interface. Modal AddonPlugins are only allowed for legacy reasons, for new developments non modal mode (i.e. return value 0) is highly recommended.
A modal AddonPlugin must return 1 and a non-modal must return 0.

String **Plugin_MenuItemText**

Unicode string that represents the name of the plugin. This will be displayed in AxisVM plugin menu as menu item.

---

long **Addon_ButtonPlaceFormId**

Determines the id of the form where addon button will be displayed. List of Form IDs where to display the addon toolbar button.

---

long **Addon_ButtonPlaceToolbarId**

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).

---

long **AddOn_Execute** ([in] AxisVM::IAxisVMApplication * **AxApp**)

AxApp *COM object pointer of the AxisVM application which initiated the execution*

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.

---

long **Addon_FormHandle**

This function is called by AxisVM to pass the handle of the form window where the button is.

String **Addon_GetTranslatedHintText** ([in] ELanguage **PrgLang**)

PrgLang *the current program language of AxisVM*

Returns AddonPlugin' 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.

String **Addon_HintText**

Returns AddonPlugin's hint (text). Hint is shown when cursor hovers over the Addon's button.

---

byte[] **Addon_IconBitmap**

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).

---

long **Addon_IconBitmapTransparentColor**

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

---

long **Addon_IsModalWindow**

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 IAxisVMApplication interface. Modal AddonPlugins are only allowed for legacy reasons, for new developments non modal mode (i.e. return value 0) is highly recommended.
A modal AddonPlugin must return 1 and a non-modal must return 0.

---

long **Addon_Deinit** ([in] AxisVM::IAxisVMApplication* **AxApp**)

AxApp *AxisVM COM object of the application in which this AddonPlugin resides*

This function is called before AxisVM unloads Addon dlls when AxisVM application is shutting down.

---

long **Addon_Init** ([in] AxisVM::IAxisVMApplication* **AxApp**)

AxApp *AxisVM COM object of the application in which this AddonPlugin resides*

This function is called by AxisVM when AxisVM is fully loaded (IAxisVMApplication Loaded event is fired) and COM server is available.

---

long **Addon_IsItemEnabled** ([in] AxisVM::IAxisVMApplication* **AxApp**)

AxApp *AxisVM COM object of the application in which this AddonPlugin resides*

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.

---

long **Addon_IsItemVisible** ([in] AxisVM::IAxisVMApplication* **AxApp**)

AxApp *AxisVM COM object of the application in which this AddonPlugin resides*

AxisVM will call this routine when AddonPlugin's button is about to displayed. If it returns 0 then the button won't be displayed otherwise it will be displayed.

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 acces 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 conrete 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

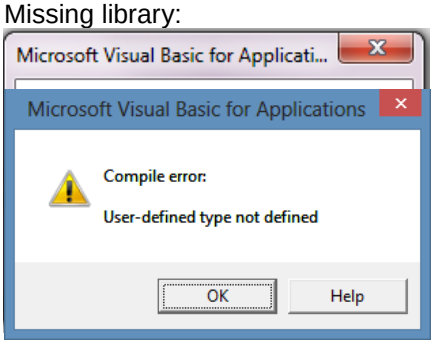
Example of using SafeArrays in VB.NET:

```
'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

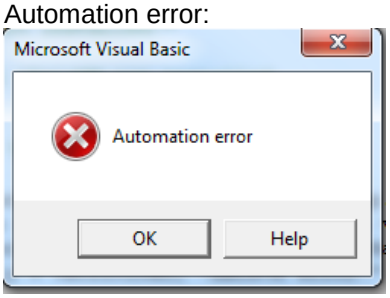
Users can get several types of error messages when the AxisVM type library is not properly linked to Excel like these:

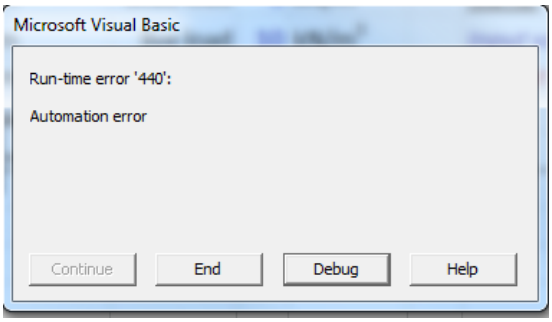
Missing library:



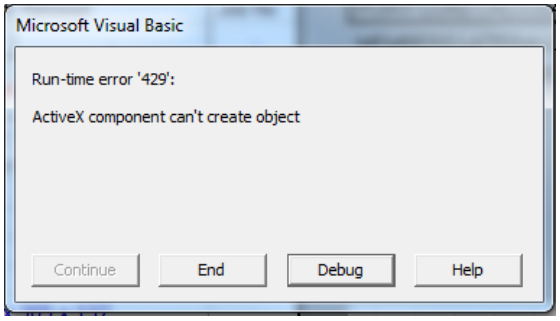
or Compile error:

Automation error:

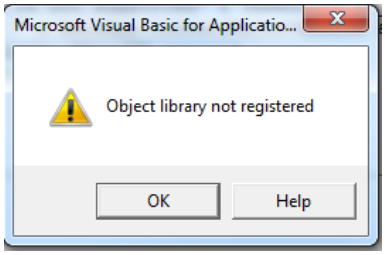




Run-time error



Object library error



Please note:

Other types of error messages can be also fixed by trying the steps below.

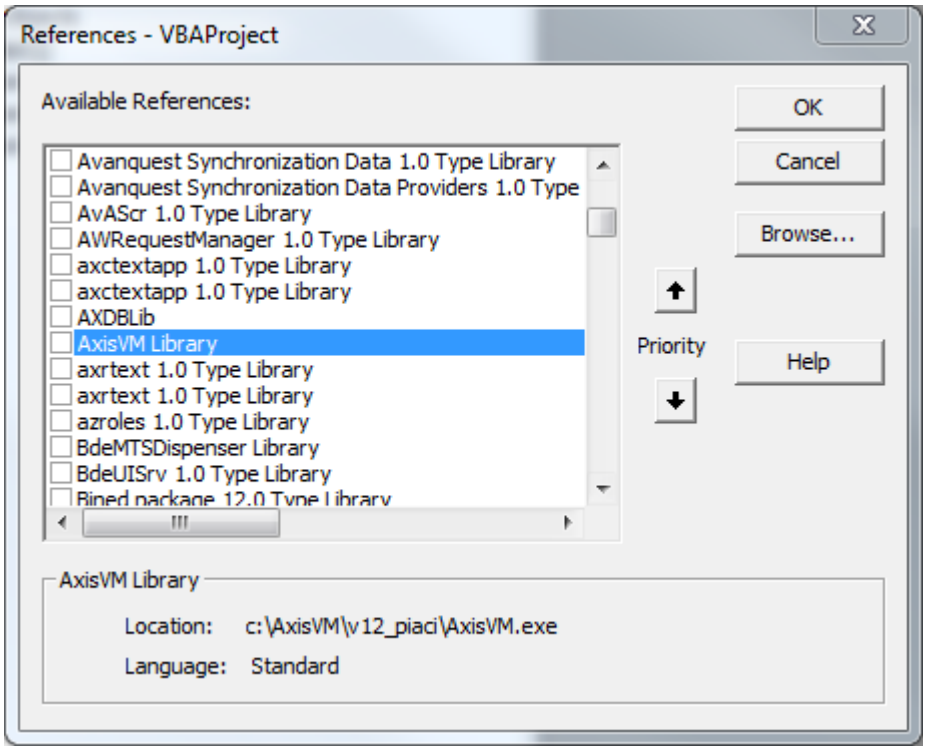
Solution:

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 remains 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.

SafeArray problems :
VBA cannot handle *[in] SafeArray (Type) Name* parameters. For some early functions, _vb versions of those functions were created, where *[in] SafeArray (Type) Name* was replaced with *[i/o] SafeArray (Type) * Name*. For example :

IAxisVMColumnRebars.SetRebars ([in] long Index, [in] SAFEARRAY(RColumnRebarPos) Rebars)
IAxisVMColumnRebars.SetRebars_vb([in] long Index, [in, out] SAFEARRAY(RColumnRebarPos) * Rebars)

Such efforts were stopped. Use VBA only for simple scripts, for serious development a proper language/development environment is recommended (for example C#, C++, Delphi), because VBA stopped supporting all the allowed COM types.

Unsigned problems :
VBA cannot handle *unsigned* parameters, like unsigned long. For some early functions, _vb versions of those functions were created, where *unsigned* was replaced with the corresponding signed one. For example :

unsigned long IAxisVMSurface. MaterialColour
long IAxisVMSurface. MaterialColour_vb

Such efforts were stopped. Use VBA only for simple scripts, for serious development a proper language/development environment is recommended (for example C#, C++, Delphi), because VBA stopped supporting all the allowed COM types.

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": "IAxisVMApplication",
        "FunctionName": "SetEmptySafeArrays",
        "Value": False
}
```

      Check this option with JSON string:

```
{
        "InterfaceName": "IAxisVMApplication",
        "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 IAxisVMApplication interface.
- Use global variables for instances of often used interfaces (IAxisVMApplication, IAxisVMModels, IAxisVMModel) 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 IAxisVMModels interface , two instances of IAxisVMModel 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 IAxisVMModels interface , one instance of IAxisVMModel 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 IAxisVMModels
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

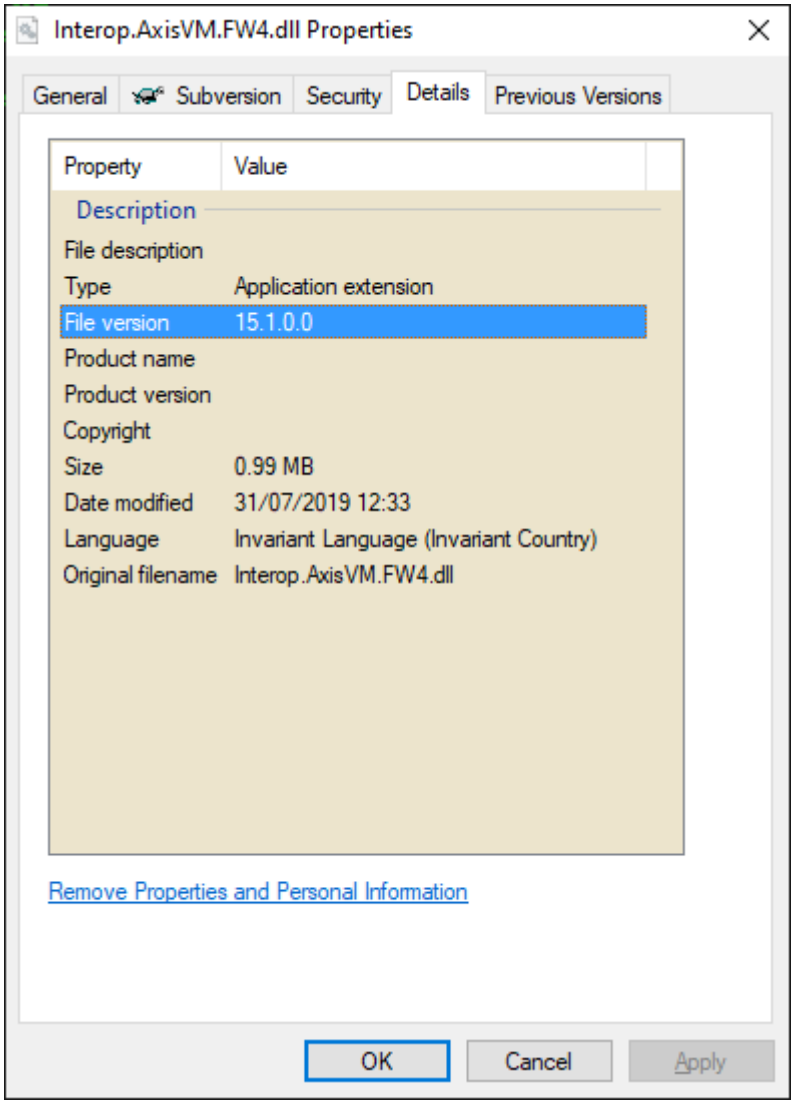
Type library should be imported or linked to the project in order to get the constant names, object structures and function calling syntax.

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).

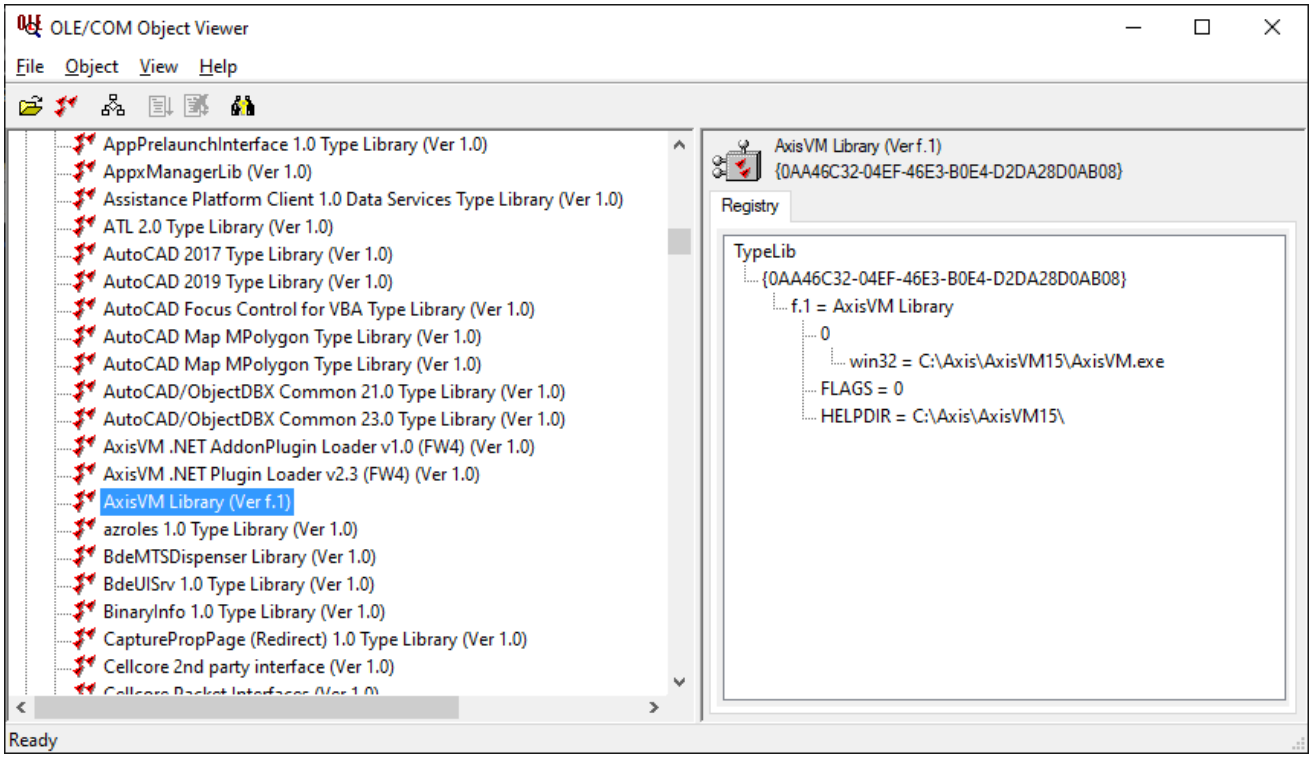
Determining the major and minor version of the COM server

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 means 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
```

An alternative way to import the previously generated library is :

```
from comtypes.gen import AxisVM as ax
```

Referencing a constant can be done as :

```
ax.lgtStraightLine,
```

referencing a record as :

```
ax.RlineGeomData
```

Starting a dedicated COM server is done by:

```
axApp = comtypes.client.CreateObject('AxisVM.AxisVMApplication')
```

Connecting to an already running AxisVM application, started with the /multiinstancecomclients parameter is done by :

```
axApp = comtypes.client.GetActiveObject('AxisVM.AxisVMApplication')
```

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 Definion 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 hMainForm = 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)
{
hMainForm = (HWND)FormHandle;
}
```

The following procedure will be called when the user clicks the *Visual C plugin sample* menu item.

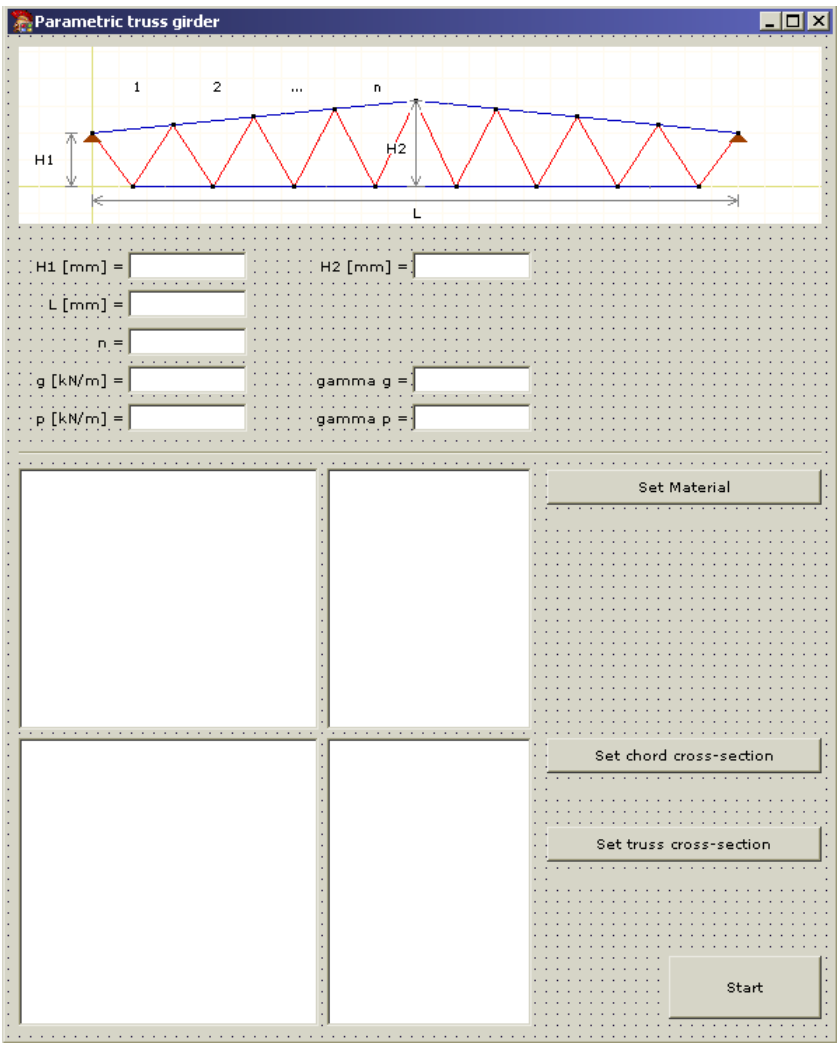
```
void OnMenuItemClick(void)
{
BOOL fgNeedToUninitialize;
int n;
unsigned int number;
       AxisVM::IAxisVMModelsPtr      iAxisModels;
       AxisVM::IAxisVMModelPtr       iAxisModel;
       AxisVM::IAxisVMLinesPtr       iAxisLines;
       AxisVM::RLineGeomData         rLineGeomData;
       AxisVM::RPoint3d              rP1, rP2;

fgNeedToUninitialize = CoInitializeEx(NULL, COINIT_APARTMENTTHREADED) == S_OK;
// create AxisVM COM server: always create 'AxisVMApplication' first!
AxisVM::IAxisVMApplicationPtr        iAxisApp(__uuidof(AxisVM::AxisVMApplication));
// 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 prespective view and fit view
iAxisModel->View = AxisVM::vPerspective;
iAxisModel->FitInView();

if (fgNeedToUninitialize)
CoUninitialize();
MessageBox(hMainForm, 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_AxisVMApplication, nil, CLSCTX_ALL, IID_IAxisVMApplication,
          fAxApp));
fAxApp.AskCloseOnLastReleased := True;
// wait until fully loaded
while fAxApp.Loaded = lbFalse do
  Sleep(100);
```

Where `fAxApp: AxisVMApplication` is a private field of the form, `CLASS_AxisVMApplication` and `IID_IAxisVMApplication` 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, lnlTensionAndCompression, 0);

lineid := AxLines.Add(LowerNodes[i], UpperNodes[i+1], lgtStraightLine, GeomData);
AxLine := AxLines.Item[lineid];
AxLine.DefineAsTruss(MaterialId, TrussCSId, lnlTensionAndCompression, 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  := lnlTensionAndCompression; Resistances.x  := 0;
Stiffnesses.y := 1e10; Nonlinearity.y  := lnlTensionAndCompression; Resistances.y  := 0;
Stiffnesses.z := 1e10; Nonlinearity.z  := lnlTensionAndCompression; Resistances.z  := 0;
Stiffnesses.xx := 1e8; Nonlinearity.xx := lnlTensionAndCompression; Resistances.xx := 0;
Stiffnesses.yy := 1e8; Nonlinearity.yy := lnlTensionAndCompression; Resistances.yy := 0;
Stiffnesses.yy := 1e8; Nonlinearity.zz := lnlTensionAndCompression; 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:

```
AxisVMModels ^AxModels;
AxisVMModel ^AxModel;
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 ‘AxisVMApplication’ first!
Dim AxisApp As New AxisVM.AxisVMApplication
‘ 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 nodalsuppors 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.RResitances
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.

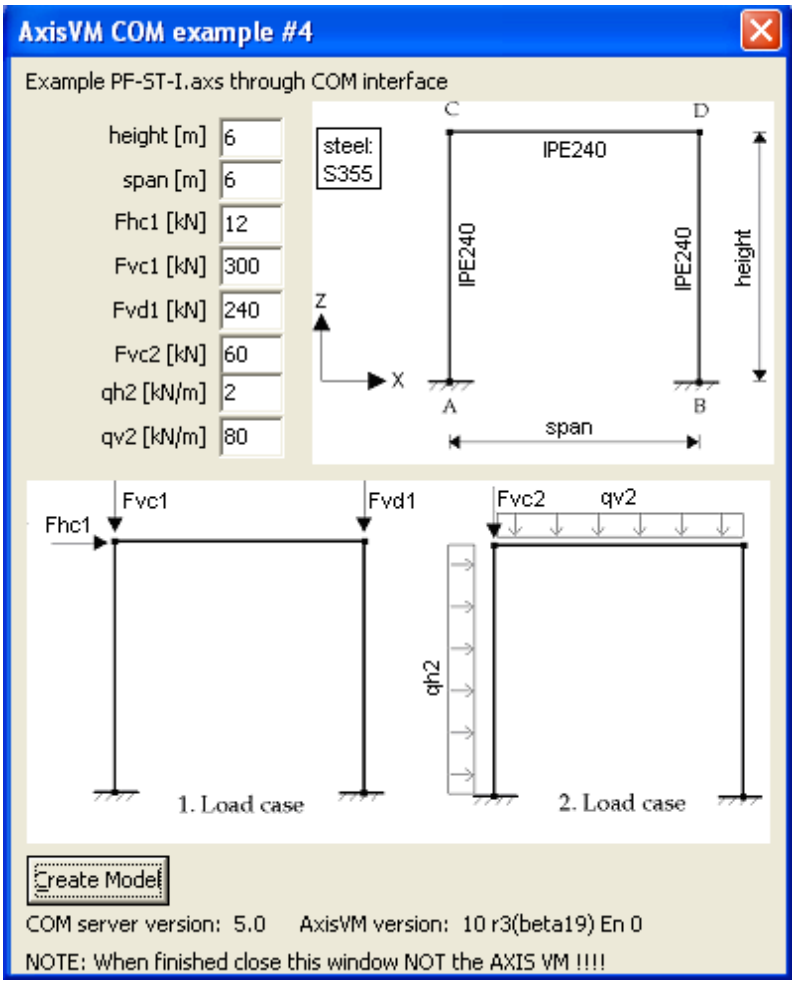
```
‘add beam distributed load to LoadCase #1
Dim rLoadBeamDist As AxisVM.RLoadBeamDistributed
rLoadBeamDist.qx1 = 0.0
rLoadBeamDist.qy1 = 0.0
rLoadBeamDist.qz1 = -200.0
rLoadBeamDist.mx1 = 0.0
rLoadBeamDist.my1 = 0.0
rLoadBeamDist.mz1 = 0.0
rLoadBeamDist.qx2 = 0.0
rLoadBeamDist.qy2 = 0.0
rLoadBeamDist.qz2 = -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 distibuted 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

ErrHandler:
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(AxisApp)

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  := lnlTensionAndCompression; Resistances.x  := 0;
Stiffnesses.y := 1e10; Nonlinearity.y  := lnlTensionAndCompression; Resistances.y  := 0;
Stiffnesses.z := 1e10; Nonlinearity.z  := lnlTensionAndCompression; Resistances.z  := 0;
Stiffnesses.xx := 1e10; Nonlinearity.xx := lnlTensionAndCompression; Resistances.xx := 0;
Stiffnesses.yy := 1e10; Nonlinearity.yy := lnlTensionAndCompression; Resistances.yy := 0;
Stiffnesses.zz := 1e10; Nonlinearity.zz := lnlTensionAndCompression; 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:=brdtLength; //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 begining 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:=brdtLength; //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 ST' needs ro be preserved at beginning will be deleleted afterwards
Factors[0] := 0;   // ST1  factor=0
LoadCaseIds[0] := 1; // ST1
Factors[1] := 1.35;
LoadCaseIds[1] := lc1;
Factors[2] := 1.5;
LoadCaseIds[2] := lc2;
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(lc1);
dec(lc2);
```

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:=lc1;
nodeID:=ANOdes[1];//point C
AxisVMDisplacements.LoadCaseId:= loadcaseID;
j:=AxisVMDisplacements.NodalDisplacementByLoadCaseId(nodeID,DisplacementValues,lcName);
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,lcName);
My_a1:=LineForceValues.lfvMy;
```

Read horizontal displacement at point C from loadcase 2

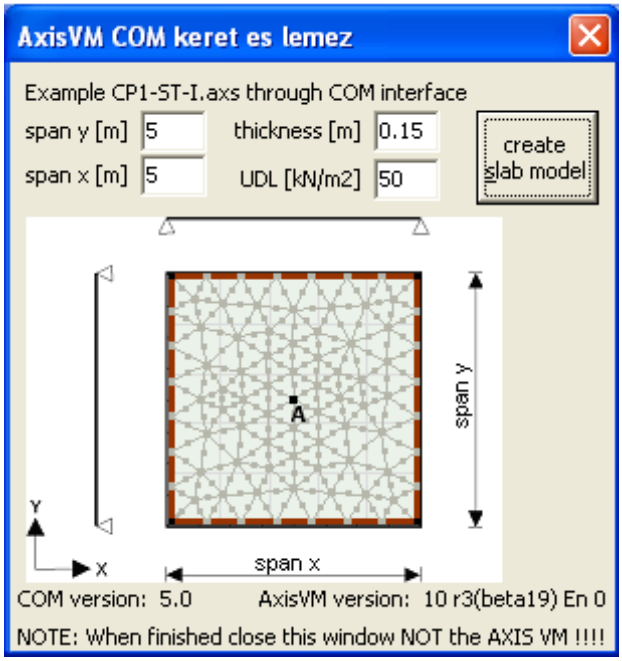
```
loadcaseID:=lc2;
nodeID:=ANOdes[1];//point C
AxisVMDisplacements.LoadCaseId:= loadcaseID;
j:=AxisVMDisplacements.NodalDisplacementByLoadCaseId(nodeID,DisplacementValues,lcName);
ex_c2:=DisplacementValues.Ex*1000; //from metres to mm
```

Read moment My at point A from loadcase 2LineID:=ALines[0];

```
SectionID:=1;  //for beginning of line
PosX:=0;
AxisVMForces.LoadCaseId:= loadcaseID;
i:=AxisVMForces.LineForceByLoadCaseId(LineID,SectionID,LineForceValues,PosX,lcName);
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  := lnlTensionAndCompression; Resistances.x  := 0;
Stiffnesses.y := 1e10; Nonlinearity.y  := lnlTensionAndCompression; Resistances.y  := 0;
Stiffnesses.z := 1e10; Nonlinearity.z  := lnlTensionAndCompression; Resistances.z  := 0;
Stiffnesses.xx := 0; Nonlinearity.xx := lnlTensionAndCompression; Resistances.xx := 0;
Stiffnesses.yy := 0; Nonlinearity.yy := lnlTensionAndCompression; Resistances.yy := 0;
Stiffnesses.zz := 0; Nonlinearity.zz := lnlTensionAndCompression; 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:AInteger;
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;
```

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 by 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.IAxisVMApplication)
except:
  sys.exit()
```