DIgSILENT

PowerFactory 2017

Technical Reference Documentation
Python Function Reference

INTEGRATED POWER SYSTEM ANALYSIS SOFTWARE FOR
TRANSMISSION / DISTRIBUTION / INDUSTRY / GENERATION / INTEGRATION OF RENEWABLES

Publisher:
DIgSILENT GmbH
Heinrich-Hertz-Straße 9
72810 Gomaringen / Germany
Tel.: +49 (0) 7072-9168-0
Fax: +49 (0) 7072-9168-88
info@digsilent.de

Please visit our homepage at:
http://www.digsilent.de

PowerFactory 2017 Service Pack 6 (17.0.8)
January 11, 2018

Contents

1 General Description

This reference manual describes the syntax of all available functions and methods provided by the *PowerFactory* module. The used Python interface version is 2 (new in *PowerFactory* 2016). For syntax of the Python interface version 1 (*PowerFactory* 15.x) please use the DPL reference with the extended return values.

Please refer to the *PowerFactory* User Manual for general information about Python as scripting language and its usage.

2 *PowerFactory* Module

Overview

GetApplication

Creates a *PowerFactory* application object and returns it. When the Python script is started from external *PowerFactory* will be started.

```
Application PowerFactory.GetApplication([str username = None,]
                                         [str password = None,]
                                         [str commandLineArguments = None])
```

ARGUMENTS

username (optional)
Name of the user to log in to *PowerFactory* (default None). None enforces the default behaviour as if PowerFactory was started via shortcut.

password (optional)
The password for the user which should be logged in (default None). None omits the password.

commandLineArguments (optional)
Additional command line options (default None). These need to be specified in the same way as if *PowerFactory* was started via a command shell. None omits the command line arguments.

RETURNS

Application object on success, otherwise None.

GetApplicationSecured

Same as PowerFactory.GetApplication() but using the password hash (see user edit dialog) instead of the password itself.

```
Application PowerFactory.GetApplicationSecured([str username = None,]
                                                [str passwordHash = None,]
                                                [str commandLineArguments = None])
```

3 Application Methods

Overview

ActivateProject

Activates a project with its name.

```
int Application.ActivateProject(str name)
```

Arguments

name	Name (”Project”), full qualified name (”Project.IntPrj”) or full qualified path (”\User\Project.IntPrj”) of a project.

Returns

0 on success and 1 if project can not be found or activated.

CommitTransaction

Writes pending changes to database.

While a script is running none of the changes are written to the database unless the script terminates. *PowerFactory* can be forced to write all pending changes to the database using this function.

```
None Application.CommitTransaction()
```

CreateFaultCase

Create fault cases from the given elements.

```
int Application.CreateFaultCase(set elms,
                                int mode,
                                [int createEvt],
                                [object folder]
                                )
```

ARGUMENTS

elms Selected elements to create fault cases.

mode How the fault cases are created:

0	Single fault case containing all elements.
1	n-1 (multiple cases).
2	n-2 (multiple cases).
3	Collecting coupling elemnts and create fault cases for line couplings.

createEvt (optional)
Switch event:

0	Do NOT create switch events.
1	Create switch events.

folder (optional)
Folder in which the fault case is stored.

RETURNS

0	On success.
1	On error.

CreateProject

Creates a new Project inside the parent object. The default project stored in the Configuration/Default folder will be copied and if it contains any Study Cases the first will be used instead of creating a new one. A new grid will always be created. Returns the newly created project.

```
DataObject Application.CreateProject(str projectName,
                                     str gridName,
                                     [DataObject parent])
```

ARGUMENTS

projectName
Name of the new project. Leave empty to open up the IntPrj dialog and let the user enter a name.

gridName
Name of the grid that‘s created for the new project. Leave empty to open up the ElmNet dialog and let the user enter a name.

parent The parent for the new project. Can be omitted to use the currently logged on user as default.

ExecuteCmd

Executes given command string as it would be executed if typed directly into the Input Window. Current script will continue after the command has been executed.
This function is mainly intended for testing purpose and should be used by experienced users only.

```
int Application.ExecuteCmd(str command)
```

ARGUMENTS

command
The command string

GetActiveCalculationStr

Gets “calculation string” of currently valid calculation.

```
str Application.GetActiveCalculationStr()
```

RETURNS

None basic

Load Flow ldf

AC Load Flow Sensitivities acsens

AC Contingency Analysis accont

DC Load Flow dcldf

DC Load Flow Sensitivities dcsens

DC Contingenciy Analysis dccont

VDE/IEC Short-Circuit shc

Complete Short-Circuit shcfull

ANSI Short-Circuit shcansi

IEC 61363 shc61363

RMS-Simulation rms

Modal Analysis modal

EMT-Simulation emt

Harmonics/Power Quality harm

Frequency Sweep fsweep

Optimal Power Flow	opf
DC Optimal Power Flow	dcopf
DC OPF with Contingencies	dccontopf
State Estimation	est
Reliability	rel
General Adequacy	genrel
Tie Open Point Opt.	topo
Motor Starting Calculation	motstart
Arc Flash Calculation	arcflash
Optimal Capacitor Placement	optcapo
Voltage Plan Optimization	mvplan
Backbone Calculation	backbone
Optimal RCS Placement	optrcs

GetActiveNetworkVariations

Returns all active variations for the 'Network Data' folder.

```
list Application.GetActiveNetworkVariations()
```

RETURNS

Returns currently active *IntScheme* objects. Set is empty in case of no scheme being currently active.

GetActiveProject

This function returns the currently active project.

```
DataObject Application.GetActiveProject()
```

RETURNS

Returns currently active *IntPrj* object or None in case of no project being currently active.

GetActiveScenario

Returns the currently active scenario. None is returned if there is no active scenario.

```
DataObject Application.GetActiveScenario()
```

RETURNS

Returns currently active *IntScenario* object or None in case of no scenario being currently active.

GetActiveScenarioScheduler

Returns currently active scenario scheduler.

```
DataObject Application.GetActiveScenarioScheduler()
```

RETURNS

Returns currently active *IntScensched* object or None in case of no scheduler being currently active.

GetActiveStages

Returns all active stages currently active for a given folder, e.g. 'Network Data' folder.

```
list Application.GetActiveStages([DataObject variedFolder])
```

ARGUMENTS

variedFolder (optional)
Folder for which all active stages will be returned; by default, the project folder 'Network Data' is taken.

RETURNS

Returns currently active *IntSstage* objects. Set is empty in case of no stages being currently active.

GetActiveStudyCase

Returns the active Study Case. None is returned if there is no active study case.

```
DataObject Application.GetActiveStudyCase()
```

RETURNS

The active study case (IntCase object) or None.

RETURNS

Returns currently active *IntCase* object or None in case of no study case being currently active.

GetAllUsers

Returns all known users, regardless of any Data Manager filters.

ARGUMENTS

forceReload

0	Default, returns the cached state if function was called before.
1	Forces the cache to be cleared, may impact performance.

RETURNS

Returns a container with all known users.

GetBorderCubicles

This function returns the border cubicles of the parent station of passed element topologically reachable from that element.
A cubicle (*StaCubic*) is considered to be a border cubicle if it resides inside the station

- and points to an element that sits outside the station
- or to a branch element that is connected to a terminal outside the station.

```
list Application.GetBorderCubicles(DataObject element)
```

ARGUMENTS

element Element from which the search for border cubicles starts

RETURNS

A set, containing border cubicles *StaCubic*. If the element does not reside in any substation or no border cubicles exist, the set is empty.

GetBrowserSelection

Returns all objects marked in the “on top” Data Manager (Browser, right side).

```
list Application.GetBrowserSelection()
```

RETURNS

Objects marked in the “on top” Data Manager (Browser, right side).

SEE ALSO

Application.GetCurrentSelection(), Application.GetDiagramSelection()

GetCalcRelevantObjects

Returns all currently calculation relevant objects, i.e. the objects which are used by the calculations.
The set of objects depends on active study case, active grid(s) and variation(s).

```
list Application.GetCalcRelevantObjects([str nameFilter,]
                                         [int includeOutOfService = 1,]
                                         [int topoElementsOnly = 0,]
                                         [int bAcSchemes = 0])
```

ARGUMENTS

nameFilter (optional)
(Class) name filter. Wildcards are supported. Multiple filters to be separated by comma ’,’. Must not contain a backslash ’\’.
If omitted, all objects are returned (corresponds to ’*.*’).
Examples for valid filter strings:

- ’ElmTerm’
- ’A*.ElmTerm’
- ’*.ElmLod,*.ElmSym’

includeOutOfService (optional)
Flag whether to include out of service objects. Default is 1 (=included).

topoElementsOnly (optional)
Flag to filter for topology relevant objects only. Default is 0 (=all objects).

bAcSchemes (optional)
Flag to include hidden objects in active schemes. Default is 0 (=not included).

RETURNS

The currently calculation relevant objects, according to the given arguments. The order of the set is undefined.

SEE ALSO

DataObject.IsCalcRelevant()

GetClassDescription

Returns a description for a PowerFactory class.

```
str Application.GetClassDescription(str name)
```

ARGUMENTS

name	Name of a *PowerFactory* class

RETURNS

Returns the description of a valid *PowerFactory* class, otherwise an empty string.

GetClassId

Returns a class identifier number.

Each class name corresponds to one unique number. The mapping of class name might be different for different build numbers of *PowerFactory*, but it is guaranteed that it will not changed while an Api instance exists. (Do not keep these numbers static, get them dynamically in your code using this method).

```
int Application.GetClassId(str className)
```

ARGUMENTS

className	Class name e.g. ”ElmTerm”.

0	Class name invalid.
$>$**0**	Class id of valid class name.

GetCurrentDiagram

This function offers access to the current diagram object (*IntGrfnet*).

```
DataObject Application.GetCurrentDiagram()
```

GetCurrentScript

Returns the current script (ComPython).

```
DataObject Application.GetCurrentScript()
```

RETURNS

The current script (ComPython) or None if started from external.

GetCurrentSelection

Returns all objects marked in the “on top” Data Manager (Browser, right side) or diagram.

```
list Application.GetCurrentSelection()
```

RETURNS

Objects marked in the “on top” Data Manager (Browser, right side) or diagram.

SEE ALSO

Application.GetBrowserSelection(), Application.GetDiagramSelection()

GetCurrentUser

Returns the PowerFactory user of current session.

```
DataObject Application.GetCurrentUser()
```

RETURNS

Returns an *IntUser* object, never None.

GetCurrentZoomScaleLevel

Returns the zoom or scale level of the currently active diagram. If the active diagram is geographic, then the scale level is returned, otherwise the zoom level is returned.

```
int Application.GetCurrentZoomScale()
```

RETURNS

Zoom or scale level of the active diagram as integer.

- For geographic diagrams the scale level is returned. E.g. returns 50000 if 1:50000 is in the zoom/ratio combo box
- For all other diagrams the zoom level is returned. E.g. returns 150 if 150

A value of -1 is returned in case of no open diagram.

GetDataFolder

This function returns the folder in which the network data for the given class are stored.

```
DataObject Application.GetDataFolder(str classname,
                                     [int iCreate])
```

ARGUMENTS

classname
Classname of the elements:

ElmBmu

ElmArea
ElmZone
ElmRoute
ElmOwner
ElmOperator
ElmFeeder
ElmCircuit
ElmBoundary
IntScales

iCreate(optional)

0	The folder is searched and returned if found. If the folder does not exist, None is returned.
1	The folder is created if it does not exist. The found or created folder is returned.

RETURNS

The network data folder, which is found or created.

SEE ALSO

DataObject.IsNetworkDataFolder()

GetDiagramSelection

Returns all objects marked in the “on top” diagram.

```
list Application.GetDiagramSelection()
```

RETURNS

Objects marked in the “on top” diagram.

SEE ALSO

Application.GetCurrentSelection(), Application.GetBrowserSelection()

GetFlowOrientation

This function returns the flow orientation setting of the active project.

```
int Application.GetFlowOrientation()
```

RETURNS

-1	No project is active
0	Flow orientation of active project is “MIXED MODE”
1	Flow orientation of active project is “LOAD ORIENTED”
2	Flow orientation of active project is “GENERATOR ORIENTED”

GetFromStudyCase

Returns the first found object of class “ClassName” from the currently active study case. The object is created when no object of the given name and/or class was found.

For commands the returned instance corresponds to the one that is used if opened via the main menue load-flow, short-circuit, transient simulation, etc.,

ARGUMENTS

ClassName
: Class name of the object (“Class”), optionally preceded by an object name without wildcards and a dot (“Name.Class”).

RETURNS

The found or created object.

GetGlobalLibrary

Returns the global library for object-types of class “ClassName”. ClassName may be omitted, in which case the complete global library folder is returned.

```
DataObject Application.GetGlobalLibrary([str ClassName])
```

ARGUMENTS

ClassName (optional)
: The classname of the objects for which the library folder is sought

RETURNS

The libary folder

SEE ALSO

Application.GetLocalLibrary()

GetGraphicsBoard

Returns the currently active Graphics Board.

```
DataObject Application.GetGraphicsBoard()
```

RETURNS

The graphics board object

GetInterfaceVersion

Returns the currently set interface version.

It holds the value set with SetInterfaceVersion() or the interface version from the current script (parameter ’interfaceVersion’) if the python script is executed from within PowerFactory.

Have a look into the PowerFactor user manual to get more informations about the interface version of a script.

```
int Application.GetInterfaceVersion()
```

RETURNS

The currently set interface version or 0 if PowerFactory is started from external and SetInterfaceVersion() is not called.

GetLanguage

Returns a string for the current program language setting.

```
str Application.GetLanguage()
```

RETURNS

en	English
de	German
es	Spanish

GetLocalLibrary

Returns the local library for object-types of class “ClassName”. ClassName may be omitted, in which case the complete local library folder is returned.

```
DataObject Application.GetLocalLibrary([str ClassName])
```

ARGUMENTS

ClassName (optional)
The classname of the objects for which the library folder is sought

RETURNS

The libary folder

SEE ALSO

Application.GetGlobalLibrary()

GetProjectFolder

Returns the project folder of a given type of active project. For each type (except ‘Generic’) there exist not more than one folder per type.

```
DataObject Application.GetProjectFolder(str type)
```

ARGUMENTS

type Type of the corresponding project folder. See IntPrjfolder.GetProjectFolderType() for a list of possible values.

RETURNS

An *IntPrjFolder* object. If no project is currently active or project folder of this type does not exist, None is returned.

GetRecordingStage

Returns the currently active recording scheme stage.

```
DataObject Application.GetRecordingStage ()
```

RETURNS

An *IntSstage* object; None if there is no recording stage.

GetSettings

Offers read-only access to some selected *PowerFactory* settings.

```
str Application.GetSettings(str key)
```

ARGUMENTS

key

Key	Return type	Description
usernm	string	Name of logged-in user (IntLogon:usernm)
ptdig	string	Fully qualified path of installation directory of *PowerFactory* (IntLogon:ptdig)
ptwrk	string	Fully qualified path of working directory of *PowerFactory* (IntLogon:ptwrk)
sessionid	integer	ID of current session
db_driver	string	Name of used database driver (IntLogon:db_driver)

RETURNS

Value of settings as string

GetSummaryGrid

Returns the summary grid in the currently active Study Case. The summary grid is the combination of all active grids in the study case.

```
DataObject Application.GetSummaryGrid()
```

RETURNS

A *ElmNet* object, or a 'None ' object when no grids are active

GetUserManager

Offers access to the user manager object (IntUserman) stored in the configuration folder.

```
DataObject Application.GetUserManager()
```

RETURNS

The user manager object

Hide

Hides the *PowerFactory* application window.

```
None Application.Hide()
```

SEE ALSO

Application.Show()

IsAttributeModeInternal

Returns whether the attribute values are accessed as internally stored.

```
int Application.IsAttributeModeInternal()
```

RETURNS

0	Attribute values accessed as displayed in *PowerFactory* (unit conversion applied).
1	Attribute values accessed as internally stored.

SEE ALSO

Application.SetAttributeModeInternal()

IsLdfValid

Checks to see if the last load-flow results are still valid and available.

```
int Application.IsLdfValid()
```

RETURNS

0 if no load-flow results are available

IsRmsValid

Checks to see if the last RMS simulation results are still valid and available.

```
int Application.IsRmsValid()
```

RETURNS

0 if no RMS simulation results are available

IsScenarioAttribute

Checks if a given attribute of a given class is recorded in scenario. It does not check whether a concrete instance is recorded at all. The check is just performed against the scenario configuration and is independent of a concrete scenario.

```
int Application.IsScenarioAttribute(str classname, str attributename)
```

ARGUMENTS

classname
Name of a *PowerFactory* class

attributename
Name of an attribute of given class

RETURNS

1	If attribute is scenario relevant according to current scenario configuration
0	If attribute is not scenario relevant

IsShcValid

Checks to see if the last short-circuit results are still valid and available.

```
int Application.IsShcValid()
```

RETURNS

0 if no short-circuit results are available

IsSimValid

Checks to see if the last simulation results are still valid and available.

```
int Application.IsSimValid()
```

RETURNS

0 if no simulation results are available

IsWriteCacheEnabled

Returns whether or not the cache method for optimizing performances is enabled.

```
int Application.IsWriteCacheEnabled()
```

RETURNS

0	Write cache is disabled.
1	Write cache is enabled.

SEE ALSO

Application.SetWriteCacheEnabled(), Application.WriteChangesToDb()

LoadProfile

Activates a profile for current user. This corresponds to the select profile action via main menue “TOOLS-Profiles”.

```
int Application.LoadProfile(str profileName)
```

ARGUMENTS

profileName
Name of profile to be loaded.

RETURNS

0	On error, e.g. profile with given name not found.
1	On success.

SEE ALSO

Application.ReloadProfile()

MarkInGraphics

Marks all objects in the diagram in which the elements are found by hatch crossing them.

```
None Application.MarkInGraphics(list objects,
                                [int searchOpenedDiagramsOnly = 0])
```

ARGUMENTS

objects Objects to be marked.

searchOpenedDiagramsOnly (optional)
Search can be restricted to currently shown diagrams on the desktop, instead of all diagrams.

0 Searching all diagrams, not only the ones which are currently shown on the desktop. If there is more than one occurrence the user will be prompted which diagrams shall be opened.

1 Only search in currently opened diagrams and open the first diagram in which the elements were found (default).

OutputFlexibleData

Outputs the Flexible Data of the given objects to the output window.

Has identical functionality to that implemented in the Object Filter dialogue, whereby the user can right-click on a single row or multiple rows in a Flexible Data page and select Output . . . Flexible Data. The OutputFlexibleData() function assumes that the user has already defined a Flexible Data page for the objects in the set. Upon execution of this function, all Flexible Data defined for the objects in the set is output to the *PowerFactory* output window in a tabular format.

```
None Application.OutputFlexibleData(list objects,
                                    [str flexibleDataPage = ''])
```

ARGUMENTS

objects Objects to output the Flexible Data for.

flexibleDataPage (optional)
Name of the Flexible Data page to be outputed. If multiple Flexible Data pages are defined and no or an empty string is given then a dialog to select a Flexible Data page is shown.

PostCommand

Adds a command to the command pipe of the “input window”. The posted commands will be executed after the currently running script has finished.

```
None Application.PostCommand(str command)
```

ARGUMENTS

command
The command string.

Rebuild

Rebuilds the currently visible single line diagram.

```
None Application.Rebuild([int iMode])
```

ARGUMENTS

iMode (optional)

0	Draws graphic objects only
1	(default) Reads graphic objects (IntGrf) from database and draws
2	Reads graphic objects (IntGrf) from database, re-calculates intersections and draws

ReloadProfile

Reloads currently selected user profile. (See main menue “TOOLS-Profiles”)

```
None Application.ReloadProfile()
```

SEE ALSO

Application.LoadProfile()

ResetCalculation

Resets all calculations and deletes all calculation results.

Results that have been written to result objects (for display in graphs) will not be destroyed. All results that are visible in the single line diagrams, however, will be destroyed.

```
None Application.ResetCalculation()
```

SEE ALSO

Application.IsAutomaticCalculationResetEnabled(), Application.SetAutomaticCalculationResetEnabled()

ResGetData

This function is deprecated. Please use ElmRes.GetValue() or IntComtrade.GetValue() instead.

```
[int error,
float d   ] Application.ResGetData(DataObject resultObject,
                                   int iX,
                                   [int col])
```

ResGetDescription

This function is deprecated. Please use ElmRes.GetDescription() or IntComtrade.GetDescription() instead.

```
str Application.ResGetDescription(DataObject resultObject,
                                  int col,
                                  [int ishort])
```

plFunctionResFirstValidObject

ResGetFirstValidObject

This function is deprecated. Please use ElmRes.GetFirstValidObject() instead.

```
int Application.ResGetFirstValidObject(DataObject resultFile,
                                       int row,
                                       [str classNames]
                                       [str variableName,]
                                       [float limit,]
                                       [int limitOperator,]
                                       [float limit2,]
                                       [int limitOperator2])
int Application.ResGetFirstValidObject([DataObject resultFile,]
                                       [int row,]
                                       [list objects])
```

ResGetFirstValidObjectVariable

This function is deprecated. Please use ElmRes.GetFirstValidObjectVariable() instead.

```
int Application.ResGetFirstValidObjectVariable(DataObject resultFile,
                                               [str variableNames])
```

ResGetFirstValidVariable

This function is deprecated. Please use ElmRes.GetFirstValidVariable() instead.

```
int Application.ResGetFirstValidVariable(DataObject resultFile,
                                         int row,
                                         [str variableNames])
```

ResGetIndex

This function is deprecated. Please use ElmRes.FindColumn() or IntComtrade.FindColumn() instead.

```
int Application.ResGetIndex(DataObject resultFile,
                            DataObject obj,
                            [str varName])
int Application.ResGetIndex(DataObject resultFile,
                            DataObject obj,
                            [int colIndex])
int Application.ResGetIndex(DataObject resultFile,
                            [str varName,]
                            [int colIndex])
```

ResGetMax

This function is deprecated. Please use ElmRes.FindMaxInColumn() or IntComtrade.FindMaxInColumn() instead.

```
[int row,
float value] Application.ResGetMax(DataObject resultFile,
                                   int col)
```

ResGetMin

This function is deprecated. Please use ElmRes.FindMinInColumn() or IntComtrade.FindMinInColumn() instead.

```
[int row,
float value] Application.ResGetMin(DataObject resultFile,
                                   int col)
```

ResGetNextValidObject

This function is deprecated. Please use ElmRes.GetNextValidObject() instead.

```
int Application.ResGetNextValidObject(DataObject resultFile,
                                      [str classNames]
                                      [str variableName,]
                                      [float limit,]
                                      [int limitOperator,]
                                      [float limit2,]
                                      [int limitOperator2])
int Application.ResGetNextValidObject(DataObject resultFile,
                                      list objects)
```

ResGetNextValidObjectVariable

This function is deprecated. Please use ElmRes.GetNextValidObjectVariable() instead.

```
int Application.ResGetNextValidObjectVariable(DataObject resultFile,
                                              [str variableNames])
```

ResGetNextValidVariable

This function is deprecated. Please use ElmRes.GetNextValidVariable() instead.

```
int Application.ResGetNextValidVariable(DataObject resultFile,
                                        [str variableNames])
```

ResGetObject

This function is deprecated. Please use ElmRes.GetObject() instead.

```
DataObject Application.ResGetObj(DataObject resultObject,
                                 int col)
```

ResGetUnit

This function is deprecated. Please use ElmRes.GetUnit() or IntComtrade.GetUnit() instead.

```
str Application.ResGetUnit(DataObject resultObject,
                           int col)
```

ResGetValueCount

This function is deprecated. Please use ElmRes.GetNumberOfRows() or IntComtrade.GetNumberOfRows() instead.

```
int Application.ResGetValueCount(DataObject resultObject,
                                 [int col])
```

ResGetVariable

This function is deprecated. Please use ElmRes.GetVariable() or IntComtrade.GetVariable() instead.

```
str Application.ResGetVariable(DataObject resultObject,
                               int col)
```

ResGetVariableCount

This function is deprecated. Please use ElmRes.GetNumberOfColumns() or IntComtrade.GetNumberOfColumns() instead.

```
int Application.ResGetVariableCount(DataObject resultObject)
```

ResLoadData

This function is deprecated. Please use ElmRes.Load() or IntComtrade.Load() instead.

```
None Application.ResLoadData(DataObject resultObject)
```

ResReleaseData

This function is deprecated. Please use ElmRes.Release() or IntComtrade.Release() instead.

```
None Application.ResReleaseData(DataObject resultObject)
```

ResSortToVariable

This function is deprecated. Please use ElmRes.SortAccordingToColumn() or IntComtrade.SortAccordingToColumn() instead.

```
int Application.ResSortToVariable(DataObject resultObject,
                                  int col)
```

SaveAsScenario

Saves the operational data or relevant network elements as a new scenario.

```
DataObject Application.SaveAsScenario(str pName,
                                      int iSetActive)
```

ARGUMENTS

pName Name of the new scenario.

iSetActive

1	Activate the new scenario afterwards.
0	Do not activate the new scenario.

RETURNS

Returns newly created *IntScenario* object. None is returned in case of creation of a new scenario was not allowed (e.g. no active project).

SearchObjectByForeignKey

Searches for an object by foreign key within an active project.

```
DataObject Application.SearchObjectByForeignKey(str foreignKey)
```

ARGUMENTS

foreignKey
Foreign key

RETURNS

Object if found, otherwise None.

SelectToolbox

Sets tool box to be displayed at a switchable tool box group.

```
int Application.SelectToolbox(int toolbar,
                              str groupName,
                              str toolboxName)
```

ARGUMENTS

toolbar	**1**	Main tool bar
	2	Drawing tool bar (SGL)

groupName
Name of tool box group.

toolboxName
Name of tool box to be selected.

RETURNS

0	On error, e.g. no matching tool box found.
1	On success.

SetAttributeModeInternal

Changes the way how attribute values are accessed.

```
None Application.SetAttributeModeInternal(int internalMode)
```

ARGUMENTS

internalMode	**0**	Access attribute values as displayed in *PowerFactory* (unit conversion applied).
	1	Access attribute values as internally stored.

SEE ALSO

Application.IsAttributeModeInternal()

SetInterfaceVersion

Sets the current interface version. Only values which can be set to the python script parameter 'interfaceVersion' are allowed. Setting the interface version does not affect the parameter 'interfaceVersion' of the current script.

Have a look into the PowerFactor user manual to get more informations about the interface version of a script.

```
int Application.SetInterfaceVersion(int version)
```

ARGUMENTS

version	interface version to be set

RETURNS

0 if the version is successfully set, otherwise 1.

SetShowAllUsers

Enables or disables the filtering of all available users in data manager. All users are only visualised in data manager when enabled.

```
int Application.SetShowAllUsers(int enabled)
```

ARGUMENTS

enabled

0	Disabled, only Demo, Public Area Users and current user are shown
1	Enabled, all available users are listed

RETURNS

Returns previous setting.

1	If enabled before.
0	If disabled before.

SetWriteCacheEnabled

This function intends to optimize performances. In order to modify objects in *PowerFactory*, those must be set in a special edit mode before any value can be changed. Switching back and forth between edit mode and stored mode is time consuming; enabling the write cache flag will set objects in edit mode and they will not be switched back until WriteChangeToDb is called.

```
None Application.SetWriteCacheEnabled(int enabled)
```

ARGUMENTS

enabled	**0**	Disables the write cache.
	1	Enables the write cache.

SEE ALSO

Application.IsWriteCacheEnabled(), Application.WriteChangesToDb()

Show

Shows the *PowerFactory* application window (only possible with a full license, not supported for engine licenses).

```
None Application.Show()
```

SEE ALSO

Application.Hide()

StatFileGetXrange

Gets the x-range for the statistic result file.

```
[int error,
double min,
double max] Application.StatFileGetXrange()
```

ARGUMENTS

min (out)	First point in time considered in statistics.
max (out)	Last point in time considered in statistics.

RETURNS

0	If time range of statistic result file was found.
1	On errors (There is no statistic result file).

StatFileResetXrange

Reset the user defined x-range of the statistic result file. The complete x-range will be considered in the statistic results after calling this function.

```
None Application.StatFileResetXrange()
```

StatFileSetXrange

Sets the user defined x-range of the statistic result file. The statistic results consider only the given time range.

```
None Application.StatFileSetXrange(float min,
                                   float max)
```

ARGUMENTS

min	First point in time to be considered in statistics.
max	Last point in time to be considered in statistics.

WriteChangesToDb

This function combined with Application.SetWriteCacheEnabled() is meant to optimize performances. If the write cache flag is enabled all objects remain in edit mode until WriteChangesToDb is called and all the modifications made to the objects are saved into the database.

```
None Application.WriteChangesToDb()
```

SEE ALSO

Application.SetWriteCacheEnabled(), Application.IsWriteCacheEnabled()

3.1 File System

Overview

GetInstallationDirectory
GetTemporaryDirectory
GetWorkspaceDirectory

GetInstallationDirectory

Returns the installation directory of *PowerFactory* .

```
str Application.GetInstallationDirectory()
```

RETURNS

Full path to installation directory of current *PowerFactory* .

DEPRECATED NAMES

GetInstallDir

SEE ALSO

Application.GetTemporaryDirectory(), Application.GetWorkspaceDirectory()

GetTemporaryDirectory

Returns the temporary directory of used by *PowerFactory* .

```
str Application.GetTemporaryDirectory()
```

RETURNS

Full path to a directory where temporary data can be stored. This directory is also used by *PowerFactory* to store temporary data.

DEPRECATED NAMES

GetTempDir

SEE ALSO

Application.GetWorkspaceDirectory(), Application.GetInstallationDirectory()

GetWorkspaceDirectory

Returns the workspace directory of *PowerFactory*.

```
str Application.GetWorkspaceDirectory()
```

RETURNS

Full path to the directory where currently used workspace is stored.

DEPRECATED NAMES

GetWorkingDir

SEE ALSO

Application.GetTemporaryDirectory(), Application.GetInstallationDirectory()

3.2 Date/Time

Overview

GetStudyTimeObject

GetStudyTimeObject

Returns the date and time object (SetTime) from the study case. This is the object being used by the characteristics, scenarios,...

RETURNS

SetTime or NULL.

3.3 Dialogue Boxes

Overview

CloseTableReports

Closes all open table reports.

```
None Application.CloseTableReports()
```

ShowModalBrowser

Opens a modal browser window and lists all given objects.

```
None Application.ShowModalBrowser(list objects,
                                  [int detailMode = 0,]
                                  [str title = '',]
                                  [str page = ''])
```

ARGUMENTS

objects Objects to be listed. The listing is in detailed mode, if only one kind of objects (e.g. only ElmTerm) is contained.

detailMode (optional)

0	Show browser in normal mode (default).
1	Show browser in detail mode.

title (optional)
String for user defined window title. The default window title is shown when no or an empty string is given.

page (optional)
Name of page to be shown in browser e.g. 'Flexible Data' (only in detailed mode). The default page is shown when no or an empty string is given.

ShowModalSelectBrowser

Opens a modal browser window and lists all given objects. The user can make a selection from the list.

```
list Application.ShowModalSelectBrowser(list objects,
                                        [str title,]
                                        [str classFilter,]
                                        [str page = ''])
```

Arguments

objects Objects to be listed. The listing is in detailed mode, if only one kind of objects (e.g. only ElmTerm) is contained.

title (optional)
String for user defined window title. The default window title is shown when no or an empty string is given.

classFilter (optional)
Class name filter. If set, only objects matching that filter will be listed in the dialog e.g. 'Elm*', 'ElmTr?' or 'ElmTr2,ElmTr3'.

page (optional)
Name of page to be shown in browser e.g. 'Flexible Data' (only in detailed mode). The default page is shown when no or an empty string is given.

Returns

Set of selected objects. The set is empty if “cancel” is pressed.

ShowModelessBrowser

Opens a modeless browser window and lists all given objects.

```
None Application.ShowModelessBrowser(list objects,
                                     [int detailMode = 0,]
                                     [str title = '',]
                                     [str page = ''])
```

Arguments

objects Objects to be listed. The listing is in detailed mode, if only one kind of objects (e.g. only ElmTerm) is contained.

detailMode (optional)

0	Show browser in normal mode (default).
1	Show browser in detail mode.

title (optional)
String for user defined window title. The default window title is shown when no or an empty string is given.

page (optional)
Name of page to be shown in browser e.g. 'Flexible Data' (only in detailed mode). The default page is shown when no or an empty string is given.

UpdateTableReports

Updates all open table reports.

```
None Application.UpdateTableReports()
```

3.4 Environment

Overview

EchoOff

Freezes (de-activates) the user-interface. For each EchoOff(), an EchoOn() should be called. An EchoOn() is automatically executed at the end of the execution of a ComDpl or ComPython. This could be changed with Application.SetFinalEchoOnEnabled().

```
None Application.EchoOff()
```

SEE ALSO

Application.EchoOn(), Application.IsFinalEchoOnEnabled(), Application.SetFinalEchoOnEnabled()

EchoOn

Re-activates the user interface. For more informations see Application.EchoOff().

```
None Application.EchoOn()
```

SEE ALSO

Application.EchoOff(), Application.IsFinalEchoOnEnabled(), Application.SetFinalEchoOnEnabled()

IsAutomaticCalculationResetEnabled

Returns whether the automatic calculation reset while setting attributes is enabled. See Application.SetAutomaticCalculationResetEnabled() for more informations.

```
int Application.IsAutomaticCalculationResetEnabled()
```

SEE ALSO

Application.SetAutomaticCalculationResetEnabled(), Application.ResetCalculation()

IsFinalEchoOnEnabled

Returns whether the automatic Application.EchoOn() at the end of each *ComDpl* or *ComPython* is enabled.

```
int Application.IsFinalEchoOnEnabled();
```

RETURNS

1	Final Application.EchoOn() is enabled.
0	Final Application.EchoOn() is disabled.

SEE ALSO

Application.SetFinalEchoOnEnabled(), Application.EchoOn(), Application.EchoOff()

SetAutomaticCalculationResetEnabled

Enables or disables the automatic calculation reset while setting attributes.

In Python/API the automatic calculation reset is by default enabled. Thus changing an object attribute could lead to a calculation reset, e.g. changing the scaling factor of a load, but do not have to, e.g. renaming an object.

Even if the automatic calculation reset is disabled, changing the ”outserv” attribute of an arbitrary network element or the ”on_off” attribute of a switch device resets automatically the current calculation.

When the calculation is reset the load-flow will be calculated with a flat start. Thus switching the automatic calculation reset off can be helpful e.g. when calculating a load-flow without a flat start. On the other side it could lead to wrong results e.g. doing short-circuit calculations after changing the short-circuit-location of a branch without calling Application.ResetCalculation().

```
None Application.SetAutomaticCalculationResetEnabled(int enabled)
```

SEE ALSO

Application.IsAutomaticCalculationResetEnabled(), Application.ResetCalculation()

SetFinalEchoOnEnabled

Enables or disables the automatic Application.EchoOn() at the end of each *ComDpl* or *ComPython*.

```
None Application.SetFinalEchoOnEnabled(int enabled);
```

ARGUMENTS

enabled

1	Enables the final Application.EchoOn().
0	Disables the final Application.EchoOn().

SEE ALSO

Application.IsFinalEchoOnEnabled(), Application.EchoOn(), Application.EchoOff()

SetGraphicUpdate

Enables or disables the updates of the single line graphics.

```
None Application.SetGraphicUpdate(int enabled)
```

ARGUMENTS

enabled

0	disabled (graphic will not be updated automatically)
1	enabled

SetGuiUpdateEnabled

Enables or disables updates of the graphical user interface (e.g. application window) while the script is running.

This can be useful to get maximum execution performance. However, the user interface might look frozen and becomes not responsive. The updates will automatically be re-enabled after termination of the script. In case of sub-scripts, the restore is done at termination of main script.

```
int Application.SetGuiUpdateEnabled(int enabled)
```

ARGUMENTS

enabled

0	Disables GUI updates.
1	Enables GUI updates.

RETURNS

Previous state before the function was called

0	GUI updates were disabled before.
1	GUI updates were enabled before.

DEPRECATED NAMES

SetRescheduleFlag

SEE ALSO

Application.SetGraphicUpdate()

SetUserBreakEnabled

Enables or disables the “Break” button in main tool bar. After script execution it is disabled automatically.

```
None Application.SetUserBreakEnabled(int enabled)
```

ARGUMENTS

enabled

0	Disables “Break” button.
1	Enable “Break” button.

DEPRECATED NAMES

SetEnableUserBreak

3.5 Mathematics

Overview

GetRandomNumber

This method is marked as deprecated since PowerFactory 2017. Please use Application.RndUnifReal() instead.

Draws a uniformly distributed random number. Uses the 'global random number generator'. If x1 and x2 are omitted, the distribution will be uniform in the intervall [0, 1]. If only x1 is given, the distribution is uniform in [0, x1] and with both x1 and x2, the distribution is uniform in [x1, x2].

```
float Application.GetRandomNumber([float x1,]
                                  [float x2])
```

ARGUMENTS

x1 (optional)
x2 not given: maximum; x1 and x2 given: minimum

x2 (optional)
maximum

RETURNS

A uniformly distributed random number

GetRandomNumberEx

This method is marked as deprecated since PowerFactory 2017. Please use Application.RndUnifReal(), Application.RndNormal() or Application.RndWeibull() instead.

Draws a random number according to a specific probability distribution. Uses the 'global random number generator'.

```
float Application.GetRandomNumberEx(int distribution,
                                    [float p1,]
                                    [float p2])
```

ARGUMENTS

distribution

0	uniform distribution

1	normal distribution
2	weibull distribution
else	returns 0.0

p1 (optional)

distribution = 0 (uniform), argument p2 is also given: min

distribution = 0 (uniform), argument p2 is not given: max (min is assumed to be 0).

distribution = 1 (normal) : mean

distribution = 2 (weibull) : scale

p2 (optional)

distribution = 0 (uniform) : max

distribution = 1 (normal) : stddev

distribution = 2 (weibull) : weibull

RETURNS

double	Newly drawn random number from the specified distribution.
0.0	On failure e.g. non-supported mode.

InvertMatrix

This routine calculates the inverse matrix by the Gauss-Jordan method. It uses scaled partial pivoting preceeded by column equilibration of the input matrix. The routine can be called in two different versions:

- **Real Inversion:** Only one matrix, $realPart$, is provided as an input to the function. Then, $realPart$ is inverted and the result, $realPart^{-1}$, is stored into the input matrix $realPart$ on success.
- **Complex Inversion:** Two matrices, $realPart$ and $imaginaryPart$, are provided as inputs to this function. Then, a complex matrix C is formed, with entries

  $$C(i,j) = A(i,j) + j \cdot imaginaryPart(i,j).$$

  The complex matrix C is inverted and, on success, the resulting real part of C^{-1} is written to $realPart$ whereas the resulting imaginary part of C^{-1} is written to $imaginaryPart$. Please note that $realPart$ and $imaginaryPart$ must have the same dimensions.

```
int Application.InvertMatrix(DataObject realPart,
                             [DataObject imaginaryPart])
```

ARGUMENTS

realPart If imaginaryPart is not set, realpart is the matrix to invert on input. In case of success, it will be overwritten by the inverted input matrix. If imaginaryPart is set, it holds the real part of the complex matrix to invert on input and is overwritten by the real part of the inverted complex matrix on output.

imaginaryPart
If this is set, it should hold the imaginary part of the matrix to invert on input and is overwritten by the imaginary part of the inverted matrix on output.

RETURNS

1	Matrix inversion failed. The provided input matrix is singular.
0	Matrix inversion was successfull. Resulting inverted matrix returned in input matrix/matrices.

RndExp

Returns a random number distributed according to exponential distribution with given rate. See the example given in the DPL description of Application.RndSetup().

```
double RndExp(double rate, [int rngNum])
```

ARGUMENTS

rate Rate of exponetial distribution.

rngNum (optional)
Number of the random number generator.

0 (default)	'Global random number generator'.
1, 2, ...	Other random number generators accessable via this number.

RETURNS

double Random number

RndGetMethod

Returns the used method of a random number generator. See the example given in the DPL description of Application.RndSetup().

```
str RndGetMethod([int rngNum])
```

ARGUMENTS

rngNum (optional)
Number of the random number generator of which the method type is returned.

0 (default)	'Global random number generator'.
1, 2, ...	Other random number generators accessable via this number.

RETURNS

string Name of the used method

RndGetSeed

Returns the used seed of a random number generator. See the example given in the DPL description of Application.RndSetup().

```
int RndGetSeed([int rngNum])
```

ARGUMENTS

rngNum (optional)
Number of the random number generator.

0 (default)	'Global random number generator'.
1, 2, ...	Other random number generators accessable via this number.

RETURNS

int	Used seed

RndNormal

Returns a random number distributed according to normal distribution with given mean and standard deviation. See the example given in the DPL description of Application.RndSetup().

```
double RndNormal(double mean, double stddev, [int rngNum])
```

ARGUMENTS

mean Mean of normal distribution.

stddev Standard deviation of normal distribution.

rngNum (optional)
Number of the random number generator.

0 (default) 'Global random number generator'.

1, 2, ... Other random number generators accessable via this number.

RETURNS

double	Random number

RndSetup

Initializes a random number generator. Allows to choose:

1. Whether to seed automatically or not.
2. The seed, if not automatically seeded.
3. The type or random number generator.
4. The random number generator to use.

Supported types of random number generators:

1. Mersenne Twister,
2. Linear Congruential,
3. Additive Lagged Fibonacci.

Internally a vector of random number generators is used. These can be accessed via the number passed as last argument. Number 0 corresponds to the 'global random number generator', updated also in ComInc and ComGenreIinc. Numbers $1,2,...$ will access different random number generators, which can be setup individually.

```
None RndSetup(int seedAutomatic, [int seed], [int rngType], [int rngNum])
```

ARGUMENTS

seedAutomatic
Seed the random number generator automatically

0	Do not seed automatically.
1	Seed automatically.

seed (optional)
Seed for the random number generator. (default: 0) Note, that for the Additive Lagged Fibonacci generator, only the seeds 0,...,9 are supported.

rngType (optional)
Type of random number generator

0	Mersenne Twister (recommended) (default).
1	Linear Congruential.
2	Additive Lagged Fibonacci.

rngNum (optional)
Number of random number generator to be used

0 (default)	'Global random number generator'.
1, 2, ...	Other random number generators accessable via this number.

RndUnifInt

Returns a random number distributed according to uniform distribution on the set of numbers $\{min, \dots, max\}$. See the example given in the DPL description of Application.RndSetup().

```
int RndUnifInt(int min, int max, [int rngNum])
```

ARGUMENTS

min Smallest possible number

max Largest possible number

rngNum (optional)
Number of the random number generator.

0 (default)	'Global random number generator'.
1, 2, ...	Other random number generators accessable via this number.

RETURNS

int Random number

RndUnifReal

Returns a random number distributed according to uniform distribution on the intervall $[min,max]$. See the example given in the DPL description of Application.RndSetup().

```
double RndUnifReal(double min, double max, [int rngNum])
```

Arguments

min	Lower endpoint of interval $[min, max]$
max	Upper endpoint of interval $[min, max]$
rngNum (optional)	Number of the random number generator.

0 (default) 'Global random number generator'.

1, 2, ... Other random number generators accessable via this number.

Returns

double Random number

RndWeibull

Returns a random number distributed according to Weibull distribution with given shape and scale parameters. See the example given in the DPL description of Application.RndSetup().

```
double RndWeibull(double shape, double scale, [int rngNum])
```

Arguments

shape	Shape parameter of Weibull distribution.
scale	Scale parameter of Weibull distribution.
rngNum (optional)	Number of the random number generator.

0 (default) 'Global random number generator'.

1, 2, ... Other random number generators accessable via this number.

Returns

double Random number

SetRandomSeed

This method is marked as deprecated since PowerFactory 2017. Please use Application.RndSetup() instead.

Initializes the 'global random number generator' as Additive Lagged Fibonacci random number generator. Sets the seed for the random number generator. One out of 10 predefined initialization seeds can be selected.

```
None Application.SetRandomSeed(int seed)
```

Arguments

seed	seed 0..9

3.6 Output Window

Overview

ClearOutputWindow
PrintError
PrintInfo
PrintPlain
PrintWarn
SetOutputWindowState

ClearOutputWindow

Clears the output window.

```
None Application.ClearOutputWindow()
```

PrintError

Prints a message as error into the *PowerFactory* Output Window.

```
None Application.PrintError(str message)
```

ARGUMENTS

message	Message to print.

PrintInfo

Prints a message as information into the *PowerFactory* Output Window.

```
None Application.PrintInfo(str message)
```

ARGUMENTS

message	Message to print.

PrintPlain

Prints a message as normal text into the *PowerFactory* Output Window.

```
None Application.PrintPlain(str message)
```

ARGUMENTS

message	Message to print.

PrintWarn

Prints a message as warning into the *PowerFactory* Output Window.

```
None Application.PrintWarn(str message)
```

ARGUMENTS

message Message to print.

SetOutputWindowState

Changes the display state of the output window.

```
None Application.SetOutputWindowState(int newState)
```

ARGUMENTS

newState

0	Minimized output window.
1	Maximized output window.
-1	Restore previous state.

4 Object Methods

4.1 General Methods

Overview

AddCopy

Copies a single object or a set of objects to the target object. “Fold.AddCopy(aObj)” copies object 'aObj' into the target object 'Fold', “Fold.AddCopy(aSet)” copies all objects in 'aSet' to “Fold”.

“Fold.AddCopy(aObj, nm1, nm2, ...)” will copy aObj and rename it to the result of the concatenation of 'nm1', 'nm2', etc. The object will not be renamed if it was an IntPrj.

The target object must be able to receive a copy of the objects. The function “Fold.AddCopy(aObj,...)” returns the copy of “aObj”, “Fold.AddCopy(aSet)” returns “Fold”, when the copy operation was successful. A None object is returned otherwise.

Copying a set of objects will respect all internal references between those objects. Copying a set of lines and their types, for example, will result in a set of copied lines and line types, where the copied lines will use the copied line types.

```
DataObject DataObject.AddCopy(DataObject obj,
                              [int or str partOfName1,]
                              [...])
DataObject DataObject.AddCopy(list objects)
```

Arguments

obj The object to copy.

partOfName1 (optional)
The first part of the new name.

objects The set of objects to copy.

Returns

Returns the copy that has been created, unless the copied object was a set of objects.

ContainsNonAsciiCharacters

Checks whether an object contains texts attributes with non-ASCII characters.

```
int DataObject.ContainsNonAsciiCharacters()
```

Returns

Returns 1 if the object contains at least one non-ASCII characters. Otherwise 0.

CopyData

Copies all parameters except for loc_name and containers from one object to another.

```
None DataObject.CopyData(DataObject source)
```

Arguments

source	Object from which parameters are to be copied

Returns

0	ok
1	error

CreateObject

Creates a new object of given class and name in the target object. The object name will be concatenated by the given object name parts. The target object must be able to store an object of the given class in its content otherwise the currently running script will stop with an error.

```
DataObject DataObject.CreateObject(str className,
                                    [int|str objectNamePart0,]
                                    [...]
                                    )
```

Arguments

className
The class name of the object to create.

objectNameParts (optional)
Parts of the name of the object to create (without classname) which will be concatenated to the object name.

Returns

object	Newly created object.
None	When no object was created.

Delete

Deletes the object from the database. The object is not destroyed but moved to the recycle bin.

```
int DataObject.Delete()
```

RETURNS

0	Object successfully deleted.
$\neq 0$	Deletion failed e.g. not allowed.

SEE ALSO

DataObject.CreateObject()

Energize

Performs an “energize” action on the network element. This corresponds to removing earthings from current region (if any) followed by a “switch on” action on the element.
The action is identical to that in the context menue.

```
[int error,
list changedSwitches] DataObject.Energize([int resetRA])
```

ARGUMENTS

changedSwitches (optional, out)
All switches whose switching state was changed by the action are filled into this set.

resetRA (optional)
Determines whether an active running arrangement that would prevent switching action should be deactivated or not.

1	All running arrangements that cause blocking of relevant switches are applied and reset automatically before the switching is performed.
0	(default) Active running arrangements are not reset. Blocked switches will cause the switching action to fail.

RETURNS

Information about the success of the action:

0	Action was successful.
1	Action failed.

SEE ALSO

DataObject.SwitchOn(), DataObject.SwitchOff(), DataObject.Isolate()

GetAttribute

Returns the value of an attribute.

```
int|float|str|DataObject|list DataObject.GetAttribute(str name)
```

ARGUMENTS

name Name of an attribute.

Returns

Value of an attribute name in its current unit (like in the edit dialog seen). An exception is thrown for invalid attribute names.

See also

DataObject.SetAttribute(), DataObject.GetAttributeType()

GetAttributeDescription

Returns the description of an attribute.

```
str DataObject.GetAttributeDescription(str name,
                                        int short = 0)
```

Arguments

name	Name of an attribute.	
short	**0**	Return long attribute description (default).
	1	Return short attribute description.

Returns

""	For an invalid attribute name.
str	Long or short attribute description.

See also

DataObject.GetAttributeType(), DataObject.GetAttributeUnit()

GetAttributeLength

Returns the length of a vector or matrix attribute. The length of a matrix attribute is the number of rows.

```
int DataObject.GetAttributeLength(str name)
```

Arguments

name	Name of an attribute.

Returns

> 0	Length of a valid vector or matrix attribute.
0	All other valid attributes.
-1	For invalid attribute names.

See also

DataObject.GetAttributeShape(), DataObject.SetAttributeLength(), DataObject.GetAttributeType()

GetAttributeShape

Returns the shape of an attribute. The shape is a list of the form [number of rows, number of colums].

```
[int rows,
int columns ] DataObject.GetAttributeShape(str name)
```

ARGUMENTS

name	Name of an attribute.

RETURNS

[$\geq 0, \geq 0$]	Shape of a valid vector or matrix attribute.
[$0, 0$]	All other valid attributes.
[$-1, 0$]	For invalid attribute names.

SEE ALSO

DataObject.GetAttributeLength(), DataObject.SetAttributeShape(), DataObject.GetAttributeType()

GetAttributeType

Returns the type of an attribute.

The following attribute types exist:

AttributeType.INVALID	Attribute does not exist.
AttributeType.INTEGER	Integer attribute.
AttributeType.INTEGER_VEC	Integer vector attribute.
AttributeType.DOUBLE	Double attribute.
AttributeType.DOUBLE_VEC	Double vector attribute.
AttributeType.DOUBLE_MAT	Double matrix attribute.
AttributeType.OBJECT	Data object attribute.
AttributeType.OBJECT_VEC	Data object vector attribute.
AttributeType.STRING	Integer attribute.
AttributeType.STRING_VEC	Integer vector attribute.
AttributeType.INTEGER64	64-bit integer attribute.
AttributeType.INTEGER64_VEC	64-bit integer vector attribute.

```
AttributeType DataObject.GetAttributeType(str name)
```

ARGUMENTS

name	Name of an attribute.

RETURNS

The type of an attribute or *AttributeType.INVALID* for an invalid attribute name.

SEE ALSO

DataObject.GetAttribute(), DataObject.SetAttribute()

GetAttributeUnit

Retruns the unit of an attribute e.g. km, MW....

```
str DataObject.GetAttributeUnit(str name)
```

ARGUMENTS

name	Name of an attribute.

RETURNS

""	For invalid attribute names.
str	Attribute unit.

SEE ALSO

DataObject.GetAttributeType(), DataObject.GetAttributeDescription()

GetChildren

This function returns the objects that are stored within the object the function was called on. In contrast to DataObject.GetContents() this function gives access to objects that are currently hidden due to scheme management.

```
list DataObject.GetChildren(int hiddenMode,
                            [str filter,]
                            [int subfolders])
```

ARGUMENTS

hiddenMode
Determines how hidden objects are handled.

0	Hidden objects are ignored. Only nonhidden objects are returned.
1	Hidden objects and nonhidden objects are returned.
2	Only hidden objects are returned. Nonhidden objects are ignored.

filter (optional)
Name filter, possibly containing '*' and '?' characters.

subfolder (optional)
Determines if children of subfolders are returned.

0	Only direct children are returned, children of subfolders are ignored (Default).
1	Also children of subfolders are returned.

RETURNS

Objects that are stored in the called object.

SEE ALSO

DataObject.GetContents()

GetClassName

Returns the class name of the object.

```
str DataObject.GetClassName()
```

RETURNS

The class name of the object.

GetCombinedProjectSource

For an object in a combined project return the intermediate folder where the object is contained, indicating the original source project.

```
DataObject DataObject.GetCombinedProjectSource()
```

RETURNS

The intermediate folder for that object or nothing when not applicable.

GetConnectedElements

Returns the set of connected elements. Only electrically connected elements are returned when the conditions of all switches are regarded. Possible connections will also be returned when rBrk and/or rDis is zero, in the case of open breakers and/or disconnectors. The default values are (0,0,0).

```
list DataObject.GetConnectedElements([int rBrk],
                                     [int rDis],
                                     [int rOut])
```

ARGUMENTS

rBrk (optional)
  if 1, regards position of breakers

rDis (optional)
  if 1, regards position of disconnectors

rOut (optional)
  if 1, regards in-service or out-of-service status

RETURNS

The set of connected elements.

GetConnectionCount

Returns the number of electrical connections.

```
int DataObject.GetConnectionCount()
```

RETURNS

The number of electrical connections.

GetContents

Returns the objects that are stored in the object and whose name matches the argument name. No object is returned if the object's container is empty, or if the object is not capable of storing objects. The argument name may contain the complete name and classname, or parts of the name with wildcard and class name.

```
list DataObject.GetContents([str Name,]
                            [int recursive])
```

ARGUMENTS

Name (optional)
  loc name.class name, name possibly contains wildcards: '*' and '?' characters

recursive (optional)

1	All contained objects will be added recursively.
0	(default) Only direct children of current object will be collected.

RETURNS

Objects that are stored in the object.

GetControlledNode

Returns the target terminal and the resulting target voltage for generators and other voltage regulating units.

```
[DataObject node,
float targetVoltage] DataObject.GetControlledNode(int bus,
                                                   [int check])
```

ARGUMENTS

bus)

-1	currently controlled bus
0	HV bus
1	MV/ LV bus
2	LV bus

targetVoltage (out)
The target voltage of the voltage regulating unit in pu.

check (optional)

0	(default) Do not check if the control mode is set to voltage control.
1	Only return the controlled node if the control mode is set to voltage control.

RETURNS

Controlled node, None if no controlled terminal exists (or not voltage controlled if check=1)

GetCubicle

Returns the cubicle of an object at the connection with index n, or None if there is no cubicle inside the object.

```
DataObject DataObject.GetCubicle(int side)
```

ARGUMENTS

side The connection number.

RETURNS

The cubicle object or None.

GetFullName

Returns the full name of the object as a string.

```
str DataObject.GetFullName([int type])
```

ARGUMENTS

type(optional)
Is used to determine the format of the returned full name:

not given
No special formatting.

$=$ **0**
Same format as used in DataObject.ShowFullName() and also clickable when printed to the output window.

$>$ **0 (but less or equal to 190)**
Formatted exactly to this length and also clickable when printed to the output window.

RETURNS

The fullname (complete database path including the name and class name) of the object.

GetImpedance

Returns the positive sequence impedance of an element referred to a given voltage.

```
[int error,
float real,
float imag] DataObject.GetImpedance(float refVoltage,
                                    [int i3Trf])
```

ARGUMENTS

real (out) Real part of the impedance in Ohm.

imag (out)
Imaginary part of the impedance in Ohm.

refVoltage
Reference voltage for the impedance in kV.

i3Trf (optional)
When used with an *ElmTr3*

0	Return the HV-MV impedance.
1	Return the HV-LV impedance.
2	Return the MV-LV impedance.

RETURNS

1	An error occurred.
0	Otherwise.

SEE ALSO

object.GetZeroImpedance()

GetInom

Returns the nominal current of the object at given bus index.

```
float DataObject.GetInom([int busIndex = 0])
```

ARGUMENTS

busIndex (optional)
Bus index, default value is 0.

RETURNS

The nominal current at bus index.

SEE ALSO

DataObject.GetUnom()

GetNode

Returns the node connected to the object at specified bus index.

```
DataObject DataObject.GetNode(int busIndex,
                              [int considerSwitches = 0])
```

ARGUMENTS

busIndex Bus index.

considerSwitches (optional)

0	Ignore the status of the switches (default).
1	Consider the status of the switches.

RETURNS

object	Connected node object at specified bus index.
None	If no node at bus index is found.

GetOperator

Returns the element's operator (*ElmOperator*).

```
DataObject DataObject.GetOperator()
```

RETURNS

Object of class *ElmOperator* determined according to following rules

- If operator is set in current object instance (attribute "pOperator") this operator object is retured.
- Else the operator inherited from its parent is used (recursively applied).
- None if none if its parents have an operator set.

GetOwner

Returns the elements's owner (*ElmOwner*).

```
DataObject DataObject.GetOwner()
```

RETURNS

Object of class *ElmOwner* determined according to following rules

- If owner is set in current object instance (attribute “pOwner”) this owner object is retured.
- Else the owner inherited from its parent is used (recursively applied).
- None if none if its parents have an operator set.

GetParent

Returns the parent folder object (same as parameter ’fold_id’).

```
DataObject DataObject.GetParent()
```

RETURNS

DataObject The parent folder object.

None On the root database folder e.g. parent of a user.

SEE ALSO

DataObject.GetContents()

GetReferences

Returns a set containing all objects with references to the object the method was called on.

```
list DataObject.GetReferences([str filter,] [int includeSubsets])
```

ARGUMENTS

filter (optional)
Object filter to get only objects whose name matches this filter string, e.g. '*.'.

includeSubsets (optional)
Forces references from IntSubset objects to be evaluated. These are normally not included for performance reasons.

RETURNS

Set of referenced objects.

GetRegion

All network components are internally associated with an artificial region. A region consists of topologically connected elements. This means, two elements *elm1* and *elm2* are topologically connected ⇔ elm1.GetRegion() == elm2.GetRegion().
A region is simply identified by a number that can be access via this function.

```
int DataObject.GetRegion()
```

RETURNS

Region index >0. A value of ‘-1’ means status is unknown for that element (normally for not topology relevant elements).

GetSupplyingSubstations

Returns the closest supplying substation(s) for a network component.
"Closest" means that there is no other supplying element of same type in topological path between network component and the supplying component(s) returned by this function.

```
list DataObject.GetSupplyingSubstations()
```

RETURNS

List of substations (objects of class ElmSubstat). Can be empty.

SEE ALSO

ElmTr2.GetSuppliedElements(), ElmTr3.GetSuppliedElements(), ElmSubstat.GetSuppliedElements(), ElmTrfstat.GetSuppliedElements(), DataObject.GetSupplyingTransformers(), DataObject.GetSupplyingTrfstations

GetSupplyingTransformers

Returns the closest supplying transformer(s) for a network component. "Closest" means that there is no other supplying element of same type in topological path between network component and the supplying component(s) returned by this function.

```
list DataObject.GetSupplyingTransformers()
```

RETURNS

List of transformers (objects of class *ElmTr2* or *ElmTr3*). Can be empty.

SEE ALSO

ElmTr2.GetSuppliedElements(), ElmTr3.GetSuppliedElements(), ElmSubstat.GetSuppliedElements(), ElmTrfstat.GetSuppliedElements(), DataObject.GetSupplyingSubstations(), DataObject.GetSupplyingTrfstations()

GetSupplyingTrfstations

Returns the closest supplying transformer station(s) for a network component.
"Closest" means that there is no other supplying element of same type in topological path between network component and the supplying component(s) returned by this function.

```
list DataObject.GetSupplyingTrfstations()
```

RETURNS

List of transformer stations (objects of class ElmTrfstat). Can be empty.

SEE ALSO

ElmTr2.GetSuppliedElements(), ElmTr3.GetSuppliedElements(), ElmSubstat.GetSuppliedElements(), ElmTrfstat.GetSuppliedElements(), DataObject.GetSupplyingTransformers(), DataObject.GetSupplyingSubstatior

GetSystemGrounding

Returns the grounding type employed in the grounding area of the grid the object belongs to. The grounding area is defined by network components separating the zero sequence system (e.g. star-delta transformers).

```
int DataObject.GetSystemGrounding()
```

RETURNS

-1	grounding type can not be determined
0	system is solidly grounded
1	system is compensated
2	system is isolated

GetUnom

Returns the nominal voltage of the object.

```
float DataObject.GetUnom([int busIndex = 0])
```

ARGUMENTS

busIndex (optional)
Bus index, default value is 0.

RETURNS

The nominal voltage at bus index.

SEE ALSO

DataObject.GetInom()

GetUserAttribute

This function offers read-access to a simple form of user-defined attributes. These attributes must be defined in an XML-like syntax in the description field of an object (variable 'desc').

The syntax for a user-defined variable is:
<Attribute Name="name" Type="type">value</>

With:

name	Name of the attribute
type	Attribute type, valid values 'string', 'integer', 'double'
value	Current value of the attribute

Note: The format is case-sensitive!

Example:
<Attribute Name="var1" Type="int">2008< / >
<Attribute Name="var2" Type="string">Hello *PowerFactory* < / >

```
[int|float|str value,
int error            ] DataObject.GetUserAttribute(str attName)
```

ARGUMENTS

attName Name of the user-defined attribute

error (out)
Parameter for returned error value

RETURNS

0	Attribute found and value returned
1	Attribute could not be accessed (e.g. attribute not found in given object, definition is incomplete or wrong, format value is not compatible with given type).

GetZeroImpedance

Returns the zero sequence impedance of an element referred to a given voltage.

```
[int error,
float real,
float imag] DataObject.GetZeroImpedance(float refVoltage,
                                        [int i3Trf])
```

ARGUMENTS

real (out) Real part of the impedance in Ohm.

imag (out)
Imaginary part of the impedance in Ohm.

refVoltage
Reference voltage for the impedance in kV.

i3Trf (optional)
When used with an *ElmTr3*

0	Return the HV-MV impedance.
1	Return the HV-LV impedance.
2	Return the MV-LV impedance.

RETURNS

1	An error occurred.
0	Otherwise.

SEE ALSO

object.GetImpedance()

HasAttribute

Returns whether the given name is a currently valid attribute name.

```
int DataObject.HasAttribute(str name)
```

ARGUMENTS

name Name of an attribute.

RETURNS

0	Given name is not a currently valid attribute name.
1	Given name is a currently valid attribute name.

SEE ALSO

DataObject.GetAttribute(), DataObject.SetAttribute()

HasResults

Checks if the object has calculated result parameters.

```
int DataObject.HasResults([int ibus])
```

ARGUMENTS

ibus (optional)
Bus index

-1(default)	Checks if “c:” quantities exist
>= 0	Checks if 'm:xxxx:bus ' quantities exist for bus index=ibus
2	Hidden objects are returned

RETURNS

0	no results available
1	results exist

IsCalcRelevant

Returns whether the object is relevant for calculation.

```
int DataObject.IsCalcRelevant()
```

RETURNS

0	When the object is not used for calculations.
1	When the object is currently used for calculations.

SEE ALSO

Application.GetCalcRelevantObjects()

IsDeleted

Returns 1 if the object is deleted.

```
int DataObject.IsDeleted()
```

RETURNS

1	Object is already deleted.
0	Object is not deleted.

IsEarthed

Checks if a network component is topologically connected to any earthed component. Earthing components are terminals / busbars (*ElmTerm*) with attribute ‘iEarth’ = 1 and all closed grounding switches (*ElmGndswt*). An energized component is never considered to be earthed.

```
int DataObject.IsEarthed()
```

RETURNS

1	Component is earthed (connected to an earthing component)
0	Component is not earthed

IsEnergized

Checks if a network component is energized. A component is considered to be energized, if it is topologically connected to a generator. All other elements are considered to be deenergized.

```
int DataObject.IsEnergized()
```

RETURNS

1	Component is energized
0	Component is deenergized
-1	Component has no energizing status (status unknown)

IsHidden

Checks whether an object is hidden with respect to currently activated variation. An object is hidden if it is

- *deleted* in currently active variation or
- *added* in a variation that is currently not active

```
int DataObject.IsHidden()
```

RETURNS

0	not hidden, currently ’active’
1	hidden, currently ’inactive’

IsInFeeder

Indicates if the object belongs to the feeder area defined by “Feeder”.

```
int DataObject.IsInFeeder(DataObject Feeder,
                          [int OptNested=0])
```

ARGUMENTS

Feeder The Feeder definition object *ElmFeeder*

OptNested (optional

0	Nested feeders are not considered.
1	Nested feeders are considered.

RETURNS

1	If “Feeder” is a feeder definition and the object is in the feeder area.
0	Otherwise

IsNetworkDataFolder

Checks whether given object is a special folder within a project that stores specific data elements. Each project can not have more than one instance per folder type.
The following folder types are distinguished (*PowerFactory* class names):

IntArea stores *ElmArea* objects

IntBbone stores *ElmBbone* and *SetBbone* objects

IntBmu stores *ElmBmu* objects

IntBoundary stores *ElmBoundary* objects

IntCircuit stores *ElmCircuit* objects

IntFeeder stores *ElmFeeder* objects

IntMeteostat stores *ElmMeteostat* objects

IntOperator stores *ElmOperator* objects

IntOwner stores *ElmOwner* objects

IntPath stores *SetPath* objects

IntRoute stores *ElmRoute* objects

IntScales stores *Tri** objects

```
int DataObject.IsNetworkDataFolder()
```

RETURNS

0	false, object is not a network data folder
1	true, object is a network data folder

SEE ALSO

Application.GetDataFolder()

IsNode

Indicates wtheter an object is a node (terminal or busbar).

```
int DataObject.IsNode()
```

RETURNS

1	Object is a node.
0	Otherwise.

IsObjectActive

Check if an object is active for specific time.

```
int DataObject.ReportUnusedObjects(int time)
```

RETURNS

0	Object is not active (hidden or deleted)
1	Object is active

IsObjectModifiedByVariation

Check if an object is active for specific time.

```
int DataObject.ReportUnusedObjects(int considerADD, int considerDEL, int considerDELTA)
```

ARGUMENTS

considerADD
checks if an ADD-object exists

0	ignore ADD-objects
1	consider ADD-objects

considerDEL
check if a DELETE-Object exists or exist for the parent objects

0	ignore DELETE-objects
1	consider DELETE-objects

considerDELTA
check if a DELTA-Object exists

0	ignore DELTA-objects
1	consider DELTA-objects

RETURNS

0	Object is not modified by an active variation
1	Object is modified by an active variation

Isolate

Performs an “isolate” action on the network element. This corresponds to performing a “switch off” action followed by an additional earthing of switched off region.
The action is identical to that in the context menue.

```
[int error,
list changedSwitches] DataObject.Isolate([int resetRA,]
                                         [int isolateCBs])
```

ARGUMENTS

changedSwitches (optional, out)
All switches whose switching state was changed by the action are filled into this set

resetRA (optional)
Determines whether an active running arrangement that would prevent switching action should be deactivated or not.

1	All running arrangements that cause blocking of relevant switches are applied and reset automatically before the switching is performed.

0	(default) Active running arrangements are not reset. Blocked switches will cause the switching action to fail

isolateCBs (optional)
Determines if, in addition, circuit breakers should be isolated by opening its adjacent disconnectors (if not given, default will be taken from project settings)

0	No additional opening of disconnectors
1	Also open disconnectors adjacent to switched circuit breakers)

RETURNS

Information about the success of the action:

0	Action was successful
1	Action failed

SEE ALSO

DataObject.SwitchOn(), DataObject.SwitchOff(), DataObject.Energize()

IsOutOfService

Indicates whether or not the object is currently out of service.

```
int DataObject.IsOutOfService()
```

RETURNS

0	When the object is in service.
1	When the object is out of service.

IsReducible

Checks if object can be reduced during network reduction.

```
int DataObject.IsReducible()
```

RETURNS

0	object can never be reduced.
1	object can be reduced (e.g. switch, zero-length lines)
2	in principle the object can be reduced, but not now (e.g. switch that is set to be detailed)

IsShortCircuited

Returns whether an element is short-circuited or not.

RETURNS

0	No short-circuit found.
1	Element is short-circuited.

MarkInGraphics

Marks the object in the diagram in which the element is found by hatch crossing it. By default all the currently opened diagrams are searched for the element to mark beginning with the diagram shown. The first diagram in which the element is found will be opened and the element is marked.
Alternatively the search can be extended to all existing diagrams by passing 1 as parameter. If the element exists in more than one diagram the user can select from a list of diagrams which diagram shall be opened.

```
None DataObject.MarkInGraphics([int SearchAllDiagramsAndSelect])
```

ARGUMENTS

SearchAllDiagramsAndSelect (optional)
Search can be extended to all diagrams, not only the ones which are currently shown on the desktop.

0	Only search in currently opened diagrams and open the first diagram in which the element was found.
1	Searching all diagrams, not only the ones which are currently shown on the desktop. If there is more than one occurrence the user will be prompted which diagrams shall be opened.

RETURNS

A diagram in which the element is drawn is opened and the element is marked.

Move

Moves an object or a set of objects to this folder.

```
int DataObject.Move(DataObject O)
int DataObject.Move(list S)
```

ARGUMENTS

O(optional)
Object to move

S(optional)
Set of objects to move

RETURNS

0	on success
1	on error

PasteCopy

This function pastes the copy of the given object(s) into this (=target) using the merge tool when source and target are inside different projects (equivalent to a manual copy&paste operation).

```
int DataObject.PasteCopy(DataObject oCopyObj)
int DataObject.PasteCopy(list oCopyObj)
```

ARGUMENTS

oCopyObj
Object to be copied

sCopySet
Set of object to be copied

RETURNS

0	Object(s) successfully copied
1	Error

PurgeUnusedObjects

The function deletes the following child objects:

1. All 'hidden' objects without corresponding "Add" object. These objects are only deleted, if the condition is fulfilled for all child objects (hidden without corresponding 'Add' object).
2. All internal expansion stage objects with invalid target object (target object reference is missing).

It's crucial that there is no study case active when executing the function.

```
None DataObject.PurgeUnusedObjects()
```

SEE ALSO

DataObject.ReportUnusedObjects()

ReplaceNonAsciiCharacters

Replaces all non-ASCII characters in all text attributes by similar ASCII characters. Emits a warning if a charcter can not replaced, because no replacement character was defined.

```
int DataObject.ReplaceNonAsciiCharacters(DataObject map,
                                         str defaultReplacementCharacter)
```

ARGUMENTS

map IntMat object with two columns: the first column contains the codes of the non-ASCII character, the second column contains the code of the ASCII character.

defaultReplacementCharacter
String containing one ASCII character. If map does not contain a replacement for a non-ASCII character, it is replaced by defaultReplacementCharacter.

RETURNS

Returns 1 when the function was executed successfully.

ReportNonAsciiCharacters

Reports all text attributes of this objects containing non-ASCII characters in the output window.

```
None DataObject.ReportNonAsciiCharacters()
```

ReportUnusedObjects

Prints a report in the PowerFactory output window, which object will be deleted when function DataObject.PurgeUnusedObjects() is called. It’s crucial that there is no study case active when executing the function.

```
None DataObject.ReportUnusedObjects()
```

SEE ALSO

DataObject.PurgeUnusedObjects()

SearchObject

Searches for an object with a full name, such as 'rootfolder.class\subfolder.class\...\locname.class '.

```
DataObject DataObject.SearchObject(str name)
```

ARGUMENTS

name	string to search

RETURNS

Returns the searched object.

SEE ALSO

DataObject.GetFullName()

SetAttribute

Sets the value of an attribute.

```
None DataObject.SetAttribute(str name,
                             int|float|str|DataObject|list value)
```

ARGUMENTS

name	Name of an attribute.
value	Value to be set in its current unit (like in the edit dialog seen). An exception is thrown for invalid attribute names.

SEE ALSO

DataObject.GetAttribute(), DataObject.GetAttributeType()

SetAttributeLength

Sets the length of a vector or matrix attribute. The length of a matrix attribute is the number of rows.

```
int DataObject.SetAttributeLength(str name, int length)
```

ARGUMENTS

name	Name of an attribute.
length	New length of the attribute.

RETURNS

0	On success.
1	On error.

SEE ALSO

DataObject.SetAttributeShape(), DataObject.GetAttributeLength(), DataObject.GetAttributeType()

SetAttributeShape

Sets the shape of a matrix or vector attribute. The shape is a list of the form [number of rows, number of colums]. Number of colums has to be 0 for vectors.

```
int DataObject.SetAttributeShape(str name, list(int, int) shape)
```

ARGUMENTS

name	Name of an attribute.
shape	New shape of the attribute.

RETURNS

0	On success.
1	On error.

SEE ALSO

DataObject.SetAttributeLength(), DataObject.GetAttributeShape(), DataObject.GetAttributeType()

ShowEditDialog

Opens the edit dialogue of the object. Command objects (such as *ComLdf*) will have their "Execute" button disabled. The execution of the running script will be halted until the edit dialogue is closed again.

Editing of command objects (*ComDPL*, *ComPython*) is not supported.

```
int DataObject.ShowEditDialog()
```

RETURNS

1	Edit dialogue was cancelled by the user.
0	Otherwise.

ShowModalSelectTree

Shows the current database object tree. The element on which the function is called on is initially selected.

```
DataObject DataObject.ShowModalSelectTree([str title,]
                                          [str filter])
```

ARGUMENTS

title (optional)
Title of the dialog. If omitted, a default title will be used.

filter (optional)
Classname filter e.g. 'ElmLne' or 'Com*'. If set, a selection is only accepted if the classname of the selected object matches that filter.

RETURNS

DataObject	Selected object.
None	No object selcted e.g. 'Cancel' clicked.

SwitchOff

Performs a “switch off” action on the network element. This action is identical to that in the context menue.

```
[int error,
list changedSwitches] DataObject.SwitchOff([int resetRA]
                                           [int simulateOnly])
```

ARGUMENTS

changedSwitches (optional, out)
All switches whose switching state was changed by the action are filled into this set

resetRA (optional)
Determines whether an active running arrangement that would prevent switching action should be deactivated or not.

1	All running arrangements that cause blocking of relevant switches are applied and reset automatically before the switching is performed.
0	(default) Active running arrangements are not reset. Blocked switches will cause the switching action to fail

simulateOnly (optional)
Can be used to get the switches that would be changed. No switching is performed if set to ‘1’. (default is ’0’)

RETURNS

Information about the success of the action:

0	Action was successful
1	Action failed

SEE ALSO

DataObject.SwitchOn(), DataObject.Isolate(), DataObject.Energize()

SwitchOn

Performs a “switch on” action on the network element. This action is identical to that in the context menue.

```
[int error,
list changedSwitches] DataObject.SwitchOn([int resetRA,]
                                          [int simulateOnly])
```

ARGUMENTS

changedSwitches (optional, out)
All switches whose switching state was changed by the action are filled into this set

resetRA (optional)
Determines whether an active running arrangement that would prevent switching action should be deactivated or not.

1	All running arrangements that cause blocking of relevant switches are applied and reset automatically before the switching is performed.
0	(default) Active running arrangements are not reset. Blocked switches will cause the switching action to fail

simulateOnly (optional)
Can be used to get the switches that would be changed. No switching is performed if set to '1'. (default is '0')

RETURNS

Iinformation about the success of the action:

0	Action was successful
1	Action failed

SEE ALSO

DataObject.SwitchOff(), DataObject.Isolate(), DataObject.Energize()

WriteChangesToDb

See Application.WriteChangesToDb() for a detailed description.

```
None DataObject.WriteChangesToDb()
```

4.2 Network Elements

4.2.1 ElmArea

Overview

CalcBoundary
CalculateInterchangeTo
GetAll
GetBranches
GetBuses
GetObjs

CalcBoundary

Defines boundary with this area as exterior part. Resulting cubicles of boundary are branch-oriented away from the area.

```
[int error,
DataObject boundary] ElmArea.CalcBoundary(float shift)
```

ARGUMENTS

shift	Elements that are within a distance of shift many elements to a boundary cubicle of the area are added to the exterior part of the resulting boundary.
boundary (out)	Defined boundary.

RETURNS

0	Successful call, boundary defined.
1	Error during determination of boundary cubicles.

CalculateInterchangeTo

Calculates interchange power to the given area (calculated quantities are: Pinter, Qinter, Pexport, Qexport, Pimort, Qimport). Prior the calculation the valid load flow calculation is required.

```
int ElmArea.CalculateInterchangeTo(DataObject area)
```

ARGUMENTS

area	Area to which the interchange is calculated

RETURNS

$<$ **0**	Calculation error (i.e. no valid load flow, empty area...)
0	No interchange power to the given area
1	Interchange power calculated

GetAll

Returns all objects which belong to this area.

```
list ElmArea.GetAll()
```

RETURNS

The set of contained objects.

GetBranches

Returns all branches which belong to this area.

```
list ElmArea.GetBranches()
```

RETURNS

The set of branch objects.

GetBuses

Returns all buses which belong to this area.

```
list ElmArea.GetBuses()
```

RETURNS

The set of objects.

GetObjs

Returns all objects of the given class which belong to this area.

```
list ElmArea.GetObjs(str classname)
```

ARGUMENTS

classname
Name of the class (i.e. ”ElmTr2”).

RETURNS

The set of objects.

4.2.2 ElmAsm

Overview

GetAvailableGenPower
GetElecTorque
GetGroundingImpedance
GetMechTorque
GetMotorStartingFlag
GetStepupTransformer
IsPQ

GetAvailableGenPower

Returns the available power that can be dispatched from the generator, for the particular study time.
For the case of conventional generators (no wind generation selected), the available power is equal to the nominal power specified.
For wind generators, the available power will depend on the wind model specified:

- No Wind Model: No available power.
- Stochastic Wind Model: Given the specified mean wind speed, the available power is calculated from the Power Curve. If the units of the Power Curve are in MW, the returned value is directly the available power. In the other hand, if the units are in PU, the returned value is multiplied by the nominal power of the generator to return the available power.
- Time Series Characteristics of Active Power Contribution: The available power is the average of the power values (in MW) obtained from all the specified time characteristics for the current study time.
- Time Series Characteristics of Wind Speed: The available power is calculated with the average of the power values (in MW) calculated for all the specified time characteristics. A power value for any time characteristic is calculated by obtaining the wind speed for the current study time, and then calculating the power from the specified Power Curve. If the units of the Power Curve are in MW, the returned value is directly the power value. In the other hand, if the units are in PU, the returned value is multiplied by the nominal power of the generator to return the power value.

For motors, the available power is zero.

```
float ElmAsm.GetAvailableGenPower()
```

RETURNS

Available generation power

GetElecTorque

Calculates the electrical torque for a given speed and voltage.

```
float ElmAsm.GetElecTorque(float speed,
                           float uReal,
                           [float addZReal,]
                           [float addZImag]
                           )
```

ARGUMENTS

speed speed value in p.u.

uReal voltage value (real part) in p.u.

addZReal (optional)
additional impedance (real part) in p.u.

addZImag (optional)
additional impedance (imaginary part) in p.u.

RETURNS

Returns the calculated electrical torque.

GetGroundingImpedance

Returns the impedance of the internal grounding.

```
[int valid,
float resistance,
float reactance  ] ElmAsm.GetGroundingImpedance(int busIdx)
```

ARGUMENTS

busIdx Bus index where the grounding should be determined.

resistance (out)
Real part of the grounding impedance in Ohm.

reactance (out)
Imaginary part of the grounding impedance in Ohm.

RETURNS

0	The values are invalid (e.g. because there is no internal grounding)
1	The values are valid.

GetMechTorque

Calculates the electrical torque for a given speed and voltage.

```
float ElmAsm.GetMechTorque(float speed,
                           float uReal
                           )
```

ARGUMENTS

speed	speed value in p.u.
uReal	voltage value (real part) in p.u.

RETURNS

Returns the calculated mechanical torque.

GetMotorStartingFlag

Returns the starting motor condition.

```
int ElmAsm.GetMotorStartingFlag()
```

RETURNS

Returns the motor starting condition. Possible values are:

-1	in the process of being calculated
0	not calculated
1	successful start
2	unsuccessful start

GetStepupTransformer

Performs a topological search to find the step-up transformer of the asynchronous machine.

```
DataObject ElmAsm.GetStepupTransformer(float Voltage,
                                       int swStatus
                                       )
```

ARGUMENTS

hvVoltage
voltage level at which the search will stop

ignSwtStatus
consideration of switch status. Possible values are:

0	consider all switch status
1	ignore breaker status
2	ignore all switch status

RETURNS

Returns the first collected step-up transformer object. It is empty if not found (e.g. start terminal already at hvVoltage).

IsPQ

Informs whether or not it is a ”PQ” machine (constant Q control mode).

```
int ElmAsm.IsPQ()
```

RETURNS

Returns 1 if it is a ”PQ” machine.

4.2.3 ElmAsmsc

Overview

GetAvailableGenPower
GetGroundingImpedance
GetStepupTransformer

GetAvailableGenPower

Returns the available power that can be dispatched from the generator, for the particular study time.
For the case of conventional generators (no wind generation selected), the available power is equal to the nominal power specified.
For wind generators, the available power will depend on the wind model specified:

- No Wind Model: No available power.
- Stochastic Wind Model: Given the specified mean wind speed, the available power is calculated from the Power Curve. If the units of the Power Curve are in MW, the returned value is directly the available power. In the other hand, if the units are in PU, the returned value is multiplied by the nominal power of the generator to return the available power.
- Time Series Characteristics of Active Power Contribution: The available power is the average of the power values (in MW) obtained from all the specified time characteristics for the current study time.
- Time Series Characteristics of Wind Speed: The available power is calculated with the average of the power values (in MW) calculated for all the specified time characteristics. A power value for any time characteristic is calculated by obtaining the wind speed for the current study time, and then calculating the power from the specified Power Curve. If the units of the Power Curve are in MW, the returned value is directly the power value. In the other hand, if the units are in PU, the returned value is multiplied by the nominal power of the generator to return the power value.

For motors, the available power is zero.

```
float ElmAsmsc.GetAvailableGenPower()
```

RETURNS

Available generation power

GetGroundingImpedance

Returns the impedance of the internal grounding.

```
[int valid,
float resistance,
float reactance  ] ElmAsmsc.GetGroundingImpedance(int busIdx)
```

ARGUMENTS

busIdx Bus index where the grounding should be determined.

resistance (out)
Real part of the grounding impedance in Ohm.

reactance (out)
Imaginary part of the grounding impedance in Ohm.

RETURNS

0	The values are invalid (e.g. because there is no internal grounding)
1	The values are valid.

GetStepupTransformer

Performs a topological search to find the step-up transformer of the asynchronous machine.

```
DataObject ElmAsmsc.GetStepupTransformer(float Voltage,
                                         int swStatus
                                         )
```

ARGUMENTS

hvVoltage
voltage level at which the search will stop

ignSwtStatus
consideration of switch status. Possible values are:

0	consider all switch status
1	ignore breaker status
2	ignore all switch status

RETURNS

Returns the first collected step-up transformer object. It is empty if not found (e.g. start terminal already at hvVoltage).

4.2.4 ElmBbone

Overview

CheckBbPath
GetBbOrder
GetCompleteBbPath
GetFOR
GetMeanCs
GetMinCs
GetTieOpenPoint
GetTotLength
HasGnrlMod

CheckBbPath

Check whether the backbone object is still valid. This means:

a	Terminals determining backbone path are still directly connected.
b	One switch is open on the path of an inter-feeder backbone.
c	Contents of backbone match specified starting-feeder (and end feeder).
d	Start and end of feeder are calculation-relevant.
e	Path is unique via the defined terminals (no parallel elements (only warning!)).

```
int ElmBbone.CheckBbPath(int outputMsg)
```

ARGUMENTS

outputMsg

1	Output resulting messages of check function.
0	Only check, no output of messages.

RETURNS

0	Backbone is valid.
1	Backbone is invalid because of one or more of the above listed reasons.

GetBbOrder

Get order of backbone object, determined by backbone calculation according to the selected criterion.

```
int ElmBbone.GetBbOrder()
```

RETURNS

The order of the backbone object. The smaller the returned value, the better the backbone according to chosen criterion. The order 1 is returned for the best backbone.

GetCompleteBbPath

Get the complete (ordered) path containing all terminals and connecting elements of the backbone.

```
list ElmBbone.GetCompleteBbPath(int iReverse,
                                [int iStopAtTieOpen = 0])
```

ARGUMENTS

AllElmsOnBb (out)
Ordered path containing all terminals and connecting elements of the backbone.

iReverse

0	Return ordered path from start feeder to end feeder
1	Return ordered path from end feeder to start feeder

iStopAtTieOpen

0	return complete path

1	only return part of path in start feeder (iReverse=0) / in end feeder (iReverse=1)

GetFOR

Get aggregated forced outage rate (FOR) of all elements on the path of the backbone.

```
float ElmBbone.GetFOR()
```

RETURNS

The aggregated forced outage rate (FOR) of all elements on the path of the backbone [in 1/a].

GetMeanCs

Get mean cross section value of all elements on the path of the backbone. Every cross section value is weighted with the relative length corresponding to the total length of the backbone.

```
float ElmBbone.GetMeanCs()
```

RETURNS

The mean cross section of the elements on the backbone path [in mm2].

GetMinCs

Get minimum cross section value of all elements on the path of the backbone. Optional: a set with all elements on the backbone path featuring this cross section may be returned.

```
[float minCs,
set ElmsMinCs] ElmBbone.ElmBoundary.IsSplitting()
```

ARGUMENTS

ElmsMinCs
Elements on the backbone path featuring minimum cross section value.

RETURNS

The minimum cross section of all elements on the backbone path [in mm2].

GetTieOpenPoint

Search and obtain the first open switching device (ElmCoup, StaSwitch) on the backbone path (starting from the infeeding point of the starting feeder).

```
DataObject ElmBbone.GetTieOpenPoint()
```

RETURNS

The switching device (ElmCoup or StaSwitch) or None if backbone is invalid.

GetTotLength

Get total lenth of all elements on the path of the backbone.

```
float ElmBbone.GetTotLength()
```

RETURNS

The total length of the backbone path [in km].

HasGnrlMod

Check whether backbone object ElmBbone has a valid CalBbone where corresponding results are stored. This is only the case after a backbone calculation by scoring method (until the calculation is reset).

```
int ElmBbone.HasGnrlMod()
```

RETURNS

1	ElmBbone has a calculation model,
0	no calculation model available.

4.2.5 ElmBmu

Overview

Apply

Applies the power dispatch. Depending on the selected 'Distribution Mode ' this is done by a built-in algorithm based on 'Merit Order' or by a user-defined DPL script that is stored in the contents of the virtual power plant object.

```
int ElmBmu.Apply()
```

RETURNS

0	on success, no error occurred
1	error during dispatch by virtual power plant. Please note, a value of 1 is also returned in case the power plant is current set out-of-service.

Update

Updates the list of machines in the tables: 'Dispatchable Machines' and 'Non-dispatchable (fixed) Machines'.

```
None ElmBmu.Update()
```

4.2.6 ElmBoundary

Overview

AddCubicle

Adds a given cubicle with given orientation to an existing boundary. The cubicle is added only if it is not already contained within the boundary.

```
int ElmBoundary.AddCubicle(DataObject cubicle,
                           int orientation
                           )
```

RETURNS

0	cubicle was successfully added
1	cubicle was not added because it is already contained (including given orientation)

CalcShiftedReversedBoundary

Defines boundary where exterior and interior part of this boundary are exchanged. Resulting boundary cubicles are branch-oriented.

```
[int error,
DataObject boundary] ElmBoundary.CalcShiftedReversedBoundary(float shift)
```

ARGUMENTS

shift Elements that are within a distance of shift many elements to a boundary cubicle of this boundary are added to the exterior part of the resulting boundary.

boundary (out)
Defined boundary.

RETURNS

0	Successful call, boundary defined.
1	Error during determination of boundary cubicles.

Clear

Removes all boundary cubicles from an existing boundary.

```
None ElmBoundary.Clear()
```

GetInterior

Returns a set of all elements that are contained in the interior region of the boundary.

```
list ElmBoundary.GetInterior()
```

RETURNS

Returns the set of interior elements.

IsSplitting

Checks if the boundary splits the network into two regions. A boundary is called splitting, if and only if, for each boundary cubicle, the adjacent terminal and the adjacent branch component belong to different sides of the boundary.

```
[int isSplitting,
list notSplittingCubicles] ElmBoundary.IsSplitting()
```

ARGUMENTS

notSplittingCubicles (optional, out)
All cubicles that prevent the boundary from being splitting are filled into this set.

RETURNS

0	not splitting boundary
1	splitting boundary

Resize

Resizes the boundary cubicle vector or the cubicle orientation vector. It is strongly advised that the size of both vectors must be the same.

```
None ElmBoundary.Resize(float size,
                        str name
                        )
```

ARGUMENTS

size	size of the referenced vector (number of cubicles)
name	reference to the vector ('iorient' or 'cubicles')

RETURNS

If the resize is unsuccessful the error message shall be issued.

Update

Updates cached information (such as topological interiour). Required when boundary definition was changed via DPL or Python.

```
None ElmBoundary.Update()
```

4.2.7 ElmBranch

Overview

Update

Update

Updates connection points and contained elements of the branch. If the branch element externally modified by the user, then the update shall refresh all connections in the correct manner. Behaves same as the update button within the ElmBranch.

```
None ElmBranch.Update()
```

4.2.8 ElmCabsys

Overview

FitParams
GetLineCable
Update

FitParams

Calculates distributed parameters for cable system elements. Whether this function calculates constant parameters or frequency dependent parameters depends on the user setting of the parameter 'i_model' in the ElmCabsys dialog. The settings are as follows: i_model=0: constant parameters; i_model=1: frequency dependent parameters.

```
int ElmCabsys.FitParams()
```

RETURNS

0	on success
1	on error

GetLineCable

Gets cable type for the corresponding line, within the cable system.

RETURNS

cable type	on success
None	on error

```
DataObject ElmCabsys.GetLineCable()
```

Update

Updates cable system element depending on configuration of the associated cable system type.

```
int ElmCabsys.Update()
```

RETURNS

1	On success.
0	On error.

4.2.9 ElmComp

Overview

SlotUpdate

SlotUpdate

Performs a slot update for the composite model, to try to reassign each model found in the composite model contents to the corresponding slot.

```
None ElmComp.SlotUpdate()
```

DEPRECATED NAMES

Slotupd

4.2.10 ElmCoup

Overview

Close
GetRemoteBreakers
IsBreaker
IsClosed
IsOpen
Open

Close

Closes the switch by changing its status to 'close'. This action will fail if the status is currently determined by an active running arrangement.

```
int ElmCoup.Close()
```

RETURNS

0	On success
$\neq$ **0**	On error

SEE ALSO

ElmCoup.Open()

GetRemoteBreakers

Returns the remote circuit breakers and connected bus bars.
This information is determined by a topological search that starts at given breaker in all directions, stopping at

- switches of type circuit breaker
- switches that are open
- busbars (ElmTerm::iUsage == 0)

which are connected by non-reducible components (see DataObject.IsReducible()) only. If search stops at a breaker that is in given breaker state (desiredBreakerState), it is added to the returned breakers collection. All busbars at which the search stops are added to the busbar collection.
Note: the remote breakers found in the same direction as a found bus bar are excluded.

```
[list remoteBreakers,
list foundBreakers,
list foundBusbars    ] ElmCoup.GetRemoteBreakers(int desiredBreakerState)
```

ARGUMENTS

desiredBreakerState
Only breakers with given status are collected.

-1	Return all remote circuit breakers
1	Return all closed remoted circuit breakers
0	Return all opened remoted circuit breakers

foundBreakers (out)
The list of the remote circuit breakers

foundBusbars (optional, out)
The list of the local bus bars

IsBreaker

Checks if type of current switch is ’circuit-breaker’.

```
int ElmCoup.IsBreaker()
```

RETURNS

1	switch is a circuit-breaker,
0	switch is not a circuit-breaker

IsClosed

Returns information about current switch state.

```
int ElmCoup.IsClosed()
```

RETURNS

1	switch is closed
0	switch is open

SEE ALSO

ElmCoup.IsOpen()

IsOpen

Returns information about current switch state.

```
int ElmCoup.IsOpen()
```

RETURNS

1	switch is open
0	switch is closed

SEE ALSO

ElmCoup.IsClosed()

Open

Opens the switch by changing its status to 'open'. This action will fail if the status is currently determined by an active running arrangement.

```
int ElmCoup.Open()
```

RETURNS

0	On success
$\neq$ **0**	On error

SEE ALSO

ElmCoup.Close()

4.2.11 ElmDsl

Overview

ExportToClipboard
ExportToFile

ExportToClipboard

Export the parameter list to clipboard.

```
None ElmDsl.ExportToClipboard([str colSeparator],
                              [int useLocalHeader]
                              )
```

ARGUMENTS

colSeparator (optional)
Separator between the columns (default: tab character).

useLocalHeader (optional)
Use the localised version of the header. Possible values are:

1	Yes (default).
0	No (use English language header).

ExportToFile

Export the parameter list to CSV file(s).

```
None ElmDsl.ExportToFile(str filePath,
                         [str colSeparator],
                         [int useLocalHeader]
                         )
```

ARGUMENTS

filePath Path of the CSV target file. In case of array and matrix parameters (names: “array_NAME” and “matrix_NAME”), additional CSV files are created in the same location with names obtained by appending “_array_NAME” and “_matrix_NAME” to the target file name.

colSeparator (optional)
Separator between the columns (default: “;”).

useLocalHeader (optional)
Use the localised version of the header. Possible values are:

1	Yes (default).
0	No (use English language header).

4.2.12 ElmFeeder

Overview

CalcAggrVarsInRadFeed
GetAll
GetBranches
GetBuses
GetNodesBranches
GetObjs

CalcAggrVarsInRadFeed

Computes all the aggregated variables in radial feeders.

```
int ElmFeeder.CalcAggrVarsInRadFeed([int lookForRoot,]
                                    [int considerNested])
```

ARGUMENTS

lookForRoot (optional)
Calculates the variables from the deepest root. Possible values are:

0	Start from this feeder
1	(default) Find the deepest root.

considerNested (optional)
Calculates the variables also for any nested subfeeders. Possible values are:

0	Ignore any nested feeders
1	(default) Consider nested feeders.

RETURNS

Returns whether or not the aggregated variables were calculated. Possible values are:

0	error during calculation
1	calculated correctly

GetAll

Returns a set with all objects belonging to this feeder.

```
list ElmFeeder.GetAll([int iNested])
```

ARGUMENTS

iNested (optional)
Affects the collection of objects in case of nested feeders:

0	Only the objects of this feeder will be returned.
1	(default) All elements including those of nested feeders will be returned.

RETURNS

The set of network elements belonging to this feeder. Can be empty.

GetBranches

Returns a set with all branch elements belonging to this feeder.

```
list ElmFeeder.GetBranches([int iNested])
```

ARGUMENTS

iNested (optional)
Affects the collection of objects in case of nested feeders:

0	Only the objects of this feeder will be returned.
1	(default) All elements including those of nested feeders will be returned.

RETURNS

The set of bus and branch elements in feeder.

GetBuses

Returns a set with all buses belonging to this feeder.

```
list ElmFeeder.GetBuses([int iNested])
```

ARGUMENTS

iNested (optional)
Affects the collection of objects in case of nested feeders:

0	Only the objects of this feeder will be returned.
1	(default) All elements including those of nested feeders will be returned.

RETURNS

The set of bus elements in feeder.

GetNodesBranches

Returns a set with all buses and branches belonging to this feeder.

```
list ElmFeeder.GetNodesBranches([int iNested])
```

ARGUMENTS

iNested (optional)
Affects the collection of objects in case of nested feeders:

0	Only the objects of this feeder will be returned.
1	(default) All elements including those of nested feeders will be returned.

RETURNS

The set of bus and branch elements in feeder.

GetObjs

Returns a set with all objects of class 'ClassName' which belong to this feeder.

```
list ElmFeeder.GetObjs(str ClassName,
                       [int iNested])
```

ARGUMENTS

iNested (optional)
Affects the collection of objects in case of nested feeders:

0	Only the objects of this feeder will be returned.
1	(default) All elements including those of nested feeders will be returned.

RETURNS

The set of feeder objects.

4.2.13 ElmFile

Overview

LoadFile
SaveFile

LoadFile

(Re)Loads the file into a buffer.

```
int ElmFile.LoadFile([int loadComplete = 1])
```

ARGUMENTS

loadComplete (optional)

0	Removes all points in the future simulation time and adds all points from the file (including the current interpolated value).
1	Clears the buffer and reloads the complete file (default).

RETURNS

0	On success.
$\neq 0$	On error.

SaveFile

Saves the buffer and overwrites the file.

```
int ElmFile.SaveFile()
```

RETURNS

0	On success.
$\neq 0$	On error.

4.2.14 ElmFilter

Overview

GetGroundingImpedance

GetGroundingImpedance

Returns the impedance of the internal grounding. Single phase filters connected to neutral are considered as grounding devices themselves; i.e. instead of the dedicated grounding parameters, the filters parameters are used.

```
[int valid,
float resistance,
float reactance  ] ElmFilter.GetGroundingImpedance(int busIdx)
```

ARGUMENTS

busIdx Bus index where the grounding should be determined.

resistance (out)
Real part of the grounding impedance in Ohm.

reactance (out)
Imaginary part of the grounding impedance in Ohm.

RETURNS

0	The values are invalid (e.g. because there is no internal grounding)
1	The values are valid.

4.2.15 ElmGenstat

Overview

Derate

Derates the value of the Max. Active Power Rating according to the specified value given in MW.
The following formula is used: $Pmax\_uc = Pmax\_uc - "Deratingvalue"$.

```
None ElmGenstat.Derate(float deratingP)
```

ARGUMENTS

deratingP Derating value

Disconnect

Disconnects a static generator by opening the first circuit breaker. The topological search performed to find such a breaker, stops at any busbar.

```
int ElmGenstat.Disconnect()
```

RETURNS

0	breaker already open or successfully opened
1	an error occurred (no breaker found, open action not possible (earthing / RA))

GetAvailableGenPower

Returns the available power that can be dispatched from the generator, for the particular study time.
For the case of conventional generators (no wind generation selected), the available power is equal to the nominal power specified.
For wind generators, the available power will depend on the wind model specified:

- No Wind Model: No available power.
- Stochastic Wind Model: Given the specified mean wind speed, the available power is calculated from the Power Curve. If the units of the Power Curve are in MW, the returned value is directly the available power. In the other hand, if the units are in PU, the returned value is multiplied by the nominal power of the generator to return the available power.
- Time Series Characteristics of Active Power Contribution: The available power is the average of the power values (in MW) obtained from all the specified time characteristics for the current study time.

- Time Series Characteristics of Wind Speed: The available power is calculated with the average of the power values (in MW) calculated for all the specified time characteristics. A power value for any time characteristic is calculated by obtaining the wind speed for the current study time, and then calculating the power from the specified Power Curve. If the units of the Power Curve are in MW, the returned value is directly the power value. In the other hand, if the units are in PU, the returned value is multiplied by the nominal power of the generator to return the power value.

For motors, the available power is zero.

```
float ElmGenstat.GetAvailableGenPower()
```

RETURNS

Available generation power

GetGroundingImpedance

Returns the impedance of the internal grounding.

```
[int valid,
float resistance,
float reactance  ] ElmGenstat.GetGroundingImpedance(int busIdx)
```

ARGUMENTS

busIdx Bus index where the grounding should be determined.

resistance (out)
Real part of the grounding impedance in Ohm.

reactance (out)
Imaginary part of the grounding impedance in Ohm.

RETURNS

0	The values are invalid (e.g. because there is no internal grounding)
1	The values are valid.

GetStepupTransformer

Performs a topological search to find the step-up transformer of the static generator.

```
DataObject ElmGenstat.GetStepupTransformer(float voltage,
                                           int swStatus
                                           )
```

ARGUMENTS

voltage voltage level at which the search will stop

swStatus consideration of switch status. Possible values are:

0	consider all switch status
1	ignore breaker status
2	ignore all switch status

RETURNS

Returns the first collected step-up transformer object. It is empty if not found (e.g. start terminal already at hvVoltage).

IsConnected

Checks if generator is topologically connected to any busbar.

```
int ElmGenstat.IsConnected()
```

RETURNS

0	false, not connected to a busbar
1	true, generator is connected to a busbar

Reconnect

Connects a static generator by closing all switches (breakers and isolators) up to the first breaker on the HV side of a transformer. The topological search to find all the switches, stops at any busbar.

```
int ElmGenstat.Reconnect()
```

RETURNS

0	the machine was successfully closed
1	a error occurred and the machine could not be connected to any busbar

ResetDerating

Resets the derating value, setting the Max. Active Power Rating according to the rating factor. The following formula is used: $Pmax\_uc = pmaxratf * Pn * ngnum$.

```
None ElmGenstat.ResetDerating()
```

4.2.16 ElmGndswt

Overview

Close
GetGroundingImpedance
IsClosed
IsOpen
Open

Close

Closes the switch by changing its status to 'close'. If closed, the connected node will be considered as being earthed.

```
int ElmGndswt.Close()
```

RETURNS

1, always

SEE ALSO

ElmGndswt.Open()

GetGroundingImpedance

Returns the impedance of the internal grounding. ElmGndswt is only considered to have an internal grounding if it is single phase and connected to neutral.

```
[int valid,
float resistance,
float reactance   ] ElmGndswt.GetGroundingImpedance(int busIdx)
```

ARGUMENTS

busIdx	Bus index where the grounding should be determined.
resistance (out)	Real part of the grounding impedance in Ohm.
reactance (out)	Imaginary part of the grounding impedance in Ohm.

RETURNS

0	The values are invalid (e.g. because there is no internal grounding)
1	The values are valid.

IsClosed

Returns information about current switch state.

```
int ElmGndswt.IsClosed()
```

RETURNS

1	switch is closed
0	switch is open

SEE ALSO

ElmGndswt.IsOpen()

IsOpen

Returns information about current switch state.

```
int ElmGndswt.IsOpen()
```

RETURNS

1	switch is open
0	switch is closed

SEE ALSO

ElmGndswt.IsClosed()

Open

Opens the switch by changing its status to 'open'.

```
int ElmGndswt.Open()
```

RETURNS

0, always

SEE ALSO

ElmGndswt.Close()

4.2.17 ElmLne

Overview

AreDistParamsPossible
CreateFeederWithRoutes
FitParams
GetIthr
GetType
GetY0m
GetY1m
GetZ0m
GetZ1m
GetZmatDist
HasRoutes
HasRoutesOrSec
IsCable
IsNetCoupling
MeasureLength
SetDetailed

AreDistParamsPossible

Check if the line fulfills conditions for the calculation of distributed parameters:

ElmLne	No routes, no sections
TypTow	only 1 circuit x 3 phases
TypGeo	only 1 circuit x 3 phases
TypLne	AC system, 3 phases and 0 neutral
TypCabsys	only 1 circuit x 3 phases

```
int ElmLne.AreDistParamsPossible()
```

RETURNS

The returned value are:

0	All conditions fulfilled
1	Line contains routes
2	Line contains sections
3	Line has no type
4	TypTow/TypCabsys does not fulfill conditions for distributed paramters
5	TypLne does not fulfill conditions for distributed parameters
6	Short-circuit flag is set (EMT or RMS simulations)
7	TypLne/TypTow: B0 and B1 = 0
8	Error, no condition state could be determined

CreateFeederWithRoutes

Creates a new feeder in the line by splitting the line into 2 routes and inserting a terminal.

```
int ElmLne.CreateFeederWithRoutes(float dis,
                                  float rem,
                                  DataObject O)
int ElmLne.CreateFeederWithRoutes(float dis,
                                  float rem,
                                  DataObject O,
                                  [int sw0,]
                                  [int sw1])
```

ARGUMENTS

dis	Inserting operation occurs after this distance
rem	Remaining distance, percentage of distance ’dis’
O	Branch object that is to be connected at the inserted terminal
sw0	If set to (1), switch is inserted on the first side
sw1	If set to (1), switch is inserted on the second side

RETURNS

0	Success, feeders created
1	Error

FitParams

Calculates distributed parameters for line elements. Whether this function calculates constant parameters or frequency dependent parameters depends on the user setting of the parameter 'i_model' in the ElmLne dialogue. The settings are as follows: i_model=0: constant parameters; i_model=1: frequency dependent parameters.

```
int ElmLne.FitParams()
```

RETURNS

0	Success
1	Error

GetIthr

Returns the rated short-time current of the line element.

```
float ElmLne.GetIthr()
```

RETURNS

Returns rated short-time current value

GetType

Returns the line type object.

```
DataObject ElmLne.GetType()
```

RETURNS

The TypLne object if exists or None

GetY0m

The function returns the zero-sequence mutual coupling admittance (G0m, B0m) in Ohm of the line and input argument line (object Lne2). When Lne2 = line, the function returns the zero-sequence self admittance.

```
[int error,
float G0m,
float B0m ] ElmLne.GetY0m(DataObject Lne2)
```

ARGUMENTS

Lne2 Line element

G0m (out)
Resulting G0m value

B0m (out)
Resulting B0m value

RETURNS

0	Success, data obtained
1	Error, e.g. no coupling objects defined

GetY1m

The function returns the positive-sequence mutual coupling admittance (G1m, B1m) in Ohm of the line and input argument line (object Lne2). When Lne2 = line, the function returns the positive-sequence self admittance.

```
[int error,
float G1m,
float B1m ] ElmLne.GetY1m (DataObject Lne2)
```

ARGUMENTS

Lne2 Line element

G1m (out)
Resulting G1m value

B1m (out)
Resulting B1m value

RETURNS

0	Success, data obtained
1	Error, e.g. no coupling objects defined

GetZ0m

Gets the zero-sequence mutual coupling impedance (R0m, X0m) in Ohm of the line and input argument line (object otherLine). When otherLine = line, the function returns the zero-sequence self impedance.

```
[int error,
float R0m,
float X0m ] ElmLne.GetZ0m(DataObject otherLine)
```

ARGUMENTS

otherLine Line element

R0m (out)
To be obtained R0m value

X0m (out)
To be obtained X0m value

RETURNS

0	Success, data obtained
1	Error, e.g. no coupling objects defined

GetZ1m

The function returns the positive-sequence mutual coupling impedance (R1m, X1m) in Ohm of the line and input argument line (object Lne2). When Lne2 = line, the function returns the positive-sequence self impedance.

```
[int error,
float R1m,
float X1m ] ElmLne.GetZ1m(DataObject Lne2)
```

ARGUMENTS

Lne2	Line element
R1m (out)	Resulting R1m value
X1m (out)	Resulting X1m value

RETURNS

0	Success, data obtained
1	Error, e.g. no coupling objects defined

GetZmatDist

The function gets impedance matrix in phase domain (only amplitudes), for a line with distributed parameters, short-circuit ended.

```
int ElmLne.GetZmatDist(float frequency,
                       int exact,
                       DataObject matrix)
```

ARGUMENTS

frequency	Frequency for which the calculation is carried out
exact	0: Approximated solution, 1: Exact solution for ‘frequency’
matrix	Impedance matrix to be filled with the impedance amplitudes

RETURNS

The returned value reports if the impedance matrix acquired:

1	Error, no matrix acquired
0	Success, matrix acquired

HasRoutes

Checks if the line is subdivided into routes.

```
int ElmLne.HasRoutes()
```

RETURNS

0	When the line is a single line
1	When the line is subdivided into routes

HasRoutesOrSec

Checks if the line is subdivided into routes or sections.

```
int ElmLne.HasRoutesOrSec()
```

RETURNS

0	When the line is a single line
1	When the line is subdivided into routes
2	When the line is subdivided into sections

IsCable

Checks if this line is a cable.

```
int ElmLne.IsCable()
```

RETURNS

1	Line is a cable
0	Line is not a cable

IsNetCoupling

Checks if the line connects two grids.

```
int ElmLne.IsNetCoupling()
```

RETURNS

The returned value reports if the line is a coupler:

1	The line is a coupler (connects two grids)
0	The line is not a coupler

MeasureLength

Measures the length of this line using the active diagram. For graphical measurement the active diagram needs to have a scaling factor. Geographic diagrams by default have a scaling factor. If iUseGraphic = 1, the line length is determined directly from the positions given in (latitude/longitude) considering the earth as a perfect sphere. In this case no graphic needs to be open.

```
float ElmLne.MeasureLength([int iUseGraphic])
```

ARGUMENTS

iUseGraphic (optional)
Use SGL diagram for calculation or not.

1	Use displayed diagram for calculation (default)
0	Calculate distance without diagram

RETURNS

≥ 0	Returns the graphical length of this line in its current unit
< 0	Error: E.g. when line is not represented in the active diagram and iUseGraphic=1

SetDetailed

The function can be used to prevent the automatically reduction of a line e.g. if the line is a line dropper (length = 0). The function should be called when no calculation method is valid (before first load flow). The internal flag is automatically reset after the first calculation is executed.

```
int ElmLne.SetDetailed()
```

4.2.18 ElmLnesec

Overview

IsCable

IsCable

Checks if this line section is a cable.

```
int ElmLnesec.IsCable()
```

RETURNS

1	Line section is a cable
0	Line section is not a cable
-1	Error

4.2.19 ElmNec

Overview

GetGroundingImpedance

GetGroundingImpedance

Returns the impedance of the internal grounding.

```
[int valid,
float resistance,
float reactance  ] ElmNec.GetGroundingImpedance(int busIdx)
```

ARGUMENTS

busIdx Bus index where the grounding should be determined.

resistance (out)
Real part of the grounding impedance in Ohm.

reactance (out)
Imaginary part of the grounding impedance in Ohm.

RETURNS

0	The values are invalid (e.g. because there is no internal grounding)
1	The values are valid.

4.2.20 ElmNet

Overview

Activate
CalcBoundary
CalculateInterchangeTo
Deactivate

Activate

Adds a grid to the active study case. Can only be applied if there are is no currently active calculation (i.e. running contingency analysis).

```
int ElmNet.Activate()
```

RETURNS

0	on success
1	on error

CalcBoundary

Defines boundary with this grid as exterior part. Resulting cubicles of boundary are branch-oriented away from the grid.

```
[int error,
DataObject boundary] ElmNet.CalcBoundary(float shift)
```

ARGUMENTS

shift elements that are within a distance of shift many elements to a boundary cubicle of the grid are added to the exterior part of the resulting boundary

boundary (out)
defined boundary

RETURNS

0	successful call, boundary defined
1	error during determination of boundary cubicles

CalculateInterchangeTo

This function calculates the power flow from current grid to a connected grid. The values are stored in current grid in the following attributes (values from the previous load flow calculation are overwritten):

- Pinter: Active Power Flow
- Qinter: Reactive Power Flow
- ExportP: Export Active Power Flow
- ExportQ: Export Reactive Power Flow
- ImportP: Import Active Power Flow

- ImportQ: Import Reactive Power Flow

```
int ElmNet.CalculateInterchangeTo(DataObject net)
```

ARGUMENTS

net	Connected grid

RETURNS

$<$ **0**	error
$=$ **0**	grids are not connected, no interchange exists
$>$ **0**	ok

Deactivate

Removes a grid from the active study case.Can only be applied if there are is no currently active calculation.

```
int ElmNet.Deactivate()
```

RETURNS

0	on success
1	on error

4.2.21 ElmPvsys

Overview

Derate
Disconnect
GetAvailableGenPower
GetGroundingImpedance
IsConnected
Reconnect
ResetDerating

Derate

Derates the value of the Max. Active Power Rating according to the specified value given in MW.
The following formula is used: $Pmax\_uc = Pmax\_uc - "Deratingvalue"$.

```
None ElmPvsys.Derate(float deratingP)
```

ARGUMENTS

deratingP	Derating value

Disconnect

Disconnects a PV system by opening the first circuit breaker. The topological search performed to find such a breaker, stops at any busbar.

```
int ElmPvsys.Disconnect()
```

RETURNS

0	breaker already open or successfully opened
1	an error occurred (no breaker found, open action not possible (earthing / RA))

GetAvailableGenPower

Returns the available power that can be dispatched from the generator, for the particular study time.
For the case of conventional generators (no wind generation selected), the available power is equal to the nominal power specified.
For wind generators, the available power will depend on the wind model specified:

- No Wind Model: No available power.
- Stochastic Wind Model: Given the specified mean wind speed, the available power is calculated from the Power Curve. If the units of the Power Curve are in MW, the returned value is directly the available power. In the other hand, if the units are in PU, the returned value is multiplied by the nominal power of the generator to return the available power.
- Time Series Characteristics of Active Power Contribution: The available power is the average of the power values (in MW) obtained from all the specified time characteristics for the current study time.
- Time Series Characteristics of Wind Speed: The available power is calculated with the average of the power values (in MW) calculated for all the specified time characteristics. A power value for any time characteristic is calculated by obtaining the wind speed for the current study time, and then calculating the power from the specified Power Curve. If the units of the Power Curve are in MW, the returned value is directly the power value. In the other hand, if the units are in PU, the returned value is multiplied by the nominal power of the generator to return the power value.

For motors, the available power is zero.

```
float ElmPvsys.GetAvailableGenPower()
```

RETURNS

Available generation power

GetGroundingImpedance

Returns the impedance of the internal grounding.

```
[int valid,
float resistance,
float reactance  ] ElmPvsys.GetGroundingImpedance(int busIdx)
```

ARGUMENTS

busIdx Bus index where the grounding should be determined.

resistance (out)
Real part of the grounding impedance in Ohm.

reactance (out)
Imaginary part of the grounding impedance in Ohm.

RETURNS

0	The values are invalid (e.g. because there is no internal grounding)
1	The values are valid.

IsConnected

Checks if a PV system is already connected to any busbar.

```
int ElmPvsys.IsConnected()
```

RETURNS

0	false, not connected to a busbar
1	true, generator is connected to a busbar

Reconnect

Connects a PV system by closing all switches (breakers and isolators) up to the first breaker on the HV side of a transformer. The topological search to find all the switches, stops at any busbar.

```
int ElmPvsys.Reconnect()
```

RETURNS

0	the machine was successfully closed
1	a error occurred and the machine could not be connected to any busbar

ResetDerating

Resets the derating value, setting the Max. Active Power Rating according to the rating factor. The following formula is used: $Pmax\_uc = pmaxratf * Pn * ngnum$.

```
None ElmPvsys.ResetDerating()
```

4.2.22 ElmRelay

Overview

CheckRanges
GetCalcRX
GetMaxFdetectCalcI
GetSlot
GetUnom
IsStarted
SetImpedance
SetMaxI
SetMaxIearth
SetMinI
SetMinIearth
SetOutOfService
SetTime
SlotUpdate

CheckRanges

Checks the settings of all elements in the relay for range violations.

```
int ElmRelay.CheckRanges()
```

RETURNS

0	All settings are valid.
1	At least one setting was forced into range.
-1	An error occurred.

GetCalcRX

Gets the calculated impedance from the polarising unit.

```
[int error,
float real,
float imag ] ElmRelay.GetCalcRX(int inSec,
                                int unit)
```

ARGUMENTS

inSec

0	Get the value in pri. Ohm.
1	Get the value in sec. Ohm.

unit

0	Get the value from Phase-Phase or Multifunctional polarizing.
1	Get the value from Phase-Earth or Multifunctional polarizing.
2	Get the value from Multifunctional polarizing

real (out) Real part of the impedance in Ohm.

imag (out)
Imaginary part of the impedance in Ohm.

RETURNS

0	No error occurred, the output is valid.
1	An error occurred, the output is invalid.

GetMaxFdetectCalcI

Get the current measured by the starting unit.

```
[int error,
float Iabs ] ElmRelay.GetMaxFdetectCalcI(int earth,
                                         int unit
                                         )
```

ARGUMENTS

Iabs (out) The measured current in A

earth

0	Get the phase current.

1	Get the earth current.

unit

0	Get the current in pri. A.
1	Get the current in sec. A.

RETURNS

0	No error, output is valid.
1	An error occourred, the output is invalid.

GetSlot

Returns the element in the slot with the given name.

```
DataObject ElmRelay.GetSlot(str name,
                            [int iShowErr]
                            )
```

ARGUMENTS

name Exact name of the slot to search for (no wildcards).

iShowErr (optional)

0	Do not show error messages.
1	Show error messages if a slot is not found or empty.

RETURNS

The object in the slot or None.

GetUnom

Returns the nominal voltage of the local bus of the relay.

```
float ElmRelay.GetUnom()
```

RETURNS

The nominal voltage of the local bus of the relay in kV.

IsStarted

Checks if the starting unit detected a fault.

```
int ElmRelay.IsStarted()
```

RETURNS

0	No fault was detected.
1	Fault was detected.
-1	An error occourred.

SetImpedance

Sets the the given impedance to the distance blocks matching the criteria.

```
int ElmRelay.SetImpedance(float real,
                          float imag,
                          int inSec,
                          int zone,
                          int unit
                          )
int ElmRelay.SetImpedance(float real,
                          float imag,
                          float lineAngle,
                          float Rarc,
                          int inSec,
                          int zone,
                          int unit
                          )
```

ARGUMENTS

real	Real part of the impedance in Ohm.
imag	Imaginary part of the impedance in Ohm.
inSec	

0	The values are in pri. Ohm.
1	The values are in sec. Ohm.

zone	Set the impedance for elments with this zone number.
unit	

0	Set the impedance for Phase - Phase or Multifunctional elements.
1	Set the impedance for Phase - Earth or Multifunctional elements.
2	Set the impedance for Multifunctional elements.

ARGUMENTS

real	Real part of the impedance in Ohm.
imag	Imaginary part of the impedance in Ohm.
lineAngle	The line angle in deg.
Rarc	The arc resistance in Ohm.
inSec	

0	The values are in pri. Ohm.
1	The values are in sec. Ohm.

zone	Set the impedance for elments with this zone number.
unit	

0	Set the impedance for Phase - Phase or Multifunctional elements.
1	Set the impedance for Phase - Earth or Multifunctional elements.
2	Set the impedance for Multifunctional elements.

RETURNS

0	No error occurred.
1	An error occurred or no element was found.

SetMaxI

Sets the “Max. Phase Fault Current” of the relay to the currently measured value.

```
None ElmRelay.SetMaxI()
```

SetMaxIearth

Sets the “Max. Earth Fault Current” of the relay to the currently measured value.

```
None ElmRelay.SetMaxIearth()
```

SetMinI

Sets the “Min. Phase Fault Current” of the relay to the currently measured value.

```
None ElmRelay.SetMinI()
```

SetMinIearth

Sets the “Min. Earth Fault Current” of the relay to the currently measured value.

```
None ElmRelay.SetMinIearth()
```

SetOutOfService

Sets the “Out of Service” flag of elements contained in the relay.

```
int ElmRelay.SetOutOfService(int outServ,
                             int type,
                             int zone
                             int unit
                             )
```

ARGUMENTS

outServ

0	Set elements in service.
1	Set Elements out of service.

type

1	Set the flag for overcurrent elements.
2	Set the flag for distance elements.

zone Set the flag for elments with this zone number (only when settings distance elements).

unit

0	Set the flag for Phase-Phase or Multifunctional elements.
1	Set the flag for Phase-Earth or Multifunctional elements.
2	Set the flag for Multifunctional elements.

RETURNS

0	No error occurred.
1	An error occurred or no element was found.

SetTime

Sets the tripping time for elements contained in the relay.

```
int ElmRelay.SetTime(float time,
                     int type,
                     int zone,
                     int unit
                     )
```

ARGUMENTS

time Time in s.

type

1	Set the time for overcurrent elements.
2	Set the time for distance elements.

zone Set the time for elments with this zone number (only when settings distance elements).

unit

0	Set the time for Phase-Phase or Multifunctional elements.
1	Set the time for Phase-Earth or Multifunctional elements.
2	Set the time for Multifunctional elements.

RETURNS

0	No error occurred.
1	An error occurred or no element was found.

SlotUpdate

Triggers a slot update of the relay.

```
None ElmRelay.SlotUpdate()
```

DEPRECATED NAMES

slotupd

4.2.23 ElmRes

Overview

AddVariable

Adds a variable to the list of monitored variables for the Result object.

```
None ElmRes.AddVariable(DataObject element,
                        str varname)
```

ARGUMENTS

element	An object.
varname	Variable name for object O.

DEPRECATED NAMES

AddVars

Clear

Clears all data (calculation results) written to the result file. The Variable definitions stored in the contents of ElmRes are not modified.

```
int ElmRes.Clear()
```

RETURNS

Always 0 and can be ignored.

FindColumn

Returns the index of the first header column matching the given object and/or variable name.

```
int ElmRes.FindColumn(DataObject obj,
                      [str varName]
                      )
int ElmRes.FindColumn(DataObject obj,
                      [int startCol]
                      )
int ElmRes.FindColumn(DataObject obj,
                      str varName
                      )
```

ARGUMENTS

obj (optional)
Object of matching column

varName (optional)
Variable name of matching column

startCol (optional)
Index of first checked column; Search starts at first column if colIndex is not given

RETURNS

≥ 0	column index
< 0	no valid column found

The index can be used in the ElmRes method GetData to retrieve the value of the column.

FindMaxInColumn

Find the maximum value of the variable in the given column.

```
[int row,
float value] ElmRes.FindMaxInColumn(int column)
```

ARGUMENTS

column The column index.

value (optional, out)
The maximum value found. The value is 0. in case that the maximum value was not found.

RETURNS

< 0	The maximum value of column was not found.
≥ 0	The row with the maximum value of the column.

FindMaxOfVariableInRow

Find the maximum value for the given row and variable.

```
[int col,
float maxValue] ElmRes.FindMaxOfVariableInRow(str variable,
                                              int row)
```

ARGUMENTS

variable The variable name

variable The row

maxValue (optional)
The corresponding maximum value.

RETURNS

< 0	There is no valid value of the corresponding variable in the row.
≥ 0	Column index of variable.

FindMinInColumn

Find the minimum value of the variable in the given column.

```
[int row,
float value] ElmRes.FindMinInColumn(int column)
```

ARGUMENTS

column The column index.

value (optional, out)
The minimum value found. The value is 0. in case that the minimum value was not found.

RETURNS

< 0	The minimum value of column was not found.
≥ 0	The row with the minimum value of the column.

FindMinOfVariableInRow

Find the minimum value for the given row and variable.

```
[int col,
float minValue] ElmRes.FindMinOfVariableInRow(str variable,
                                              int row)
```

ARGUMENTS

variable	The variable name
variable	The row
minValue (optional, out)	The corresponding minimum value.

RETURNS

< 0	There is no valid value of the corresponding variable in the row.
≥ 0	Column index of variable.

FinishWriting

Finishes the writing of values to a result file.

```
None ElmRes.FinishWriting()
```

DEPRECATED NAMES

Close

SEE ALSO

ElmRes.InitialiseWriting(), ElmRes.Write(), ElmRes.WriteDraw()

Flush

This function is required in scripts which perform both file writing and reading operations. While writing to a results object (ElmRes), a small portion of this data is buffered in memory. This is required for performance reasons. Therefore, all data must be written to the disk before attempting to read the file. 'Flush' copies all data buffered in memory to the disk. After calling 'Flush'all data is available to be read from the file.

```
int ElmRes.Flush()
```

GetDescription

Get the description of a column.

```
str ElmRes.GetDescription([int column],
                          [int short]
                          )
```

ARGUMENTS

column (optional)
The column index. The description name of the default variable is returned if the parameter is nor passed to the function.

short (optional)

0	long desc. (default)
1	short description

RETURNS

Returns the description which is empty in case that the column index is not part of the data.

GetFirstValidObject

Gets the index of the column for the first valid variable in the given line. Starts at the beginning of the given line and sets the internal iterator of the result file to the found position.

```
int ElmRes.GetFirstValidObject(int row,
                               [str classNames]
                               [str variableName,]
                               [float limit,]
                               [int limitOperator,]
                               [float limit2,]
                               [int limitOperator2]
                               )
int ElmRes.GetFirstValidObject(int row,
                               list objects
                               )
```

ARGUMENTS

row Result file row

classNames (optional)
Comma separated list of class names for valid objects. The next object of one of the given classes is searched. If not set all objects are considered as valid (default).

variableName (optional)
Name of the limiting variable. The searched object must have this variable. If not set variables are not considered (default).

limit (optional)
Limiting value for the variable.

limitOperator (optional)
Operator for checking the limiting value:

0	all values are valid (default)
1	valid values must be $<$ limit
2	valid values must be $\leq$ limit
3	valid values must be $>$ limit
4	valid values must be $\geq$ limit

limit2 (optional)
Second limiting value for the variable.

limitOperator2 (optional)
Operator for checking the second limiting value:

< 0	first OR second criterion must match,
> 0	first AND second criterion must match,
0	all values are valid (default)
1/-1	valid values must be $<$ limit2
2/-2	valid values must be $\leq$ limit2

3/-3	valid values must be $>$ limit2
4/-4	valid values must be $\geq$ limit2

objects Valid objects

RETURNS

≥ 0	column index
< 0	no valid column found

GetFirstValidObjectVariable

Gets the index of the first valid variable of the current object in the current line. Starts at the internal iterator of the given result file and sets it to the position found.

```
int ElmRes.GetFirstValidObjectVariable([str variableNames])
```

ARGUMENTS

variableNames (optional)
Comma separated list of valid variable names. The next column with one of the given variables is searched. If empty all variables of the current object are considered as valid (default).

RETURNS

≥ 0	column index
< 0	no valid column found

GetFirstValidVariable

Gets the index of the column for the first valid variable in the given line. Starts at the beginning of the given line and sets the internal iterator of the result file to the found position.

```
int ElmRes.GetFirstValidVariable(int row,
                                  [str variableNames]
                                  )
```

ARGUMENTS

row Result file row

variableNames (optional)
Comma separated list of valid variable names. The next column with one of the given variables is searched. If not set all variables are considered as valid (default).

RETURNS

≥ 0	column index
< 0	no valid column found

GetNextValidObject

Gets the index of the column for the next valid variable (after current iterator) in the given line. Sets the internal iterator of the result file to the position found.

```
int ElmRes.GetNextValidObject([str classNames]
                              [str variableName,]
                              [float limit,]
                              [int limitOperator,]
                              [float limit2,]
                              [int limitOperator2]
                              )
int ElmRes.GetNextValidObject(list objects)
```

ARGUMENTS

row Result file row

classNames (optional)
Comma separated list of class names for valid objects. The next object of one of the given classes is searched. If not set all objects are considered as valid (default).

variableName (optional)
Name of the limiting variable. The searched object must have this variable. If not set variables are not considered (default).

limit (optional)
Limiting value for the variable.

limitOperator (optional)
Operator for checking the limiting value:

0	all values are valid (default)
1	valid values must be $<$ limit
2	valid values must be $\leq$ limit
3	valid values must be $>$ limit
4	valid values must be $\geq$ limit

limit2 (optional)
Second limiting value for the variable.

limitOperator2 (optional)
Operator for checking the second limiting value:

< 0	first OR second criterion must match,
> 0	first AND second criterion must match,
0	all values are valid (default)
1/-1	valid values must be $<$ limit2
2/-2	valid values must be $\leq$ limit2
3/-3	valid values must be $>$ limit2
4/-4	valid values must be $\geq$ limit2

objects Valid objects

RETURNS

≥ 0	column index
< 0	no valid column found

GetNextValidObjectVariable

Gets the index of the column for the next valid variable of the current object in the current line. Starts at the internal iterator of the given result file and sets it to the found position.

```
int ElmRes.GetNextValidObjectVariable([str variableNames])
```

ARGUMENTS

variableNames (optional)
Comma separated list of valid variable names. The next column with one of the given variables is searched. If not set all variables are considered as valid (default).

RETURNS

≥ 0	column index
< 0	no valid column found

GetNextValidVariable

Gets the index of the column for the next valid variable in the given line. Starts at the internal iterator of the given line and sets the internal iterator of the result file to the found position.

```
int ElmRes.GetNextValidVariable([str variableNames])
```

ARGUMENTS

variableNames (optional)
Comma separated list of valid variable names. The next column with one of the given variables is searched. If not set all variables are considered as valid (default).

RETURNS

≥ 0	column index
< 0	no valid column found

GetNumberOfColumns

Returns the number of variables (columns) in result file excluding the default variable (e.g. time for time domain simulation).

```
int ElmRes.GetNumberOfColumns()
```

RETURNS

Number of variables (columns) in result file.

GetNumberOfRows

Returns the number of values per column (rows) stored in result object.

```
int ElmRes.GetNumberOfRows()
```

RETURNS

Returns the number of values per column stored in result object.

GetObj

Returns an object used in the result file. Positive index means objects for which parameters are being monitored (i.e. column objects). Negative index means objects which occur in written result rows as values.

```
DataObject ElmRes.GetObj(int index)
```

ARGUMENTS

index	index of the object.

RETURNS

The object found or None.

GetObject

Get object of given column.

```
DataObject ElmRes.GetObject([int column])
```

ARGUMENTS

col	Column index. Object of default column is returned if col is not passed.

RETURNS

The object of the variable stored in column ’column’.

GetRelCase

Get the contingency object for the given case number from the reliability result file.

```
DataObject ElmRes.GetRelCase(int caseNumber)
```

ARGUMENTS

caseNumber	The reliability case number

RETURNS

Returns the contingency of case number. None is returned if there is no corresponding contingency.

GetSubElmRes

Get sub-result file stored inside this.

```
DataObject ElmRes.GetSubElmRes(int value)
DataObject ElmRes.GetSubElmRes(DataObject obj)
```

ARGUMENTS

value	The cnttime to look for
obj	The pResElm to look for

RETURNS

None	The sub result file with value=cnttime (obj=pResElm) was not found.
any other value	The sub result file with value=cnttime (obj=pResElm).

GetUnit

Get the unit of a column.

```
str ElmRes.GetUnit([int column])
```

ARGUMENTS

column (optional)
The column index. The unit of the default variable is returned if the parameter is nor passed to the function.

RETURNS

Returns the unit which is empty in case that the column index is not part of the data.

GetValue

Returns a value from a result object for row iX of curve col.

```
[int error,
float d   ] ElmRes.GetValue(int iX,
                            [int col])
```

ARGUMENTS

d (out)	The value retrieved from the data.
iX	The row.
col (optional)	The curve number, which equals the variable or column number, first column value (time,index, etc.) is returned when omitted.

RETURNS

0	when ok
1	when iX out of bound
2	when col out of bound
3	when invalid value is returned from a sparse file. Sparse files are written e.g. by the contingency, the value is invalid in case that it was not written, bcause it was below the recording limit. Result files created using DPL/Python are always full and will not return invalid values.

GetVariable

Get variable name of column

```
str ElmRes.GetVariable([int column])
```

ARGUMENTS

column (optional)
The column index. The variable name of the default variable is returned if the parameter is nor passed to the function.

RETURNS

Returns the variable name which is empty in case that the column index is not part of the data.

InitialiseWriting

Opens the result object for writing. This function must be called before writing data for result files not stored in the script object. If arguments are passed to the function they specify the variable name, unit... of the default variable (e.g. to be used by plots as x-axis).

```
int ElmRes.InitialiseWriting()
int ElmRes.InitialiseWriting(str variableName,
                             str unit,
                             str description,
                             [str shortDescription]
                             )
```

ARGUMENTS

variableName
The variable name for the default variable (e.g. “distance”)

unit The unit (e.g. “km”)

description
The description of the variable (e.g. “Distance from infeed”)

shortDescription
The short description (e.g. “Dist. Infeed”)

RETURNS

Always 0 and can be ignored

DEPRECATED NAMES

Init

SEE ALSO

ElmRes.FinishWriting(), ElmRes.Write(), ElmRes.WriteDraw()

Load

Loads the data of a result object (**ElmRes**) in memory for reading.

```
None ElmRes.Load()
```

Release

Releases the data loaded to memory. This function should be used whenever several result objects are processed in a loop. Data is always released from memory automatically after execution of the current script.

```
None ElmRes.Release()
```

SetAsDefault

Sets this results object as the default results object. Plots using the default result file will use this file for displaying data.

```
None ElmRes.SetAsDefault()
```

SetObj

Adds an object to the objects assigned to the result file

```
int ElmRes.SetObj(DataObject element)
```

Arguments

element	Element to store in result file

Returns

The index which can be used to retrieve the object from the results file. The index is <0 if no results are recorded for the given object (e.g. a contingency in reliability calculation). The index is $\geq$ if variables are recorded for the object.

SetSubElmResKey

Assigns a value or an object to the according ElmRes parameter.

```
None ElmRes.SetSubElmResKey(int value)
None ElmRes.SetSubElmResKey(DataObject obj)
```

Arguments

value	Value to be assigned to parameter cnttime of ElmRes
value	Object to be assigned to parameter pResElm of ElmRes

SortAccordingToColumn

Sorts all rows in the data loaded according to the given column. The ElmRes itself remains unchanged.

```
int ElmRes.SortAccordingToColumn(int column)
```

Arguments

col	The column number.

Returns

0	The function executed correctly, the data was sorted correctly according to the given column.
1	The column with index column does not exist.

Write

Writes the current results to the result object.

```
int ElmRes.Write([float defaultValue])
```

Returns

0 on success

See also

ElmRes.WriteDraw(), ElmRes.InitialiseWriting(), ElmRes.FinishWriting()

WriteDraw

Writes current results to the result objects and updates all plots that display values from the result object.

```
int ElmRes.WriteDraw()
```

Returns

0 on success

4.2.24 ElmShnt

Overview

GetGroundingImpedance

GetGroundingImpedance

Returns the impedance of the internal grounding. Single phase shunts connected to neutral are considered as grounding devices themselves; i.e. instead of the dedicated grounding parameters, the shunt parameters are used.

```
[int valid,
float resistance,
float reactance  ] ElmShnt.GetGroundingImpedance(int busIdx)
```

Arguments

busIdx Bus index where the grounding should be determined.

resistance (out)
Real part of the grounding impedance in Ohm.

reactance (out)
Imaginary part of the grounding impedance in Ohm.

RETURNS

0	The values are invalid (e.g. because there is no internal grounding)
1	The values are valid.

4.2.25 ElmStactrl

Overview

GetControlledHVNode
GetControlledLVNode
GetStepupTransformer
Info

GetControlledHVNode

Returns the corresponding voltage controlled HV node for the machine at the specified index. Switch status are always considered.

```
DataObject ElmStactrl.GetControlledHVNode(int index)
```

ARGUMENTS

index	Index of machine (starting from 0 – . . .).

RETURNS

object	Busbar/Terminal ()
None	not found

GetControlledLVNode

Returns the corresponding voltage controlled LV node for the machine at specified index. Switch status are always considered.

```
DataObject ElmStactrl.GetControlledLVNode(int index)
```

ARGUMENTS

index	Index of machine (starting from 0 – . . .).

RETURNS

object	Terminal ()
None	not found

GetStepupTransformer

Performs a topological search to find the step-up transformer of the machine at the specified index.

```
DataObject ElmStactrl.GetStepupTransformer([int index,]
                                           [int iBrkMode]
                                           )
```

ARGUMENTS

index Index of machine (starting from 0 – . . .).

iBrkMode (optional)

0 (default)	All switch status (open,close) are considered
1	Ignore breaker status (jump over open breakers)
2	Ignore all switch status (jump over open switches)

RETURNS

object	step-up transformer
None	step-up transformer not found

Info

Prints the control information in the output window. It is the same information that the button ”Info” of the Station Control dialog prints.

```
int ElmStactrl.Info()
```

4.2.26 ElmSubstat

Overview

ApplyAndResetRA
GetSplit
GetSplitCal
GetSplitIndex
GetSuppliedElements
OverwriteRA
ResetRA
SaveAsRA
SetRA

ApplyAndResetRA

This function applies switch statuses of currently selected running arrangement to corresponding switches and resets the running arrangement selection afterwards. Nothing happens if no running arrangement is selected.

```
int ElmSubstat.ApplyAndResetRA()
```

RETURNS

1	on success
0	otherwise, especially if no running arrangement is selected

GetSplit

A split of a station is a group of topologically connected elements. Such a group is called split if all contained components are energized and there is at least one busbar (terminal of usage 'busbar') contained or it has connections to at least two main components (= all components

except switch devices and terminals).
These splits are ordered according to the count of nodes contained and according to their priority. So each split becomes a unique index.

The function GetSplit offers access to the elements contained in a split. By calling GetSplit with an index from 0 to n, the elements belonging to the corresponding split are filled into given sets and returned.

```
[int error
list mainNodes,
list connectionCubicles,
list allElements        ] ElmSubstat.GetSplit(int index)
```

ARGUMENTS

index Index of the split used to access the elements of the corresponding split. Value must be ≥ 0.

mainNodes (out)
Terminals of same usage considered to form the most important nodes for that group. In most cases, this is the group of contained busbars.

connectionCubicles (optional, out)
All cubicles (of terminals inside the station) that point to an element that sits outside the station or to an element that is connected to a terminal outside the station are filled into the set connectionCubicles. (The connection element (branch) can be accessed by calling GetBranch() on each of these cubicles. The terminals of these cubicles (parents) must not necessarily be contained in any split. They could also be separated by a disconnecting component.)

allElements(optional, out)
All elements (class Elm*) of the split that have no connection to elements outside the station are filled into this set.

RETURNS

0 success, split of that index exists and is returned.

1 indicates that there exists no split with given index. (Moreover, this means that there is no split with index n greater than this value.)

SEE ALSO

ElmSubstat.GetSplitCal(), ElmSubstat.GetSplitIndex(),

GetSplitCal

This function determines the elements that belong to a split. In contrast to ElmSubstat.GetSplit() it is based on calculation instead of pure edit object topology. This means the returned nodes correspond to the calculation nodes, the interconnecting cubicles are those connecting nodes of different splits.
Note: As this function relies on calculation nodes it can only be executed after a calculation has been performed (e.g. load flow calculation).

```
[int error,
list nodes,
list connectionCubicles,
list elements           ] ElmSubstat.GetSplitCal(int index)
```

ARGUMENTS

index Index of the split used to access the elements of the corresponding split. Refers to same split as index in ElmSubstat.GetSplit().
Value must be ≥ 0.

nodes (out)
A set that is filled with terminals. There is one terminal returned for each calculation node in the split.

connectionCubicles (optional, out)
This set is filled with all cubicles that point from a calculation node of current split to another calculation node that does not belong to that split. The connecting element can be accessed by calling GetBranch() on such a cubicle.

elements (optional, out)
This set is filled with network elements that are connected to a calculation node of current split and have exactly one connection, i.e. these elements are completely contained in the split.

RETURNS

0	success, split of that index exists and is returned.
1	indicates that there exists no split with given index. (Moreover, this means that there is no split with index n greater than this value.)

SEE ALSO

ElmSubstat.GetSplit()

GetSplitIndex

This function returns the index of the split that contains passed object.

```
int ElmSubstat.GetSplitIndex(DataObject o)
```

ARGUMENTS

o Object for which the split index is to be determined.

RETURNS

$\geq$ **0**	index of split in which element is contained
-1	given object does not belong to any split of that station

SEE ALSO

ElmSubstat.GetSplit()

GetSuppliedElements

Returns all network components that are supplied by the transformers located in the station.

```
list ElmSubstat.GetSuppliedElements([int inclNested])
```

ARGUMENTS

inclNested (optional)

0	Do not include components that are supplied by nested supplying stations
1	(default) Include components that are supplied by nested stations

SEE ALSO

ElmTr2.GetSuppliedElements(), ElmTr3.GetSuppliedElements()

OverwriteRA

This function overwrites switch statuses stored in an existing running arrangement with actual switch statuses of the substation. This is only possible if the substation has no running arrangement selected and given running arrangement is valid for substation the method was called on.

```
int ElmSubstat.OverwriteRA(DataObject ra)
```

ARGUMENTS

ra	Given running arrangement

RETURNS

1	If given running arrangement was successfully overwritten;
0	otherwise

ResetRA

This function resets the running arrangement selection for the substation it was called on.

```
None ElmSubstat.ResetRA()
```

SaveAsRA

When called on a substation that has no running arrangement selected, a new running arrangement is created and all switch statuses of all running arrangement relevant switches (for that substation) are saved in it. The running arrangement is stored in project folder Running Arrangement" and its name is set to given locname. The new running arrangement is not selected automatically.

(No new running arrangement is created if this method is called on a substation that has currently a running arrangement selected).

```
DataObject ElmSubstat.SaveAsRA(str locname)
```

ARGUMENTS

locname	Name of the new running arrangement (if name is already used, an increment (postfix) is added to make it unique).

RETURNS

Newly created 'IntRunarrange' object on success, otherwise None.

SetRA

This function sets the running arrangement selection for the substation it was called on. The switch statuses are now determined by the values stored in the running arrangement.

```
int ElmSubstat.SetRA(DataObject ra)
```

ARGUMENTS

ra	running arrangement that is valid for the substation

RETURNS

1	If given running arrangement was successfully set;
0	otherwise (e.g. given ra is not valid for that substation)

4.2.27 ElmSvs

Overview

GetStepupTransformer

GetStepupTransformer

Performs a topological search to find the step-up transformer of the static VAR system.

```
DataObject ElmSvs.GetStepupTransformer(float voltage,
                                       int swStatus
                                       )
```

ARGUMENTS

voltage	voltage level at which the search will stop
swStatus	consideration of switch status. Possible values are:

0	consider all switch status
1	ignore breaker status
2	ignore all switch status

RETURNS

Returns the first collected step-up transformer object. It is empty if not found (e.g. start terminal already at hvVoltage).

4.2.28 ElmSym

Overview

Derate
Disconnect
GetAvailableGenPower
GetGroundingImpedance
GetMotorStartingFlag
GetStepupTransformer
IsConnected
Reconnect
ResetDerating

Derate

Derates the value of the Max. Active Power Rating according to the specified value given in MW.
The following formula is used: $Pmax\_uc = Pmax\_uc - "Deratingvalue"$.

```
None ElmSym.Derate(float deratingP)
```

ARGUMENTS

deratingP Derating value

Disconnect

Disconnects a synchronous machine by opening the first circuit breaker. The topological search performed to find such a breaker, stops at any busbar.

```
int ElmSym.Disconnect()
```

RETURNS

0	breaker already open or successfully opened
1	an error occurred (no breaker found, open action not possible (earthing / RA))

GetAvailableGenPower

Returns the available power that can be dispatched from the generator, for the particular study time.
For the case of conventional generators (no wind generation selected), the available power is equal to the nominal power specified.
For wind generators, the available power will depend on the wind model specified:

- No Wind Model: No available power.
- Stochastic Wind Model: Given the specified mean wind speed, the available power is calculated from the Power Curve. If the units of the Power Curve are in MW, the returned value is directly the available power. In the other hand, if the units are in PU, the returned value is multiplied by the nominal power of the generator to return the available power.
- Time Series Characteristics of Active Power Contribution: The available power is the average of the power values (in MW) obtained from all the specified time characteristics for the current study time.
- Time Series Characteristics of Wind Speed: The available power is calculated with the average of the power values (in MW) calculated for all the specified time characteristics. A power value for any time characteristic is calculated by obtaining the wind speed for the current study time, and then calculating the power from the specified Power Curve. If the units of the Power Curve are in MW, the returned value is directly the power value. In the other hand, if the units are in PU, the returned value is multiplied by the nominal power of the generator to return the power value.

For motors, the available power is zero.

```
float ElmSym.GetAvailableGenPower()
```

RETURNS

Available generation power

GetGroundingImpedance

Returns the impedance of the internal grounding.

```
[int valid,
float resistance,
float reactance  ] ElmSym.GetGroundingImpedance(int busIdx)
```

ARGUMENTS

busIdx Bus index where the grounding should be determined.

resistance (out)
Real part of the grounding impedance in Ohm.

reactance (out)
Imaginary part of the grounding impedance in Ohm.

RETURNS

0	The values are invalid (e.g. because there is no internal grounding)
1	The values are valid.

GetMotorStartingFlag

Returns the starting motor condition.

```
int ElmSym.GetMotorStartingFlag()
```

RETURNS

Returns the motor starting condition. Possible values are:

-1	in the process of being calculated
0	not calculated
1	successful start
2	unsuccessful start

GetStepupTransformer

Performs a topological search to find the step-up transformer of the synchronous machine.

```
DataObject ElmSym.GetStepupTransformer(float voltage,
                                       int swStatus
                                       )
```

ARGUMENTS

voltage voltage level at which the search will stop

swStatus consideration of switch status. Possible values are:

0	consider all switch status
1	ignore breaker status
2	ignore all switch status

RETURNS

Returns the first collected step-up transformer object. It is empty if not found (e.g. start terminal already at hvVoltage).

IsConnected

Checks if a synchronous machine is already connected to any busbar.

```
int ElmSym.IsConnected()
```

RETURNS

0	false, not connected to a busbar
1	true, generator is connected to a busbar

Reconnect

Connects a synchronous machine by closing all switches (breakers and isolators) up to the first breaker on the HV side of a transformer. The topological search to find all the switches, stops at any busbar.

```
int ElmSym.Reconnect()
```

RETURNS

0	the machine was successfully closed
1	a error occurred and the machine could not be connected to any busbar

ResetDerating

Resets the derating value, setting the Max. Active Power Rating according to the rating factor. The following formula is used: $Pmax\_uc = pmaxratf * Pn * ngnum$.

```
None ElmSym.ResetDerating()
```

4.2.29 ElmTerm

Overview

GetBusType
GetCalcRelevantCubicles
GetConnectedBrkCubicles
GetConnectedCubicles
GetConnectedMainBuses
GetConnectionInfo
GetMinDistance
GetNextHVBus
GetNodeName
GetSepStationAreas
HasCreatedCalBus
IsElectrEquivalent
IsEquivalent
IsInternalNodeInStation
UpdateSubstationTerminals

GetBusType

Gets busbar calculation type.

```
int ElmTerm.GetBusType()
```

RETURNS

0	No valid calculation (load flow).
1	QV busbar.
2	PV busbar.
3	Slack busbar.

GetCalcRelevantCubicles

This function gets calculation relevant cubicles of this terminal.

```
list ElmTerm.GetCalcRelevantCubicles()
```

RETURNS

Set of calculation relevant cubicles.

GetConnectedBrkCubicles

Function gets the set of cubicles connected with the breaker and this terminal.

```
list ElmTerm.GetConnectedBrkCubicles([float ignoreSwitchStates])
```

ARGUMENTS

ignoreSwitchStates (optional)
Ignore switch status flag 1 or not 0 (=default).

RETURNS

Set of cubicles.

GetConnectedCubicles

Function gets the set of cubicles connected with this terminal.

```
list ElmTerm.GetConnectedCubicles([float ignoreSwitchStates])
```

ARGUMENTS

ignoreSwitchStates (optional)
Ignore switch status flag 1 or not 0 (=default).

RETURNS

Set of cubicles.

GetConnectedMainBuses

Function gets the set of connected main buses.

```
list ElmTerm.GetConnectedMainBuses([float considerSwitches])
```

ARGUMENTS

considerSwitches (optional)
    Consider switch state (default 1).

RETURNS

Set of main buses connected to the terminal.

GetConnectionInfo

Gets connection information of this terminal. Requires valid load flow calculation. Input arguments are filled with the value after function call.

```
[int error,
float closedSwitches,
float allSwitches,
float nonSwitchingDevices,
float closedAndNonSwitchingDevices,
float allDevices,
float connectedNodes,
float mainNodes] ElmTerm.GetConnectionInfo()
```

ARGUMENTS

closedSwitches
    Number of closed switch devices.

allSwitches
    Number of total switch devices.

nonSwitchingDevices
    Number of non-switch devices.

closedAndNonSwitchingDevices
    Number of total closed and non-switch devices (closedSwitches+nonSwitchingDevices).

allDevices
    Number of total switch and non-switch devices (allSwitches+nonSwitchingDevices).

connectedNodes
    Number of total nodes connected via couplers.

mainNodes
    Number of total main nodes.

RETURNS

Return value is always 0 and has no meaning.

GetMinDistance

This function determines the shortest path between the terminal the function was called on and the terminal that was passed as first argument. The distance is determined on network topology regarding the length of the traversed component (i.e. only lines have an influence on distance).

```
[float minDistance,
set path          ] ElmTerm.GetMinDistance(DataObject term,
                                           [int considerSwitches,]
                                           [set limitToNodes])
```

ARGUMENTS

term Terminal to which the shortest path is determined.

considerSwitches (optional)

0	Traverse all components, ignore switch states
1	Do not traverse open switch devices (default)

path (optional, out)
If given, all components of the found shortest path are put into this set.

limitToNodes(optional)
If given, the shortest path is searched only within this set of nodes. Please note, when limiting search to a given set of nodes, the start and end terminals (for which the distance is determined) must be part of this set (otherwise distance =-1).

RETURNS

< 0	If there is no path between the two terminals
≥ 0	Distance of shortest path in km

GetNextHVBus

This function returns the nearest connected busbar that has a higher voltage level. To detect this bus, a breath-first search on the net topology is executed. The traversal stops on each element that is out of service and on each opened switch device. The criterion for higher voltage level is passing a transformer to HV side. No junction nor internal nodes shall be considered.

```
DataObject ElmTerm.GetNextHVBus()
```

RETURNS

object	First busbar found.
None	If no busbar was found.

GetNodeName

For terminals inside a station, this function returns a unique name for the split the terminal is located in. The name is built on first five characters of the station‘s short name plus the split index separated by an underscore. E.g. “USTAT_1”.
For terminals inside a branch (*ElmBranch*) the returned name is just a concatenation of the branch name and the terminal‘s name.
For all other terminals not inside a branch or a station the node name corresponds to the terminal‘s name.

```
str ElmTerm.GetNodeName()
```

RETURNS

Node name as described above. Never empty.

GetSepStationAreas

Function gets all separate areas within the substation linked to this terminal. In this manner, area is any part between two nodes.

```
list ElmTerm.GetSepStationAreas([float considerSwitches])
```

ARGUMENTS

considerSwitches (optional)
Consider switch state (default 1).

RETURNS

Set of all separate areas in this substation.

HasCreatedCalBus

This function checks if the valid calculation exists for this terminal (i.e. load flow). If it exists, then the calculation parameters could be retrieved.

```
int ElmTerm.HasCreatedCalBus()
```

RETURNS

1	Valid calculation exists.
0	No valid calculation.

IsElectrEquivalent

Function checks if two terminals are electrically equivalent. Two terminals are said to be electrically equivalent if they are topologically connected only by

- closed switching devices (*ElmCoup*, *RelFuse*) or
- lines of zero length (line droppers) or
- branch components whose impedance is below given thresholds (R $\leq$ maxR and X $\leq$ maxX)

```
int ElmTerm.IsElectrEquivalent(DataObject terminal,
                               float maxR,
                               float maxX
                               )
```

ARGUMENTS

terminal Terminal to which the 'method called terminal' is connected to.

double maxR
Given threshold for the resistance of branch elements (must be given in Ohm).

double maxX
Given threshold for the reactance of branch elements (must be given in Ohm).

RETURNS

1	If terminal on which the method was called is electrical equivalent to terminal that was passed as argument
0	Otherwise

SEE ALSO

ElmTerm.IsEquivalent()

IsEquivalent

Function checks if two terminals are topologically connected only by

- closed switching devices (ElmCoup, RelFuse) or
- lines of zero length (line droppers).

IsEquivalent defines a relation that is

- symmetric (Term1.IsEquivalent(Term2) -> Term2.IsEquivalent(Term1)),
- reflexive (Term1.IsEquivalent(Term1)) and
- transitive (Term1.IsEquivalent(Term2) and Term2.IsEquivalent(Term3) -> Term1.IsEquivalent(Term3));

```
int ElmTerm.IsEquivalent(DataObject terminal)
```

ARGUMENTS

terminal Terminal (object of class ElmTerm) that is checked to be equivalent to the terminal on which the function was called on. Passing None is not allowed and will result in a scripting error.

RETURNS

1	If terminal on which the method was called is connected to terminal that was passed as argument only by closed switching devices or by lines of zero length
0	Otherwise (terminals are not connected or connected by other components than switching devices / lines of zero length)

SEE ALSO

ElmTerm.IsElectrEquivalent()

IsInternalNodeInStation

Function checks if the terminal is an internal node and in a station (*ElmSubstat*, *ElmTrfstat*).

```
int ElmTerm.IsInternalNodeInSubStation()
```

RETURNS

1	Terminal is a node of usage ‘internal’ and is located in a station.
0	Not internal node or not in a station, or both.

UpdateSubstationTerminals

Updates all nodes within the substation to the new voltage and/or phase technology. Applicable for all busbars and junction nodes. The highest voltage is taken as the leading one.

```
None ElmTerm.UpdateSubstationTerminals(int volt,
                                       int phs
                                       )
```

ARGUMENTS

volt	Updates nominal voltages (<> 0)
phs	Updates phase technology (<> 0)

4.2.30 ElmTr2

Overview

CreateEvent

For the corresponding transformer, a Tap Event (EvtTap) is created for the simulation.

```
int ElmTr2.CreateEvent([float tapAction,]
                       [float tapPos]
                       )
```

ARGUMENTS

tapAction (optional)
0=increase tap; 1=decrease tap; 2=set tap to tapPos; 3=manual; 4=automatic

tapPos (optional)
Position of tap

RETURNS

0 on success

GetGroundingImpedance

Returns the impedance of the internal grounding.

```
[int valid,
float resistance,
float reactance  ] ElmTr2.GetGroundingImpedance(int busIdx)
```

ARGUMENTS

busIdx Bus index where the grounding should be determined.

resistance (out)
Real part of the grounding impedance in Ohm.

reactance (out)
Imaginary part of the grounding impedance in Ohm.

RETURNS

0	The values are invalid (e.g. because there is no internal grounding)
1	The values are valid.

GetSuppliedElements

Returns the network components that are supplied by the transformer.
A network component is considered to be supplied by a transformer if a topological path from the transformer to the component exists. A valid topological path in this sense is a path that starts at the transformer's HV side in direction of transformer (not in direction of HV connected node) and stops at

- network components that are out of calculation,
- network components that are not active (e.g. hidden or those of currently inactive grids),
- open switches,
- connections leading to a higher voltage level.

Generally all network components of such a path are considered to be supplied by the transformer. Exceptions are components that are out of calculation or in-active. Those components are never considered to be supplied by any transformer.
A transformer is never considered to supply itself.
Composite components such as *ElmBranch*, *ElmSubstat*, *ElmTrfstat* are considered to be supplied by a transformer if all energized components inside that composite are supplied by the transformer.

```
list ElmTr2.GetSuppliedElements([int inclNested])
```

ARGUMENTS

inclNested (optional)

0	Only include components which are directly supplied by the transformer (not nested components)
1	Include nested components and components that are directly supplied by the transformer (default)

SEE ALSO

ElmTr3.GetSuppliedElements(), ElmSubstat.GetSuppliedElements(), ElmTrfstat.GetSuppliedElements()

GetTapPhi

Gets the tap phase shift in deg of the transformer for given tap position.

```
float ElmTr2.GetTapPhi(int itappos,
                       int inclPhaseShift
                       )
```

ARGUMENTS

itappos Tap position

inclPhaseShift
1 = Includes the vector group phase shift, 0 = consider only the tap phase shift

RETURNS

Returns the tap phase shift angle of the transforrmer for given tap position

GetTapRatio

Gets the voltage ratio of the transformer for given tap position.

```
float ElmTr2.GetTapRatio(int itappos,
                         int onlyTapSide,
                         int includeNomRatio
                         )
```

ARGUMENTS

itappos Tap position

onlyTapSide
1 = ratio only for given side., 0 = total ratio

includeNomRatio
1 = Includes nominal ratio of the transformer, 0 = consider only tap ratio

RETURNS

Returns the voltage ratio of the transforrmer for given tap position

GetZ0pu

Gets the zero-sequence impedance in p.u. of the transformer for the specified tap position. If the tap position is out of the tap changer range, the respective min. or max. position will be used.

```
[float r0pu,
float x0pu ] ElmTr2.GetZ0pu(int itappos,
                            int systembase)
```

ARGUMENTS

itappos Tap position

r0pu (out)
Resistance in p.u.

x0pu (out)
Reactance in p.u.

systembase

0	p.u. is based on rated power.
1	p.u. is based on system base (e.g. 100MVA).

GetZpu

Gets the impedance in p.u. of the transformer for the specified tap position. If the tap position is out of the tap changer range, the respective min. or max. position will be used.

```
[float rpu,
float xpu ] ElmTr2.GetZpu(int itappos,
                          int systembase)
```

ARGUMENTS

itappos	Tap position
rpu (out)	Resistance in p.u.
xpu (out)	Reactance in p.u.

systembase

0	p.u. is based on rated power.
1	p.u. is based on system base (e.g. 100MVA).

IsQuadBooster

Returns whether transformer is a quadbooster; i.e. checks phase shift angle modulus 180°.

```
int ElmTr2.IsQuadBooster()
```

RETURNS

'1' if quadbooster, else '0'

NTap

Gets the transformer tap position.

```
int ElmTr2.NTap()
```

RETURNS

The tap position.

4.2.31 ElmTr3

Overview

CreateEvent
GetGroundingImpedance
GetSuppliedElements
GetTapPhi
GetTapRatio
GetTapZDependentSide
GetZ0pu
GetZpu
IsQuadBooster
NTap

CreateEvent

For the corresponding transformer, a Tap Event (EvtTap) is created for the simulation.

```
int ElmTr3.CreateEvent([float tapAction,]
                       [float tapPos,]
                       [float busIdx]
                       )
```

ARGUMENTS

tapAction (optional)
0=increase tap; 1=decrease tap; 2=set tap to tapPos; 3=manual; 4=automatic

tapPos (optional)
Position of tap

busIdx (optional)
Bus index

RETURNS

0 on success

GetGroundingImpedance

Returns the impedance of the internal grounding.

```
[int valid,
float resistance,
float reactance  ] ElmTr3.GetGroundingImpedance(int busIdx)
```

ARGUMENTS

busIdx Bus index where the grounding should be determined.

resistance (out)
Real part of the grounding impedance in Ohm.

reactance (out)
Imaginary part of the grounding impedance in Ohm.

RETURNS

0	The values are invalid (e.g. because there is no internal grounding)
1	The values are valid.

GetSuppliedElements

Returns the network components that are supplied by the transformer.
A network component is considered to be supplied by a transformer if a topological path from the transformer to the component exists. A valid topological path in this sense is a path that starts at the transformer's HV side in direction of transformer (not in direction of HV connected node) and stops at

- network components that are out of calculation,
- network components that are not active (e.g. hidden or those of currently inactive grids),
- open switches,
- connections leading to a higher voltage level.

Generally all network components of such a path are considered to be supplied by the transformer. Exceptions are components that are out of calculation or in-active. Those components are never considered to be supplied by any transformer.
A transformer is never considered to supply itself.
Composite components such as *ElmBranch*, *ElmSubstat*, *ElmTrfstat* are considered to be supplied by a transformer if all energized components inside that composite are supplied by the transformer.

```
list ElmTr3.GetSuppliedElements([int inclNested])
```

ARGUMENTS

inclNested (optional)

0	Only include components which are directly supplied by the transformer (not nested components)
1	Include nested components and components that are directly supplied by the transformer (default)

SEE ALSO

ElmTr2.GetSuppliedElements(), ElmSubstat.GetSuppliedElements(), ElmTrfstat.GetSuppliedElements()

GetTapPhi

Gets the tap phase shift in deg of the transformer for given tap position and side.

```
float ElmTr2.GetTapPhi(int iSide,
                       int itappos,
                       int inclPhaseShift
                       )
```

ARGUMENTS

iSide for tap at side (0=Hv, 1=Mv, 2=Lv)

itappos Tap position for corresponding side

inclPhaseShift
1 = Includes the vector group phase shift, 0 = consider only the tap phase shift

RETURNS

Returns the tap phase shift angle of the transforrmer for given tap position and side

GetTapRatio

Gets the voltage ratio of the transformer for given tap position and side.

```
float ElmTr2.GetTapRatio(int iSide,
                         int itappos,
                         int includeNomRatio
                         )
```

ARGUMENTS

iSide for tap at side (0=Hv, 1=Mv, 2=Lv)

itappos Tap position at corresponding side

includeNomRatio
1 = Includes nominal ratio of the transformer, 0 = consider only tap ratio

RETURNS

Returns the voltage ratio of the transforrmer for given tap position and side

GetTapZDependentSide

Get tap side used for the dependent impedance

```
None  ElmTr3.GetTapZDependentSide()
```

RETURNS

-1	if no tap dependent impedance is defined
0	for HV tap
1	for MV tap
2	for LV tap

GetZ0pu

Gets the zero-sequence impedance in p.u. of the transformer for the specified tap position. If the tap position is out of the tap changer range, the respective min. or max. position will be used.

```
[float r0pu,
float x0pu ] ElmTr3.GetZ0pu(int itappos,
                            int iSide,
                            int systembase)
```

ARGUMENTS

itappos Tap position of the z-dependent tap

iSide

0	Get the HV-MV impedance.
1	Get the MV-LV impedance.
2	Get the LV-HV impedance.

r0pu (out)
Resistance in p.u.

x0pu (out)
Reactance in p.u.

systembase

0	p.u. is based on rated power.
1	p.u. is based on system base (e.g. 100MVA).

GetZpu

Gets the impedance in p.u. of the transformer for the specified tap position. If the tap position is out of the tap changer range, the respective min. or max. position will be used.

```
[float rpu,
float xpu ] ElmTr3.GetZpu(int itappos,
                          int iSide,
                          int systembase)
```

ARGUMENTS

itappos	Tap position of the z-dependent tap
iSide	
0	Get the HV-MV impedance.
1	Get the MV-LV impedance.
2	Get the LV-HV impedance.
rpu (out)	Resistance in p.u.
xpu (out)	Reactance in p.u.
systembase	
0	p.u. is based on rated power.
1	p.u. is based on system base (e.g. 100MVA).

IsQuadBooster

Returns whether transformer is a quadbooster or not, i.e. checks phase shift angle modulus 180°.

```
int ElmTr3.IsQuadBooster()
```

RETURNS

'1' if the transformer phase shift angle modulus 180° does not equal 0 at any of the sides LV, MV, HV, else '0'

NTap

Gets the transformer tap position.

```
int ElmTr3.NTap(float busIdx)
```

ARGUMENTS

busIdx 0=HV, 1=MV, 2=LV

RETURNS

The tap position.

4.2.32 ElmTr4

Overview

CreateEvent
GetGroundingImpedance
GetSuppliedElements
GetTapPhi
GetTapRatio
GetTapZDependentSide
GetZ0pu
GetZpu
IsQuadBooster
NTap

CreateEvent

For the corresponding transformer, a Tap Event (EvtTap) is created for the simulation.

```
int ElmTr4.CreateEvent([float tapAction,]
                       [float tapPos,]
                       [float busIdx]
                       )
```

ARGUMENTS

tapAction (optional)
0=increase tap; 1=decrease tap; 2=set tap to tapPos; 3=manual; 4=automatic

tapPos (optional)
Position of tap

busIdx (optional)
Bus index

RETURNS

0 on success

GetGroundingImpedance

Returns the impedance of the internal grounding.

```
[int valid,
float resistance,
float reactance  ] ElmTr4.GetGroundingImpedance(int busIdx)
```

ARGUMENTS

busIdx Bus index where the grounding should be determined.

resistance (out)
Real part of the grounding impedance in Ohm.

reactance (out)
Imaginary part of the grounding impedance in Ohm.

RETURNS

0	The values are invalid (e.g. because there is no internal grounding)
1	The values are valid.

GetSuppliedElements

Returns the network components that are supplied by the transformer.
A network component is considered to be supplied by a transformer if a topological path from the transformer to the component exists. A valid topological path in this sense is a path that starts at the transformer's HV side in direction of transformer (not in direction of HV connected node) and stops at

- network components that are out of calculation,
- network components that are not active (e.g. hidden or those of currently inactive grids),
- open switches,

- connections leading to a higher voltage level.

Generally all network components of such a path are considered to be supplied by the transformer. Exceptions are components that are out of calculation or in-active. Those components are never considered to be supplied by any transformer.
A transformer is never considered to supply itself.
Composite components such as *ElmBranch*, *ElmSubstat*, *ElmTrfstat* are considered to be supplied by a transformer if all energized components inside that composite are supplied by the transformer.

```
list ElmTr4.GetSuppliedElements([int inclNested])
```

ARGUMENTS

inclNested (optional)

0	Only include components which are directly supplied by the transformer (not nested components)
1	Include nested components and components that are directly supplied by the transformer (default)

SEE ALSO

ElmTr2.GetSuppliedElements(), ElmSubstat.GetSuppliedElements(), ElmTrfstat.GetSuppliedElements()

GetTapPhi

Gets the tap phase shift in deg of the transformer for given tap position and side.

```
float ElmTr4.GetTapPhi(int iSide,
                       int itappos,
                       int inclPhaseShift
                       )
```

ARGUMENTS

iSide	for tap at side (0=HV, 1=LV1, 2=Lv2, 3=Lv3)
itappos	Tap position for corresponding side
inclPhaseShift	1 = Includes the vector group phase shift, 0 = consider only the tap phase shift

RETURNS

Returns the tap phase shift angle of the transforrmer for given tap position and side

GetTapRatio

Gets the voltage ratio of the transformer for given tap position and side.

```
float ElmTr4.GetTapRatio(int iSide,
                         int itappos,
                         int includeNomRatio
                         )
```

ARGUMENTS

iSide	for tap at side (0=HV, 1=LV1, 2=Lv2, 3=Lv3)
itappos	Tap position at corresponding side
includeNomRatio	1 = Includes nominal ratio of the transformer, 0 = consider only tap ratio

RETURNS

Returns the voltage ratio of the transforrmer for given tap position and side

GetTapZDependentSide

Get tap side used for the dependent impedance

```
None  ElmTr4.GetTapZDependentSide()
```

RETURNS

-1	if no tap dependent impedance is defined
0	for HV tap
1	for LV1 tap
2	for LV2 tap
2	for LV3 tap

GetZ0pu

Gets the zero-sequence impedance in p.u. of the transformer for the specified tap position. If the tap position is out of the tap changer range, the respective min. or max. position will be used.

```
[float r0pu,
float x0pu ] ElmTr4.GetZ0pu(int itappos,
                            int iSide,
                            int systembase)
```

ARGUMENTS

itappos Tap position of the z-dependent tap

iSide

0	Get the HV-LV1 impedance.
1	Get the HV-LV2 impedance.
2	Get the HV-LV3 impedance.
3	Get the LV1-LV2 impedance.
4	Get the LV1-LV3 impedance.
5	Get the LV2-LV3 impedance.

r0pu (out)
Resistance in p.u.

x0pu (out)
Reactance in p.u.

systembase

0	p.u. is based on rated power.
1	p.u. is based on system base (e.g. 100MVA).

GetZpu

Gets the impedance in p.u. of the transformer for the specified tap position. If the tap position is out of the tap changer range, the respective min. or max. position will be used.

```
[float rpu,
float xpu ] ElmTr4.GetZpu(int itappos,
                          int iSide,
                          int systembase)
```

ARGUMENTS

itappos Tap position of the z-dependent tap

iSide

0	Get the HV-LV1 impedance.
1	Get the HV-LV2 impedance.
2	Get the HV-LV3 impedance.
3	Get the LV1-LV2 impedance.
4	Get the LV1-LV3 impedance.
5	Get the LV2-LV3 impedance.

rpu (out) Resistance in p.u.

xpu (out) Reactance in p.u.

systembase

0	p.u. is based on rated power.
1	p.u. is based on system base (e.g. 100MVA).

IsQuadBooster

Returns whether transformer is a quadbooster or not, i.e. checks phase shift angle modulus 180°.

```
int ElmTr4.IsQuadBooster()
```

RETURNS

'1' if the transformer phase shift angle modulus 180° does not equal 0 at any of the sides HV, LV1, LV2, LV3, else '0'

NTap

Gets the transformer tap position.

```
int ElmTr4.NTap(float busIdx)
```

ARGUMENTS

busIdx 0=HV, 1=MV, 2=LV

RETURNS

The tap position.

4.2.33 ElmTrfstat

Overview

GetSplit
GetSplitCal
GetSplitIndex
GetSuppliedElements

GetSplit

A split of a station is a group of topologically connected elements. Such a group is called split if all contained components are energized and there is at least one busbar (terminal of usage 'busbar') contained or it has connections to at least two main components (= all components except switch devices and terminals).
These splits are ordered according to the count of nodes contained and according to their priority. So each split becomes a unique index.

The function GetSplit offers access to the elements contained in a split. By calling GetSplit with an index from 0 to n, the elements belonging to the corresponding split are filled into given sets and returned.

```
[int error
list mainNodes,
list connectionCubicles,
list allElements        ] ElmTrfstat.GetSplit(int index)
```

ARGUMENTS

index Index of the split used to access the elements of the corresponding split. Value must be ≥ 0.

mainNodes (out)
Terminals of same usage considered to form the most important nodes for that group. In most cases, this is the group of contained busbars.

connectionCubicles (optional, out)
All cubicles (of terminals inside the station) that point to an element that sits outside the station or to an element that is connected to a terminal outside the station are filled into the set connectionCubicles. (The connection element (branch) can be accessed by calling GetBranch() on each of these cubicles. The terminals of these cubicles (parents) must not necessarily be contained in any split. They could also be separated by a disconnecting component.)

allElements(optional, out)
All elements (class Elm*) of the split that have no connection to elements outside the station are filled into this set.

RETURNS

0 success, split of that index exists and is returned.

1 indicates that there exists no split with given index. (Moreover, this means that there is no split with index n greater than this value.)

SEE ALSO

ElmTrfstat.GetSplitCal(), ElmTrfstat.GetSplitIndex(),

GetSplitCal

This function determines the elements that belong to a split. In contrast to ElmTrfstat.GetSplit() it is based on calculation instead of pure edit object topology. This means the returned nodes correspond to the calculation nodes, the interconnecting cubicles are those connecting nodes of different splits.
Note: As this function relies on calculation nodes it can only be executed after a calculation has been performed (e.g. load flow calculation).

```
[int error,
list nodes,
list connectionCubicles,
list elements           ] ElmTrfstat.GetSplitCal(int index)
```

ARGUMENTS

index Index of the split used to access the elements of the corresponding split. Refers to same split as index in ElmTrfstat.GetSplit().
Value must be $\geq$ 0.

nodes (out)
A set that is filled with terminals. There is one terminal returned for each calculation node in the split.

connectionCubicles (optional, out)
This set is filled with all cubicles that point from a calculation node of current split to another calculation node that does not belong to that split. The connecting element can be accessed by calling GetBranch() on such a cubicle.

elements (optional, out)
This set is filled with network elements that are connected to a calculation node of current split and have exactly one connection, i.e. these elements are completely contained in the split.

RETURNS

0 success, split of that index exists and is returned.

1 indicates that there exists no split with given index. (Moreover, this means that there is no split with index n greater than this value.)

SEE ALSO

ElmTrfstat.GetSplit()

GetSplitIndex

This function returns the index of the split that contains passed object.

```
int ElmTrfstat.GetSplitIndex(DataObject o)
```

ARGUMENTS

o Object for which the split index is to be determined.

Returns

≥ 0	index of split in which element is contained
-1	given object does not belong to any split of that station

See also

ElmTrfstat.GetSplit()

GetSuppliedElements

Returns all network components that are supplied by the transformers located in the station.

```
list ElmTrfstat.GetSuppliedElements([int inclNested])
```

Arguments

inclNested (optional)

0	Do not include components that are supplied by nested supplying stations
1	(default) Include components that are supplied by nested stations

See also

ElmTr2.GetSuppliedElements(), ElmTr3.GetSuppliedElements()

4.2.34 ElmVoltreg

Overview

CreateEvent
GetGroundingImpedance
GetZpu
NTap

CreateEvent

For the corresponding voltage regulator, a Tap Event (EvtTap) is created for the simulation.

```
int ElmVoltreg.CreateEvent([float tapAction,]
                           [float tapPos]
                           )
```

Arguments

tapAction (optional)
0=increase tap; 1=decrease tap; 2=set tap to tapPos; 3=manual; 4=automatic

tapPos (optional)
Position of tap

Returns

0 on success

GetGroundingImpedance

Returns the impedance of the internal grounding.

```
[int valid,
float resistance,
float reactance  ] ElmVoltreg.GetGroundingImpedance(int busIdx)
```

ARGUMENTS

busIdx Bus index where the grounding should be determined.

resistance (out)
Real part of the grounding impedance in Ohm.

reactance (out)
Imaginary part of the grounding impedance in Ohm.

RETURNS

0	The values are invalid (e.g. because there is no internal grounding)
1	The values are valid.

GetZpu

Gets the impedance in p.u. of the voltage regulator for the specified tap position. If the tap position is out of the tap changer range, the respective min. or max. position will be used.

```
[float rpu,
float xpu ] ElmVoltreg.GetZpu(int itappos,
                              int systembase)
```

ARGUMENTS

itappos Tap position

rpu (out) Resistance in p.u.

xpu (out) Reactance in p.u.

systembase

0	p.u. is based on rated power.
1	p.u. is based on system base (e.g. 100MVA).

NTap

Gets the voltage regulator tap position.

```
int ElmVoltreg.NTap()
```

RETURNS

The tap position.

4.2.35 ElmXnet

Overview

GetGroundingImpedance
GetStepupTransformer

GetGroundingImpedance

Returns the impedance of the internal grounding.

```
[int valid,
float resistance,
float reactance  ] ElmXnet.GetGroundingImpedance(int busIdx)
```

ARGUMENTS

busIdx Bus index where the grounding should be determined.

resistance (out)
Real part of the grounding impedance in Ohm.

reactance (out)
Imaginary part of the grounding impedance in Ohm.

RETURNS

0	The values are invalid (e.g. because there is no internal grounding)
1	The values are valid.

GetStepupTransformer

Performs a topological search to find the step-up transformer of an external grid

```
DataObject ElmXnet.GetStepupTransformer(float voltage,
                                        int swStatus
                                        )
```

ARGUMENTS

voltage voltage level at which the search will stop

swStatus consideration of switch status. Possible values are:

0	consider all switch status
1	ignore breaker status
2	ignore all switch status

RETURNS

Returns the first collected step-up transformer object. It is empty if not found (e.g. start terminal already at hvVoltage).

4.2.36 ElmZone

Overview

CalcBoundary

Defines boundary with this zone as exterior part. Resulting cubicles of boundary are branch-oriented away from the zone.

```
[int error,
DataObject boundary] ElmZone.CalcBoundary(float shift)
```

ARGUMENTS

shift Elements that are within a distance of shift many elements to a boundary cubicle of the zone are added to the exterior part of the resulting boundary.

boundary (out)
Defined boundary.

RETURNS

0	Successful call, boundary defined.
1	Error during determination of boundary cubicles.

CalculateInterchangeTo

Calculates interchange power to the given zone (calculated quantities are: Pinter, Qinter, Pexport, Qexport, Pimort, Qimport). Prior the calculation the valid load flow calculation is required.

```
int ElmZone.CalculateInterchangeTo(DataObject zone)
```

ARGUMENTS

zone zone to which the interchage is calculated

RETURNS

$<$ **0**	calculation error (i.e. no valid load flow, empty zone...)
0	no interchage power to the given zone
1	interchange power calculated

GetAll

Returns all objects which belong to this zone.

```
list ElmZone.GetAll()
```

Returns

The set of objects.

GetBranches

Returns all branches which belong to this zone.

```
list ElmZone.GetBranches()
```

Returns

The set of branch objects.

GetBuses

Returns all buses which belong to this zone.

```
list ElmZone.GetBuses()
```

Returns

The set of objects.

GetObjs

Returns all objects of the given class which belong to this zone.

```
list ElmZone.GetObjs(str classname)
```

Arguments

classname
name of the class (i.e. ”ElmTr2”)

Returns

The set of contained objects.

SetLoadScaleAbsolute

Readjusts zonal load scaling factor to the given active power. The zonal load scaling factor is the ratio of the given active power and the loads total actual power.

```
None ElmZone.SetLoadScaleAbsolute(float Pin)
```

Arguments

Pin active power in MW used for the load scaling factor.

4.3 Station Elements

4.3.1 StaCt

Overview

SetPrimaryTap

SetPrimaryTap

Function automatically sets primary tap depending on the nominal current of the measured branch. The pattern according to which the tap current is found: BranchCurrent = BranchNominalCurrent * Factor. The primary tap currents are checked wheather there are equal or at least first higher value than BranchCurrent, otherwise selects the maximum one.

```
int StaCt.SetPrimaryTap([float mltFactor])
```

ARGUMENTS

mltFactor (optional)
Multiplication factor (default 1.0)

RETURNS

0	Correctly set.
1	Error.

4.3.2 StaCubic

Overview

GetAll
GetBranch
GetConnectedMajorNodes
GetConnections
GetNearestBusbars
GetPathToNearestBusbar
IsClosed
IsConnected

GetAll

This function returns a set of network components that are collected by a topological traversal starting from this cubicle.

```
list StaCubic.GetAll([int direction = 1,]
                     [int ignoreOpenSwitches = 0]
                     )
```

ARGUMENTS

direction (optional)
Specifies the direction in which the network topology is traversed.

1	Traversal to the branch element (default).

0 Traversal to the terminal element.

ignoreOpenSwitches (optional)
Determines whether to pass open switches or to stop at them.

0 The traversal stops in a direction if an open switch is reached (default).

1 Ignore all switch statuses and pass every switch.

RETURNS

A set of network components that are collected by a topological traversal starting at the cubicle (StaCubic) where the function is called.

GetBranch

Function gets the branch of this cubicle.

```
DataObject StaCubic.GetBranch()
```

RETURNS

Branch object.

GetConnectedMajorNodes

This function returns all busbars being part of a split (inside a station) that can be reached by starting a topology search from the cubicle in direction of the branch element.

```
list StaCubic.GetConnectedMajorNodes ([float swtStat])
```

ARGUMENTS

swtStat

0 (default) First perform a search that respects switch states (stoping at open switches). If no switches are found, an additional search with ignoring switch states.

1 Perform one search ignoring switch states (passing open and closed switches).

2 Search with respecting switch states. But do no additional search when switch is found.

RETURNS

A set of all busbars that can be reached starting a topology search from the cubicle in direction of the branch element.

GetConnections

Function gets all elements connected with this cubicle.

```
list StaCubic.GetConnections(int swtStat)
```

ARGUMENTS

swtStat Consider switch status (1) or not (0).

RETURNS

Set of elements.

GetNearestBusbars

Function searches for connected and connectable nearest busbars starting at the cubicle. Search stops at the nearest busbars and out of service elements. Internal and junction nodes, all types of branch elements and all types of switches i.e. circuit-breakers and disconnectors are passed.

Connected busbars are all busbars which are topologically connected to the start cubicle without passing open switches. Connectable busbars are all busbars which are connectable to the start cubicle by closing switches. If the start cubicles terminal is a busbar then this busbar is not included in the result sets.

If more than one path exists between cubicle and a nearest busbar the relevant busbar is added only once to the result set.

```
[list connectedBusbars,
list connectableBusbars] StaCubic.GetNearestBusbars(int searchDirection)
```

ARGUMENTS

connectedBusbars (out)
Found connected busbars.

connectableBusbars (out)
Found connectable busbars.

searchDirection
Direction of the search relative to the cubicle. Possible values are

0	search in all directions
1	search in direction of cubicles terminal
2	search towards connected branch element

GetPathToNearestBusbar

Function determines the path from the cubicle to the given busbar. The busbar must be connected or connectable to the start cubicle without passing additional busbars. If the given busbar is not a nearest busbar in relation to the cubicle an empty path is returned.

If more than one closed path exists between cubicle and busbar the elements of all these paths are combined.

```
list StaCubic.GetPathToNearestBusbar(DataObject nearestBusbar)
```

ARGUMENTS

nearestBusbar
Nearest busbar in relation to cubicle.

RETURNS

Net elements of the path from cubicle to busbar.

IsClosed

Function checks if this cubicle is directly connected with the busbar, considering the switch status.

```
int StaCubic.IsClosed()
```

RETURNS

0	Disconnected cubicle.
1	Connected cubicle.

IsConnected

Function checks if the cubicle is connected to the passed terminal or coupler.

```
int StaCubic.IsConnected(DataObject elm,
                         int        swtStat
                         )
```

ARGUMENTS

elm	Terminal or coupler to check connection with.
swtStat	Consider switch status (1) or not (0).

RETURNS

0	Not connected.
1	Connected.

4.3.3 StaExtbrkmea

Overview

CopyExtMeaStatusToStatusTmp
GetMeaValue
GetStatus
GetStatusTmp
InitTmp
IsStatusBitSet
IsStatusBitSetTmp
ResetStatusBit
ResetStatusBitTmp
SetMeaValue
SetMeaValue
SetStatus
SetStatusBit
SetStatusBitTmp
SetStatusTmp
UpdateControl
UpdateCtrl

CopyExtMeaStatusToStatusTmp

Copies the (persistent) status of current measurement object to temporary (in memory) status.

```
None StaExtbrkmea.CopyExtMeaStatusToStatusTmp()
```

GetMeaValue

Returns the value for the switch position currently stored in the measurement object.

```
[int error,
float value] StaExtbrkmea.GetMeaValue()
```

Arguments

value (out)
Value for switch status.

Returns

Return value has no meaning. It is always 0.

GetStatus

Returns the status flags. Please note, this value is interpreted as a bitfield. See StaExtbrkmea.SetStatus() for details on the status bits.

```
int StaExtbrkmea.GetStatuts()
```

Returns

Status bitfield as an integer value.

GetStatusTmp

Returns the temporary (in memory) status flags. Please note, this value is interpreted as a bitfield. See StaExtbrkmea.SetStatus() for details on the status bits.

```
int StaExtbrkmea.GetStatusTmp()
```

Returns

Status bitfield as an integer value.

InitTmp

Initialises the temporary (in memory) fields of the measurement object with the values currently stored in the corresponding persistent fields. This affects temporary measurement value and temporary status fields. The temporary measurement value is used internally for comparison of new and old values for deadband violation. The temporary status is used during calculation in order to not modify initial value.

This function should be called once after the link has been established and before the calculation loop is executed.

```
None StaExtbrkmea.InitTmp()
```

IsStatusBitSet

Checks if specific bit(s) are set in the status bitfield. See StaExtbrkmea.SetStatus() for details on the status bits.

```
int StaExtbrkmea.IsStatusBitSet(int mask)
```

RETURNS

0	if at least one bit in mask is not set
1	if all bit(s) in mask are set

IsStatusBitSetTmp

Checks if specific bit(s) are set in the temporary (in memory) status bitfield. See StaExtbrkmea.SetStatus() for details on the status bits.

```
int StaExtbrkmea.IsStatusBitSetTmp(int mask)
```

RETURNS

0	if at least one bit in mask is not set
1	if all bit(s) in mask are set

ResetStatusBit

Resets specific bits in the status bitfield. See StaExtbrkmea.SetStatus() for details on the status bits.

```
None StaExtbrkmea.ResetStatusBit(int mask, int dbSync)
```

ARGUMENTS

mask Mask of bits to set to 0. A bit is unchanged if already unset before.

dbSync (optional)

0	New status flags are applied in memory only
1	Default, new status flags are stored on db (persistent)

ResetStatusBitTmp

Resets specific bits in the temporary (in memory) status bitfield. See StaExtbrkmea.SetStatus() for details on the status bits.

```
None StaExtbrkmea.ResetStatusBitTmp(int mask)
```

ARGUMENTS

mask Mask of bits to set to 0. A bit is unchanged if already unset before.

SetMeaValue

Sets the value for the switch position currently stored in the measurement object.

```
int StaExtbrkmea.SetMeaValue(int value)
```

ARGUMENTS

value	New value for switch status.

RETURNS

Return value has no meaning. It is always 0.

SetMeaValue

Sets the value stored in the measurement object.

```
int StaExtbrkmea.SetMeaValue(float value)
```

ARGUMENTS

value	New value.

RETURNS

Return value has no meaning. It is always 0.

SetStatus

Sets the status flags of the measurement object. Please note, this value is interpreted as a bitfield where the bits have the following meaning. An option is considered enabled if the corresponding bit is set to 1.

bit0 0x00000001	Manually entered data
bit1 0x00000002	Tele-Measurement
bit2 0x00000004	Disturbance
bit3 0x00000008	Protection
bit4 0x00000010	Marked suspect
bit5 0x00000020	Violated constraint
bit6 0x00000040	On Event
bit7 0x00000080	Event Block.
bit8 0x00000100	Alarm Block.
bit9 0x00000200	Update Block.
bit10 0x00000400	Control Block.
bit29 0x20000000	Read
bit30 0x40000000	Write
bit31 0x80000000	Neglected by SE

```
None StaExtbrkmea.SetStatus(int status, int dbSync)
```

ARGUMENTS

status Bitfield for status flags, see above

dbSync (optional)

0	New status flags are applied in memory only
1	Default, new status flags are stored on db (persistent)

SetStatusBit

Sets specific bits in the status bitfield. See StaExtbrkmea.SetStatus() for details on the status bits.

```
None StaExtbrkmea.SetStatusBit(int mask, int dbSync)
```

ARGUMENTS

mask Mask of bits to set to 1. A bit is unchanged if already set before.

dbSync (optional)

0	New status flags are applied in memory only
1	Default, new status flags are stored on db (persistent)

SetStatusBitTmp

Sets specific bits in the temporary (in memory) status bitfield. See StaExtbrkmea.SetStatus() for details on the status bits.

```
None StaExtbrkmea.SetStatusBitTmp(int mask)
```

ARGUMENTS

mask Mask of bits to set to 1. A bit is unchanged if already set before.

SetStatusTmp

Sets the temporary (in memory) status flags of the measurement object. This temporary value is used during calculations so that changes do not lead to object modifications and initial value remains unchanged.

Please note, this value is interpreted as a bitfield. See StaExtbrkmea.SetStatus() for details on the status bits.

```
None StaExtbrkmea.SetStatusTmp(int status)
```

ARGUMENTS

status Bitfield for status flags, see above

UpdateControl

Transfers the value of current measurement object to the controller object (target object 'pCtrl' and target attribute 'varName'). If target object is a command, it is automatically executed afterwards.
Note: Calculation results will not be reset by this value transfer.

```
int StaExtbrkmea.UpdateControl(int dbSync)
```

ARGUMENTS

dbSync (optional)

0	Value is copied in memory only
1	Default, copied value is stored on db (persistent)

RETURNS

0	on success
1	on error, e.g. target object does not have an attribute with given name

UpdateCtrl

Transfers the value of current measurement object to the controlled object (target object 'pObject' and target attribute 'variabName'). If target object is a command, it is automatically executed afterwards.
Note: Calculation results will not be reset by this value transfer.

```
int StaExtbrkmea.UpdateCtrl(int dbSync)
```

ARGUMENTS

dbSync (optional)

0	Value is copied in memory only
1	Default, copied value is stored on db (persistent)

RETURNS

0	on success
1	on error, e.g. target object does not have an attribute with given name

4.3.4 StaExtcmdmea

Overview

CopyExtMeaStatusToStatusTmp
GetMeaValue
GetStatus
GetStatusTmp
InitTmp
IsStatusBitSet
IsStatusBitSetTmp
ResetStatusBit
ResetStatusBitTmp
SetMeaValue
SetStatus

SetStatusBit
SetStatusBitTmp
SetStatusTmp
UpdateControl
UpdateCtrl

CopyExtMeaStatusToStatusTmp

Copies the (persistent) status of current measurement object to temporary (in memory) status.

```
None StaExtcmdmea.CopyExtMeaStatusToStatusTmp()
```

GetMeaValue

Returns the value for command interpreted as floating point value.

```
[int error,
float value] StaExtcmdmea.GetMeaValue()
```

ARGUMENTS

value (out)
Value obtained by parsing stored command string as floating point value.

RETURNS

Return value has no meaning. It is always 0.

GetStatus

Returns the status flags. Please note, this value is interpreted as a bitfield. See StaExtcmdmea.SetStatus() for details on the status bits.

```
int StaExtcmdmea.GetStatuts()
```

RETURNS

Status bitfield as an integer value.

GetStatusTmp

Returns the temporary (in memory) status flags. Please note, this value is interpreted as a bitfield. See StaExtcmdmea.SetStatus() for details on the status bits.

```
int StaExtcmdmea.GetStatusTmp()
```

RETURNS

Status bitfield as an integer value.

InitTmp

Initialises the temporary (in memory) fields of the measurement object with the values currently stored in the corresponding persistent fields. This affects temporary measurement value and temporary status fields. The temporary measurement value is used internally for comparison of new and old values for deadband violation. The temporary status is used during calculation in order to not modify initial value.

This function should be called once after the link has been established and before the calculation loop is executed.

```
None StaExtcmdmea.InitTmp()
```

IsStatusBitSet

Checks if specific bit(s) are set in the status bitfield. See StaExtcmdmea.SetStatus() for details on the status bits.

```
int StaExtcmdmea.IsStatusBitSet(int mask)
```

RETURNS

0	if at least one bit in mask is not set
1	if all bit(s) in mask are set

IsStatusBitSetTmp

Checks if specific bit(s) are set in the temporary (in memory) status bitfield. See StaExtcmdmea.SetStatus() for details on the status bits.

```
int StaExtcmdmea.IsStatusBitSetTmp(int mask)
```

RETURNS

0	if at least one bit in mask is not set
1	if all bit(s) in mask are set

ResetStatusBit

Resets specific bits in the status bitfield. See StaExtcmdmea.SetStatus() for details on the status bits.

```
None StaExtcmdmea.ResetStatusBit(int mask, int dbSync)
```

ARGUMENTS

mask Mask of bits to set to 0. A bit is unchanged if already unset before.

dbSync (optional)

0	New status flags are applied in memory only
1	Default, new status flags are stored on db (persistent)

ResetStatusBitTmp

Resets specific bits in the temporary (in memory) status bitfield. See StaExtcmdmea.SetStatus() for details on the status bits.

```
None StaExtcmdmea.ResetStatusBitTmp(int mask)
```

ARGUMENTS

mask	Mask of bits to set to 0. A bit is unchanged if already unset before.

SetMeaValue

Sets the value stored in the measurement object.

```
int StaExtcmdmea.SetMeaValue(float value)
```

ARGUMENTS

value	New value.

RETURNS

Return value has no meaning. It is always 0.

SetStatus

Sets the status flags of the measurement object. Please note, this value is interpreted as a bitfield where the bits have the following meaning. An option is considered enabled if the corresponding bit is set to 1.

bit0 0x00000001	Manually entered data
bit1 0x00000002	Tele-Measurement
bit2 0x00000004	Disturbance
bit3 0x00000008	Protection
bit4 0x00000010	Marked suspect
bit5 0x00000020	Violated constraint
bit6 0x00000040	On Event
bit7 0x00000080	Event Block.
bit8 0x00000100	Alarm Block.
bit9 0x00000200	Update Block.
bit10 0x00000400	Control Block.
bit29 0x20000000	Read
bit30 0x40000000	Write
bit31 0x80000000	Neglected by SE

```
None StaExtcmdmea.SetStatus(int status, int dbSync)
```

ARGUMENTS

status	Bitfield for status flags, see above

dbSync (optional)

0	New status flags are applied in memory only
1	Default, new status flags are stored on db (persistent)

SetStatusBit

Sets specific bits in the status bitfield. See StaExtcmdmea.SetStatus() for details on the status bits.

```
None StaExtcmdmea.SetStatusBit(int mask, int dbSync)
```

ARGUMENTS

mask	Mask of bits to set to 1. A bit is unchanged if already set before.

dbSync (optional)

0	New status flags are applied in memory only
1	Default, new status flags are stored on db (persistent)

SetStatusBitTmp

Sets specific bits in the temporary (in memory) status bitfield. See StaExtcmdmea.SetStatus() for details on the status bits.

```
None StaExtcmdmea.SetStatusBitTmp(int mask)
```

ARGUMENTS

mask	Mask of bits to set to 1. A bit is unchanged if already set before.

SetStatusTmp

Sets the temporary (in memory) status flags of the measurement object. This temporary value is used during calculations so that changes do not lead to object modifications and initial value remains unchanged.

Please note, this value is interpreted as a bitfield. See StaExtcmdmea.SetStatus() for details on the status bits.

```
None StaExtcmdmea.SetStatusTmp(int status)
```

ARGUMENTS

status	Bitfield for status flags, see above

UpdateControl

Transfers the value of current measurement object to the controller object (target object 'pCtrl' and target attribute 'varName'). If target object is a command, it is automatically executed afterwards.
Note: Calculation results will not be reset by this value transfer.

```
int StaExtcmdmea.UpdateControl(int dbSync)
```

ARGUMENTS

dbSync (optional)

0	Value is copied in memory only
1	Default, copied value is stored on db (persistent)

RETURNS

0	on success
1	on error, e.g. target object does not have an attribute with given name

UpdateCtrl

Transfers the value of current measurement object to the controlled object (target object 'pObject' and target attribute 'variabName'). If target object is a command, it is automatically executed afterwards.
Note: Calculation results will not be reset by this value transfer.

```
int StaExtcmdmea.UpdateCtrl(int dbSync)
```

ARGUMENTS

dbSync (optional)

0	Value is copied in memory only
1	Default, copied value is stored on db (persistent)

RETURNS

0	on success
1	on error, e.g. target object does not have an attribute with given name

4.3.5 StaExtdatmea

Overview

CopyExtMeaStatusToStatusTmp
CreateEvent
GetMeaValue
GetStatus
GetStatusTmp
InitTmp
IsStatusBitSet
IsStatusBitSetTmp
ResetStatusBit
ResetStatusBitTmp
SetMeaValue

SetStatus
SetStatusBit
SetStatusBitTmp
SetStatusTmp
UpdateControl
UpdateCtrl

CopyExtMeaStatusToStatusTmp

Copies the (persistent) status of current measurement object to temporary (in memory) status.

```
None StaExtdatmea.CopyExtMeaStatusToStatusTmp()
```

CreateEvent

Creates a “parameter change” event for controller object (’pCtrl’) and attribute (’varName’). The event is stored in simulation event list and executed immediately.

```
None StaExtdatmea.CreateEvent()
```

GetMeaValue

Returns the value stored in the measurement object.

```
[int error,
float value] StaExtdatmea.GetMeaValue(int unused,
                                      int applyMultiplicator)
```

ARGUMENTS

value (out)
Value, optionally modified by configured multiplicator

unused Not used.

applyMultiplicator
If 1 (default), returned value will be modified by the multiplicator stored in the measurement object (depending on Mode incremental/absolute). If 0, raw value will be returned.

RETURNS

Return value has no meaning. It is always 0.

GetStatus

Returns the status flags. Please note, this value is interpreted as a bitfield. See StaExtdatmea.SetStatus() for details on the status bits.

```
int StaExtdatmea.GetStatuts()
```

RETURNS

Status bitfield as an integer value.

GetStatusTmp

Returns the temporary (in memory) status flags. Please note, this value is interpreted as a bitfield. See StaExtdatmea.SetStatus() for details on the status bits.

```
int StaExtdatmea.GetStatusTmp()
```

RETURNS

Status bitfield as an integer value.

InitTmp

Initialises the temporary (in memory) fields of the measurement object with the values currently stored in the corresponding persistent fields. This affects temporary measurement value and temporary status fields. The temporary measurement value is used internally for comparison of new and old values for deadband violation. The temporary status is used during calculation in order to not modify initial value.

This function should be called once after the link has been established and before the calculation loop is executed.

```
None StaExtdatmea.InitTmp()
```

IsStatusBitSet

Checks if specific bit(s) are set in the status bitfield. See StaExtdatmea.SetStatus() for details on the status bits.

```
int StaExtdatmea.IsStatusBitSet(int mask)
```

RETURNS

0	if at least one bit in mask is not set
1	if all bit(s) in mask are set

IsStatusBitSetTmp

Checks if specific bit(s) are set in the temporary (in memory) status bitfield. See StaExtdatmea.SetStatus() for details on the status bits.

```
int StaExtdatmea.IsStatusBitSetTmp(int mask)
```

RETURNS

0	if at least one bit in mask is not set
1	if all bit(s) in mask are set

ResetStatusBit

Resets specific bits in the status bitfield. See StaExtdatmea.SetStatus() for details on the status bits.

```
None StaExtdatmea.ResetStatusBit(int mask, int dbSync)
```

ARGUMENTS

mask	Mask of bits to set to 0. A bit is unchanged if already unset before.
dbSync (optional)	
0	New status flags are applied in memory only
1	Default, new status flags are stored on db (persistent)

ResetStatusBitTmp

Resets specific bits in the temporary (in memory) status bitfield. See StaExtdatmea.SetStatus() for details on the status bits.

```
None StaExtdatmea.ResetStatusBitTmp(int mask)
```

ARGUMENTS

mask Mask of bits to set to 0. A bit is unchanged if already unset before.

SetMeaValue

Sets the value stored in the measurement object.

```
int StaExtdatmea.SetMeaValue(float value)
```

ARGUMENTS

value New value.

RETURNS

Return value has no meaning. It is always 0.

SetStatus

Sets the status flags of the measurement object. Please note, this value is interpreted as a bitfield where the bits have the following meaning. An option is considered enabled if the corresponding bit is set to 1.

bit0 0x00000001	Manually entered data
bit1 0x00000002	Tele-Measurement
bit2 0x00000004	Disturbance
bit3 0x00000008	Protection
bit4 0x00000010	Marked suspect
bit5 0x00000020	Violated constraint
bit6 0x00000040	On Event
bit7 0x00000080	Event Block.
bit8 0x00000100	Alarm Block.

bit9 0x00000200	Update Block.
bit10 0x00000400	Control Block.
bit29 0x20000000	Read
bit30 0x40000000	Write
bit31 0x80000000	Neglected by SE

```
None StaExtdatmea.SetStatus(int status, int dbSync)
```

ARGUMENTS

status Bitfield for status flags, see above

dbSync (optional)

0	New status flags are applied in memory only
1	Default, new status flags are stored on db (persistent)

SetStatusBit

Sets specific bits in the status bitfield. See StaExtdatmea.SetStatus() for details on the status bits.

```
None StaExtdatmea.SetStatusBit(int mask, int dbSync)
```

ARGUMENTS

mask Mask of bits to set to 1. A bit is unchanged if already set before.

dbSync (optional)

0	New status flags are applied in memory only
1	Default, new status flags are stored on db (persistent)

SetStatusBitTmp

Sets specific bits in the temporary (in memory) status bitfield. See StaExtdatmea.SetStatus() for details on the status bits.

```
None StaExtdatmea.SetStatusBitTmp(int mask)
```

ARGUMENTS

mask Mask of bits to set to 1. A bit is unchanged if already set before.

SetStatusTmp

Sets the temporary (in memory) status flags of the measurement object. This temporary value is used during calculations so that changes do not lead to object modifications and initial value remains unchanged.

Please note, this value is interpreted as a bitfield. See StaExtdatmea.SetStatus() for details on the status bits.

```
None StaExtdatmea.SetStatusTmp(int status)
```

ARGUMENTS

status	Bitfield for status flags, see above

UpdateControl

Transfers the value of current measurement object to the controller object (target object 'pCtrl' and target attribute 'varName'). If target object is a command, it is automatically executed afterwards.
Note: Calculation results will not be reset by this value transfer.

```
int StaExtdatmea.UpdateControl(int dbSync)
```

ARGUMENTS

dbSync (optional)

0	Value is copied in memory only
1	Default, copied value is stored on db (persistent)

RETURNS

0	on success
1	on error, e.g. target object does not have an attribute with given name

UpdateCtrl

Transfers the value of current measurement object to the controlled object (target object 'pObject' and target attribute 'variabName'). If target object is a command, it is automatically executed afterwards.
Note: Calculation results will not be reset by this value transfer.

```
int StaExtdatmea.UpdateCtrl(int dbSync)
```

ARGUMENTS

dbSync (optional)

0	Value is copied in memory only
1	Default, copied value is stored on db (persistent)

RETURNS

0	on success
1	on error, e.g. target object does not have an attribute with given name

4.3.6 StaExtfmea

Overview

CopyExtMeaStatusToStatusTmp

Copies the (persistent) status of current measurement object to temporary (in memory) status.

```
None StaExtfmea.CopyExtMeaStatusToStatusTmp()
```

GetMeaValue

Returns the value for frequency currently stored in the measurement object.

```
[int error,
float value] StaExtfmea.GetMeaValue()
```

ARGUMENTS

value (out)
Value for frequency.

RETURNS

Return value has no meaning. It is always 0.

GetStatus

Returns the status flags. Please note, this value is interpreted as a bitfield. See StaExtfmea.SetStatus() for details on the status bits.

```
int StaExtfmea.GetStatuts()
```

RETURNS

Status bitfield as an integer value.

GetStatusTmp

Returns the temporary (in memory) status flags. Please note, this value is interpreted as a bitfield. See StaExtfmea.SetStatus() for details on the status bits.

```
int StaExtfmea.GetStatusTmp()
```

RETURNS

Status bitfield as an integer value.

InitTmp

Initialises the temporary (in memory) fields of the measurement object with the values currently stored in the corresponding persistent fields. This affects temporary measurement value and temporary status fields. The temporary measurement value is used internally for comparison of new and old values for deadband violation. The temporary status is used during calculation in order to not modify initial value.

This function should be called once after the link has been established and before the calculation loop is executed.

```
None StaExtfmea.InitTmp()
```

IsStatusBitSet

Checks if specific bit(s) are set in the status bitfield. See StaExtfmea.SetStatus() for details on the status bits.

```
int StaExtfmea.IsStatusBitSet(int mask)
```

RETURNS

0	if at least one bit in mask is not set
1	if all bit(s) in mask are set

IsStatusBitSetTmp

Checks if specific bit(s) are set in the temporary (in memory) status bitfield. See StaExtfmea.SetStatus() for details on the status bits.

```
int StaExtfmea.IsStatusBitSetTmp(int mask)
```

RETURNS

0	if at least one bit in mask is not set
1	if all bit(s) in mask are set

ResetStatusBit

Resets specific bits in the status bitfield. See StaExtfmea.SetStatus() for details on the status bits.

```
None StaExtfmea.ResetStatusBit(int mask, int dbSync)
```

ARGUMENTS

mask	Mask of bits to set to 0. A bit is unchanged if already unset before.

dbSync (optional)

0	New status flags are applied in memory only
1	Default, new status flags are stored on db (persistent)

ResetStatusBitTmp

Resets specific bits in the temporary (in memory) status bitfield. See StaExtfmea.SetStatus() for details on the status bits.

```
None StaExtfmea.ResetStatusBitTmp(int mask)
```

ARGUMENTS

mask	Mask of bits to set to 0. A bit is unchanged if already unset before.

SetMeaValue

Sets the value stored in the measurement object.

```
int StaExtfmea.SetMeaValue(float value)
```

ARGUMENTS

value	New value.

RETURNS

Return value has no meaning. It is always 0.

SetStatus

Sets the status flags of the measurement object. Please note, this value is interpreted as a bitfield where the bits have the following meaning. An option is considered enabled if the corresponding bit is set to 1.

bit0 0x00000001	Manually entered data
bit1 0x00000002	Tele-Measurement
bit2 0x00000004	Disturbance
bit3 0x00000008	Protection
bit4 0x00000010	Marked suspect
bit5 0x00000020	Violated constraint
bit6 0x00000040	On Event
bit7 0x00000080	Event Block.
bit8 0x00000100	Alarm Block.

bit9 0x00000200 Update Block.

bit10 0x00000400 Control Block.

bit29 0x20000000 Read

bit30 0x40000000 Write

bit31 0x80000000 Neglected by SE

```
None StaExtfmea.SetStatus(int status, int dbSync)
```

ARGUMENTS

status Bitfield for status flags, see above

dbSync (optional)

0	New status flags are applied in memory only
1	Default, new status flags are stored on db (persistent)

SetStatusBit

Sets specific bits in the status bitfield. See StaExtfmea.SetStatus() for details on the status bits.

```
None StaExtfmea.SetStatusBit(int mask, int dbSync)
```

ARGUMENTS

mask Mask of bits to set to 1. A bit is unchanged if already set before.

dbSync (optional)

0	New status flags are applied in memory only
1	Default, new status flags are stored on db (persistent)

SetStatusBitTmp

Sets specific bits in the temporary (in memory) status bitfield. See StaExtfmea.SetStatus() for details on the status bits.

```
None StaExtfmea.SetStatusBitTmp(int mask)
```

ARGUMENTS

mask Mask of bits to set to 1. A bit is unchanged if already set before.

SetStatusTmp

Sets the temporary (in memory) status flags of the measurement object. This temporary value is used during calculations so that changes do not lead to object modifications and initial value remains unchanged.

Please note, this value is interpreted as a bitfield. See StaExtfmea.SetStatus() for details on the status bits.

```
None StaExtfmea.SetStatusTmp(int status)
```

ARGUMENTS

status	Bitfield for status flags, see above

UpdateControl

Transfers the value of current measurement object to the controller object (target object 'pCtrl' and target attribute 'varName'). If target object is a command, it is automatically executed afterwards.
Note: Calculation results will not be reset by this value transfer.

```
int StaExtfmea.UpdateControl(int dbSync)
```

ARGUMENTS

dbSync (optional)

0	Value is copied in memory only
1	Default, copied value is stored on db (persistent)

RETURNS

0	on success
1	on error, e.g. target object does not have an attribute with given name

UpdateCtrl

Transfers the value of current measurement object to the controlled object (target object 'pObject' and target attribute 'variabName'). If target object is a command, it is automatically executed afterwards.
Note: Calculation results will not be reset by this value transfer.

```
int StaExtfmea.UpdateCtrl(int dbSync)
```

ARGUMENTS

dbSync (optional)

0	Value is copied in memory only
1	Default, copied value is stored on db (persistent)

RETURNS

0	on success
1	on error, e.g. target object does not have an attribute with given name

4.3.7 StaExtfuelmea

Overview

CopyExtMeaStatusToStatusTmp

Copies the (persistent) status of current measurement object to temporary (in memory) status.

```
None StaExtfuelmea.CopyExtMeaStatusToStatusTmp()
```

GetMeaValue

Returns the value for fuel currently stored in the measurement object.

```
[int error,
float value] StaExtfuelmea.GetMeaValue(int unused,
                                       int applyMultiplicator)
```

Arguments

value (out)
Value for fuel, optionally multiplied by configured multiplicator

unused Not used.

applyMultiplicator
If 1 (default), returned value will be multiplied by the multiplicator stored in the measurement object. If 0, raw value will be returned.

Returns

Return value has no meaning. It is always 0.

GetStatus

Returns the status flags. Please note, this value is interpreted as a bitfield. See StaExtfuelmea.SetStatus() for details on the status bits.

```
int StaExtfuelmea.GetStatuts()
```

Returns

Status bitfield as an integer value.

GetStatusTmp

Returns the temporary (in memory) status flags. Please note, this value is interpreted as a bitfield. See StaExtfuelmea.SetStatus() for details on the status bits.

```
int StaExtfuelmea.GetStatusTmp()
```

Returns

Status bitfield as an integer value.

InitTmp

Initialises the temporary (in memory) fields of the measurement object with the values currently stored in the corresponding persistent fields. This affects temporary measurement value and temporary status fields. The temporary measurement value is used internally for comparison of new and old values for deadband violation. The temporary status is used during calculation in order to not modify initial value.

This function should be called once after the link has been established and before the calculation loop is executed.

```
None StaExtfuelmea.InitTmp()
```

IsStatusBitSet

Checks if specific bit(s) are set in the status bitfield. See StaExtfuelmea.SetStatus() for details on the status bits.

```
int StaExtfuelmea.IsStatusBitSet(int mask)
```

Returns

0	if at least one bit in mask is not set
1	if all bit(s) in mask are set

IsStatusBitSetTmp

Checks if specific bit(s) are set in the temporary (in memory) status bitfield. See StaExtfuelmea.SetStatus() for details on the status bits.

```
int StaExtfuelmea.IsStatusBitSetTmp(int mask)
```

Returns

0	if at least one bit in mask is not set
1	if all bit(s) in mask are set

ResetStatusBit

Resets specific bits in the status bitfield. See StaExtfuelmea.SetStatus() for details on the status bits.

```
None StaExtfuelmea.ResetStatusBit(int mask, int dbSync)
```

ARGUMENTS

mask Mask of bits to set to 0. A bit is unchanged if already unset before.

dbSync (optional)

0	New status flags are applied in memory only
1	Default, new status flags are stored on db (persistent)

ResetStatusBitTmp

Resets specific bits in the temporary (in memory) status bitfield. See StaExtfuelmea.SetStatus() for details on the status bits.

```
None StaExtfuelmea.ResetStatusBitTmp(int mask)
```

ARGUMENTS

mask Mask of bits to set to 0. A bit is unchanged if already unset before.

SetMeaValue

Sets the value stored in the measurement object.

```
int StaExtfuelmea.SetMeaValue(float value)
```

ARGUMENTS

value New value.

RETURNS

Return value has no meaning. It is always 0.

SetStatus

Sets the status flags of the measurement object. Please note, this value is interpreted as a bitfield where the bits have the following meaning. An option is considered enabled if the corresponding bit is set to 1.

bit0 0x00000001	Manually entered data
bit1 0x00000002	Tele-Measurement
bit2 0x00000004	Disturbance
bit3 0x00000008	Protection
bit4 0x00000010	Marked suspect
bit5 0x00000020	Violated constraint

bit6 0x00000040	On Event
bit7 0x00000080	Event Block.
bit8 0x00000100	Alarm Block.
bit9 0x00000200	Update Block.
bit10 0x00000400	Control Block.
bit29 0x20000000	Read
bit30 0x40000000	Write
bit31 0x80000000	Neglected by SE

```
None StaExtfuelmea.SetStatus(int status, int dbSync)
```

ARGUMENTS

status Bitfield for status flags, see above

dbSync (optional)

0	New status flags are applied in memory only
1	Default, new status flags are stored on db (persistent)

SetStatusBit

Sets specific bits in the status bitfield. See StaExtfuelmea.SetStatus() for details on the status bits.

```
None StaExtfuelmea.SetStatusBit(int mask, int dbSync)
```

ARGUMENTS

mask Mask of bits to set to 1. A bit is unchanged if already set before.

dbSync (optional)

0	New status flags are applied in memory only
1	Default, new status flags are stored on db (persistent)

SetStatusBitTmp

Sets specific bits in the temporary (in memory) status bitfield. See StaExtfuelmea.SetStatus() for details on the status bits.

```
None StaExtfuelmea.SetStatusBitTmp(int mask)
```

ARGUMENTS

mask Mask of bits to set to 1. A bit is unchanged if already set before.

SetStatusTmp

Sets the temporary (in memory) status flags of the measurement object. This temporary value is used during calculations so that changes do not lead to object modifications and initial value remains unchanged.

Please note, this value is interpreted as a bitfield. See StaExtfuelmea.SetStatus() for details on the status bits.

```
None StaExtfuelmea.SetStatusTmp(int status)
```

ARGUMENTS

status	Bitfield for status flags, see above

UpdateControl

Transfers the value of current measurement object to the controller object (target object 'pCtrl' and target attribute 'varName'). If target object is a command, it is automatically executed afterwards.
Note: Calculation results will not be reset by this value transfer.

```
int StaExtfuelmea.UpdateControl(int dbSync)
```

ARGUMENTS

dbSync (optional)

0	Value is copied in memory only
1	Default, copied value is stored on db (persistent)

RETURNS

0	on success
1	on error, e.g. target object does not have an attribute with given name

UpdateCtrl

Transfers the value of current measurement object to the controlled object (target object 'pObject' and target attribute 'variabName'). If target object is a command, it is automatically executed afterwards.
Note: Calculation results will not be reset by this value transfer.

```
int StaExtfuelmea.UpdateCtrl(int dbSync)
```

ARGUMENTS

dbSync (optional)

0	Value is copied in memory only
1	Default, copied value is stored on db (persistent)

RETURNS

0	on success
1	on error, e.g. target object does not have an attribute with given name

4.3.8 StaExtimea

Overview

CopyExtMeaStatusToStatusTmp
GetMeaValue
GetStatus
GetStatusTmp
InitTmp
IsStatusBitSet
IsStatusBitSetTmp
ResetStatusBit
ResetStatusBitTmp
SetMeaValue
SetStatus
SetStatusBit
SetStatusBitTmp
SetStatusTmp
UpdateControl
UpdateCtrl

CopyExtMeaStatusToStatusTmp

Copies the (persistent) status of current measurement object to temporary (in memory) status.

```
None StaExtimea.CopyExtMeaStatusToStatusTmp()
```

GetMeaValue

Returns the value for current currently stored in the measurement object.

```
[int error,
float value] StaExtimea.GetMeaValue(int unused,
                                    int applyMultiplicator)
```

ARGUMENTS

value (out)
Value for current, optionally multiplied by configured multiplicator

unused Not used.

applyMultiplicator
If 1 (default), returned value will be multiplied by the multiplicator stored in the measurement object. If 0, raw value will be returned.

RETURNS

Return value has no meaning. It is always 0.

GetStatus

Returns the status flags. Please note, this value is interpreted as a bitfield. See StaExtimea.SetStatus() for details on the status bits.

```
int StaExtimea.GetStatuts()
```

RETURNS

Status bitfield as an integer value.

GetStatusTmp

Returns the temporary (in memory) status flags. Please note, this value is interpreted as a bitfield. See StaExtimea.SetStatus() for details on the status bits.

```
int StaExtimea.GetStatusTmp()
```

RETURNS

Status bitfield as an integer value.

InitTmp

Initialises the temporary (in memory) fields of the measurement object with the values currently stored in the corresponding persistent fields. This affects temporary measurement value and temporary status fields. The temporary measurement value is used internally for comparison of new and old values for deadband violation. The temporary status is used during calculation in order to not modify initial value.

This function should be called once after the link has been established and before the calculation loop is executed.

```
None StaExtimea.InitTmp()
```

IsStatusBitSet

Checks if specific bit(s) are set in the status bitfield. See StaExtimea.SetStatus() for details on the status bits.

```
int StaExtimea.IsStatusBitSet(int mask)
```

RETURNS

0	if at least one bit in mask is not set
1	if all bit(s) in mask are set

IsStatusBitSetTmp

Checks if specific bit(s) are set in the temporary (in memory) status bitfield. See StaExtimea.SetStatus() for details on the status bits.

```
int StaExtimea.IsStatusBitSetTmp(int mask)
```

RETURNS

0	if at least one bit in mask is not set
1	if all bit(s) in mask are set

ResetStatusBit

Resets specific bits in the status bitfield. See StaExtimea.SetStatus() for details on the status bits.

```
None StaExtimea.ResetStatusBit(int mask, int dbSync)
```

ARGUMENTS

mask Mask of bits to set to 0. A bit is unchanged if already unset before.

dbSync (optional)

0	New status flags are applied in memory only
1	Default, new status flags are stored on db (persistent)

ResetStatusBitTmp

Resets specific bits in the temporary (in memory) status bitfield. See StaExtimea.SetStatus() for details on the status bits.

```
None StaExtimea.ResetStatusBitTmp(int mask)
```

ARGUMENTS

mask Mask of bits to set to 0. A bit is unchanged if already unset before.

SetMeaValue

Sets the value stored in the measurement object.

```
int StaExtimea.SetMeaValue(float value)
```

ARGUMENTS

value New value.

RETURNS

Return value has no meaning. It is always 0.

SetStatus

Sets the status flags of the measurement object. Please note, this value is interpreted as a bitfield where the bits have the following meaning. An option is considered enabled if the corresponding bit is set to 1.

bit0 0x00000001	Manually entered data
bit1 0x00000002	Tele-Measurement
bit2 0x00000004	Disturbance
bit3 0x00000008	Protection
bit4 0x00000010	Marked suspect
bit5 0x00000020	Violated constraint

bit6 0x00000040 On Event

bit7 0x00000080 Event Block.

bit8 0x00000100 Alarm Block.

bit9 0x00000200 Update Block.

bit10 0x00000400 Control Block.

bit29 0x20000000 Read

bit30 0x40000000 Write

bit31 0x80000000 Neglected by SE

```
None StaExtimea.SetStatus(int status, int dbSync)
```

ARGUMENTS

status Bitfield for status flags, see above

dbSync (optional)

0	New status flags are applied in memory only
1	Default, new status flags are stored on db (persistent)

SetStatusBit

Sets specific bits in the status bitfield. See StaExtimea.SetStatus() for details on the status bits.

```
None StaExtimea.SetStatusBit(int mask, int dbSync)
```

ARGUMENTS

mask Mask of bits to set to 1. A bit is unchanged if already set before.

dbSync (optional)

0	New status flags are applied in memory only
1	Default, new status flags are stored on db (persistent)

SetStatusBitTmp

Sets specific bits in the temporary (in memory) status bitfield. See StaExtimea.SetStatus() for details on the status bits.

```
None StaExtimea.SetStatusBitTmp(int mask)
```

ARGUMENTS

mask Mask of bits to set to 1. A bit is unchanged if already set before.

SetStatusTmp

Sets the temporary (in memory) status flags of the measurement object. This temporary value is used during calculations so that changes do not lead to object modifications and initial value remains unchanged.

Please note, this value is interpreted as a bitfield. See StaExtimea.SetStatus() for details on the status bits.

```
None StaExtimea.SetStatusTmp(int status)
```

ARGUMENTS

status	Bitfield for status flags, see above

UpdateControl

Transfers the value of current measurement object to the controller object (target object 'pCtrl' and target attribute 'varName'). If target object is a command, it is automatically executed afterwards.
Note: Calculation results will not be reset by this value transfer.

```
int StaExtimea.UpdateControl(int dbSync)
```

ARGUMENTS

dbSync (optional)

0	Value is copied in memory only
1	Default, copied value is stored on db (persistent)

RETURNS

0	on success
1	on error, e.g. target object does not have an attribute with given name

UpdateCtrl

Transfers the value of current measurement object to the controlled object (target object 'pObject' and target attribute 'variabName'). If target object is a command, it is automatically executed afterwards.
Note: Calculation results will not be reset by this value transfer.

```
int StaExtimea.UpdateCtrl(int dbSync)
```

ARGUMENTS

dbSync (optional)

0	Value is copied in memory only
1	Default, copied value is stored on db (persistent)

RETURNS

0	on success
1	on error, e.g. target object does not have an attribute with given name

4.3.9 StaExtpfmea

Overview

CopyExtMeaStatusToStatusTmp

Copies the (persistent) status of current measurement object to temporary (in memory) status.

```
None StaExtpfmea.CopyExtMeaStatusToStatusTmp()
```

GetMeaValue

Returns the value for power factor currently stored in the measurement object.

```
[int error,
float value] StaExtpfmea.GetMeaValue(int unused,
                                     int applyMultiplicator)
```

ARGUMENTS

value (out)
Value for current, optionally multiplied by configured multiplicator

unused Not used.

applyMultiplicator
If 1 (default), returned value will be multiplied by the multiplicator stored in the measurement object. If 0, raw value will be returned.

RETURNS

Return value has no meaning. It is always 0.

GetStatus

Returns the status flags. Please note, this value is interpreted as a bitfield. See StaExtpfmea.SetStatus() for details on the status bits.

```
int StaExtpfmea.GetStatuts()
```

RETURNS

Status bitfield as an integer value.

GetStatusTmp

Returns the temporary (in memory) status flags. Please note, this value is interpreted as a bitfield. See StaExtpfmea.SetStatus() for details on the status bits.

```
int StaExtpfmea.GetStatusTmp()
```

RETURNS

Status bitfield as an integer value.

InitTmp

Initialises the temporary (in memory) fields of the measurement object with the values currently stored in the corresponding persistent fields. This affects temporary measurement value and temporary status fields. The temporary measurement value is used internally for comparison of new and old values for deadband violation. The temporary status is used during calculation in order to not modify initial value.

This function should be called once after the link has been established and before the calculation loop is executed.

```
None StaExtpfmea.InitTmp()
```

IsStatusBitSet

Checks if specific bit(s) are set in the status bitfield. See StaExtpfmea.SetStatus() for details on the status bits.

```
int StaExtpfmea.IsStatusBitSet(int mask)
```

RETURNS

0	if at least one bit in mask is not set
1	if all bit(s) in mask are set

IsStatusBitSetTmp

Checks if specific bit(s) are set in the temporary (in memory) status bitfield. See StaExtpfmea.SetStatus() for details on the status bits.

```
int StaExtpfmea.IsStatusBitSetTmp(int mask)
```

RETURNS

0	if at least one bit in mask is not set
1	if all bit(s) in mask are set

ResetStatusBit

Resets specific bits in the status bitfield. See StaExtpfmea.SetStatus() for details on the status bits.

```
None StaExtpfmea.ResetStatusBit(int mask, int dbSync)
```

ARGUMENTS

mask Mask of bits to set to 0. A bit is unchanged if already unset before.

dbSync (optional)

0	New status flags are applied in memory only
1	Default, new status flags are stored on db (persistent)

ResetStatusBitTmp

Resets specific bits in the temporary (in memory) status bitfield. See StaExtpfmea.SetStatus() for details on the status bits.

```
None StaExtpfmea.ResetStatusBitTmp(int mask)
```

ARGUMENTS

mask Mask of bits to set to 0. A bit is unchanged if already unset before.

SetMeaValue

Sets the value stored in the measurement object.

```
int StaExtpfmea.SetMeaValue(float value)
```

ARGUMENTS

value New value.

RETURNS

Return value has no meaning. It is always 0.

SetStatus

Sets the status flags of the measurement object. Please note, this value is interpreted as a bitfield where the bits have the following meaning. An option is considered enabled if the corresponding bit is set to 1.

bit0 0x00000001	Manually entered data
bit1 0x00000002	Tele-Measurement
bit2 0x00000004	Disturbance
bit3 0x00000008	Protection
bit4 0x00000010	Marked suspect
bit5 0x00000020	Violated constraint

bit6 0x00000040	On Event
bit7 0x00000080	Event Block.
bit8 0x00000100	Alarm Block.
bit9 0x00000200	Update Block.
bit10 0x00000400	Control Block.
bit29 0x20000000	Read
bit30 0x40000000	Write
bit31 0x80000000	Neglected by SE

```
None StaExtpfmea.SetStatus(int status, int dbSync)
```

ARGUMENTS

status Bitfield for status flags, see above

dbSync (optional)

0	New status flags are applied in memory only
1	Default, new status flags are stored on db (persistent)

SetStatusBit

Sets specific bits in the status bitfield. See StaExtpfmea.SetStatus() for details on the status bits.

```
None StaExtpfmea.SetStatusBit(int mask, int dbSync)
```

ARGUMENTS

mask Mask of bits to set to 1. A bit is unchanged if already set before.

dbSync (optional)

0	New status flags are applied in memory only
1	Default, new status flags are stored on db (persistent)

SetStatusBitTmp

Sets specific bits in the temporary (in memory) status bitfield. See StaExtpfmea.SetStatus() for details on the status bits.

```
None StaExtpfmea.SetStatusBitTmp(int mask)
```

ARGUMENTS

mask Mask of bits to set to 1. A bit is unchanged if already set before.

SetStatusTmp

Sets the temporary (in memory) status flags of the measurement object. This temporary value is used during calculations so that changes do not lead to object modifications and initial value remains unchanged.

Please note, this value is interpreted as a bitfield. See StaExtpfmea.SetStatus() for details on the status bits.

```
None StaExtpfmea.SetStatusTmp(int status)
```

ARGUMENTS

status	Bitfield for status flags, see above

UpdateControl

Transfers the value of current measurement object to the controller object (target object 'pCtrl' and target attribute 'varName'). If target object is a command, it is automatically executed afterwards.
Note: Calculation results will not be reset by this value transfer.

```
int StaExtpfmea.UpdateControl(int dbSync)
```

ARGUMENTS

dbSync (optional)

0	Value is copied in memory only
1	Default, copied value is stored on db (persistent)

RETURNS

0	on success
1	on error, e.g. target object does not have an attribute with given name

UpdateCtrl

Transfers the value of current measurement object to the controlled object (target object 'pObject' and target attribute 'variabName'). If target object is a command, it is automatically executed afterwards.
Note: Calculation results will not be reset by this value transfer.

```
int StaExtpfmea.UpdateCtrl(int dbSync)
```

ARGUMENTS

dbSync (optional)

0	Value is copied in memory only
1	Default, copied value is stored on db (persistent)

RETURNS

0	on success
1	on error, e.g. target object does not have an attribute with given name

4.3.10 StaExtpmea

Overview

CopyExtMeaStatusToStatusTmp

Copies the (persistent) status of current measurement object to temporary (in memory) status.

```
None StaExtpmea.CopyExtMeaStatusToStatusTmp()
```

GetMeaValue

Returns the value for active power stored in the measurement object.

```
[int error,
float value] StaExtpmea.GetMeaValue(int unused,
                                    int applyMultiplicator)
```

ARGUMENTS

value (out)
Value for active power, optionally multiplied by configured multiplicator

unused Not used.

applyMultiplicator
If 1 (default), returned value will be multiplied by the multiplicator stored in the measurement object. If 0, raw value will be returned.

RETURNS

Return value has no meaning. It is always 0.

GetStatus

Returns the status flags. Please note, this value is interpreted as a bitfield. See StaExtpmea.SetStatus() for details on the status bits.

```
int StaExtpmea.GetStatuts()
```

RETURNS

Status bitfield as an integer value.

GetStatusTmp

Returns the temporary (in memory) status flags. Please note, this value is interpreted as a bitfield. See StaExtpmea.SetStatus() for details on the status bits.

```
int StaExtpmea.GetStatusTmp()
```

RETURNS

Status bitfield as an integer value.

InitTmp

Initialises the temporary (in memory) fields of the measurement object with the values currently stored in the corresponding persistent fields. This affects temporary measurement value and temporary status fields. The temporary measurement value is used internally for comparison of new and old values for deadband violation. The temporary status is used during calculation in order to not modify initial value.

This function should be called once after the link has been established and before the calculation loop is executed.

```
None StaExtpmea.InitTmp()
```

IsStatusBitSet

Checks if specific bit(s) are set in the status bitfield. See StaExtpmea.SetStatus() for details on the status bits.

```
int StaExtpmea.IsStatusBitSet(int mask)
```

RETURNS

0	if at least one bit in mask is not set
1	if all bit(s) in mask are set

IsStatusBitSetTmp

Checks if specific bit(s) are set in the temporary (in memory) status bitfield. See StaExtpmea.SetStatus() for details on the status bits.

```
int StaExtpmea.IsStatusBitSetTmp(int mask)
```

RETURNS

0	if at least one bit in mask is not set
1	if all bit(s) in mask are set

ResetStatusBit

Resets specific bits in the status bitfield. See StaExtpmea.SetStatus() for details on the status bits.

```
None StaExtpmea.ResetStatusBit(int mask, int dbSync)
```

ARGUMENTS

mask Mask of bits to set to 0. A bit is unchanged if already unset before.

dbSync (optional)

0	New status flags are applied in memory only
1	Default, new status flags are stored on db (persistent)

ResetStatusBitTmp

Resets specific bits in the temporary (in memory) status bitfield. See StaExtpmea.SetStatus() for details on the status bits.

```
None StaExtpmea.ResetStatusBitTmp(int mask)
```

ARGUMENTS

mask Mask of bits to set to 0. A bit is unchanged if already unset before.

SetMeaValue

Sets the value stored in the measurement object.

```
int StaExtpmea.SetMeaValue(float value)
```

ARGUMENTS

value New value.

RETURNS

Return value has no meaning. It is always 0.

SetStatus

Sets the status flags of the measurement object. Please note, this value is interpreted as a bitfield where the bits have the following meaning. An option is considered enabled if the corresponding bit is set to 1.

bit0 0x00000001	Manually entered data
bit1 0x00000002	Tele-Measurement
bit2 0x00000004	Disturbance
bit3 0x00000008	Protection
bit4 0x00000010	Marked suspect
bit5 0x00000020	Violated constraint

bit6 0x00000040 On Event

bit7 0x00000080 Event Block.

bit8 0x00000100 Alarm Block.

bit9 0x00000200 Update Block.

bit10 0x00000400 Control Block.

bit29 0x20000000 Read

bit30 0x40000000 Write

bit31 0x80000000 Neglected by SE

```
None StaExtpmea.SetStatus(int status, int dbSync)
```

ARGUMENTS

status Bitfield for status flags, see above

dbSync (optional)

0	New status flags are applied in memory only
1	Default, new status flags are stored on db (persistent)

SetStatusBit

Sets specific bits in the status bitfield. See StaExtpmea.SetStatus() for details on the status bits.

```
None StaExtpmea.SetStatusBit(int mask, int dbSync)
```

ARGUMENTS

mask Mask of bits to set to 1. A bit is unchanged if already set before.

dbSync (optional)

0	New status flags are applied in memory only
1	Default, new status flags are stored on db (persistent)

SetStatusBitTmp

Sets specific bits in the temporary (in memory) status bitfield. See StaExtpmea.SetStatus() for details on the status bits.

```
None StaExtpmea.SetStatusBitTmp(int mask)
```

ARGUMENTS

mask Mask of bits to set to 1. A bit is unchanged if already set before.

SetStatusTmp

Sets the temporary (in memory) status flags of the measurement object. This temporary value is used during calculations so that changes do not lead to object modifications and initial value remains unchanged.

Please note, this value is interpreted as a bitfield. See StaExtpmea.SetStatus() for details on the status bits.

```
None StaExtpmea.SetStatusTmp(int status)
```

ARGUMENTS

status	Bitfield for status flags, see above

UpdateControl

Transfers the value of current measurement object to the controller object (target object 'pCtrl' and target attribute 'varName'). If target object is a command, it is automatically executed afterwards.
Note: Calculation results will not be reset by this value transfer.

```
int StaExtpmea.UpdateControl(int dbSync)
```

ARGUMENTS

dbSync (optional)

0	Value is copied in memory only
1	Default, copied value is stored on db (persistent)

RETURNS

0	on success
1	on error, e.g. target object does not have an attribute with given name

UpdateCtrl

Transfers the value of current measurement object to the controlled object (target object 'pObject' and target attribute 'variabName'). If target object is a command, it is automatically executed afterwards.
Note: Calculation results will not be reset by this value transfer.

```
int StaExtpmea.UpdateCtrl(int dbSync)
```

ARGUMENTS

dbSync (optional)

0	Value is copied in memory only
1	Default, copied value is stored on db (persistent)

RETURNS

0	on success
1	on error, e.g. target object does not have an attribute with given name

4.3.11 StaExtqmea

Overview

CopyExtMeaStatusToStatusTmp

Copies the (persistent) status of current measurement object to temporary (in memory) status.

```
None StaExtqmea.CopyExtMeaStatusToStatusTmp()
```

GetMeaValue

Returns the value for reactive power currently stored in the measurement object.

```
[int error,
float value] StaExtqmea.GetMeaValue(int unused,
                                    int applyMultiplicator)
```

ARGUMENTS

value (out)
Value for reactive power, optionally multiplied by configured multiplicator

unused Not used.

applyMultiplicator
If 1 (default), returned value will be multiplied by the multiplicator stored in the measurement object. If 0, raw value will be returned.

RETURNS

Return value has no meaning. It is always 0.

GetStatus

Returns the status flags. Please note, this value is interpreted as a bitfield. See StaExtqmea.SetStatus() for details on the status bits.

```
int StaExtqmea.GetStatuts()
```

RETURNS

Status bitfield as an integer value.

GetStatusTmp

Returns the temporary (in memory) status flags. Please note, this value is interpreted as a bitfield. See StaExtqmea.SetStatus() for details on the status bits.

```
int StaExtqmea.GetStatusTmp()
```

RETURNS

Status bitfield as an integer value.

InitTmp

Initialises the temporary (in memory) fields of the measurement object with the values currently stored in the corresponding persistent fields. This affects temporary measurement value and temporary status fields. The temporary measurement value is used internally for comparison of new and old values for deadband violation. The temporary status is used during calculation in order to not modify initial value.

This function should be called once after the link has been established and before the calculation loop is executed.

```
None StaExtqmea.InitTmp()
```

IsStatusBitSet

Checks if specific bit(s) are set in the status bitfield. See StaExtqmea.SetStatus() for details on the status bits.

```
int StaExtqmea.IsStatusBitSet(int mask)
```

RETURNS

0	if at least one bit in mask is not set
1	if all bit(s) in mask are set

IsStatusBitSetTmp

Checks if specific bit(s) are set in the temporary (in memory) status bitfield. See StaExtqmea.SetStatus() for details on the status bits.

```
int StaExtqmea.IsStatusBitSetTmp(int mask)
```

RETURNS

0	if at least one bit in mask is not set
1	if all bit(s) in mask are set

ResetStatusBit

Resets specific bits in the status bitfield. See StaExtqmea.SetStatus() for details on the status bits.

```
None StaExtqmea.ResetStatusBit(int mask, int dbSync)
```

Arguments

mask Mask of bits to set to 0. A bit is unchanged if already unset before.

dbSync (optional)

0	New status flags are applied in memory only
1	Default, new status flags are stored on db (persistent)

ResetStatusBitTmp

Resets specific bits in the temporary (in memory) status bitfield. See StaExtqmea.SetStatus() for details on the status bits.

```
None StaExtqmea.ResetStatusBitTmp(int mask)
```

Arguments

mask Mask of bits to set to 0. A bit is unchanged if already unset before.

SetMeaValue

Sets the value stored in the measurement object.

```
int StaExtqmea.SetMeaValue(float value)
```

Arguments

value New value.

Returns

Return value has no meaning. It is always 0.

SetStatus

Sets the status flags of the measurement object. Please note, this value is interpreted as a bitfield where the bits have the following meaning. An option is considered enabled if the corresponding bit is set to 1.

bit0 0x00000001	Manually entered data
bit1 0x00000002	Tele-Measurement
bit2 0x00000004	Disturbance
bit3 0x00000008	Protection
bit4 0x00000010	Marked suspect
bit5 0x00000020	Violated constraint

bit6 0x00000040 On Event

bit7 0x00000080 Event Block.

bit8 0x00000100 Alarm Block.

bit9 0x00000200 Update Block.

bit10 0x00000400 Control Block.

bit29 0x20000000 Read

bit30 0x40000000 Write

bit31 0x80000000 Neglected by SE

```
None StaExtqmea.SetStatus(int status, int dbSync)
```

ARGUMENTS

status Bitfield for status flags, see above

dbSync (optional)

0	New status flags are applied in memory only
1	Default, new status flags are stored on db (persistent)

SetStatusBit

Sets specific bits in the status bitfield. See StaExtqmea.SetStatus() for details on the status bits.

```
None StaExtqmea.SetStatusBit(int mask, int dbSync)
```

ARGUMENTS

mask Mask of bits to set to 1. A bit is unchanged if already set before.

dbSync (optional)

0	New status flags are applied in memory only
1	Default, new status flags are stored on db (persistent)

SetStatusBitTmp

Sets specific bits in the temporary (in memory) status bitfield. See StaExtqmea.SetStatus() for details on the status bits.

```
None StaExtqmea.SetStatusBitTmp(int mask)
```

ARGUMENTS

mask Mask of bits to set to 1. A bit is unchanged if already set before.

SetStatusTmp

Sets the temporary (in memory) status flags of the measurement object. This temporary value is used during calculations so that changes do not lead to object modifications and initial value remains unchanged.

Please note, this value is interpreted as a bitfield. See StaExtqmea.SetStatus() for details on the status bits.

```
None StaExtqmea.SetStatusTmp(int status)
```

ARGUMENTS

status	Bitfield for status flags, see above

UpdateControl

Transfers the value of current measurement object to the controller object (target object 'pCtrl' and target attribute 'varName'). If target object is a command, it is automatically executed afterwards.
Note: Calculation results will not be reset by this value transfer.

```
int StaExtqmea.UpdateControl(int dbSync)
```

ARGUMENTS

dbSync (optional)

0	Value is copied in memory only
1	Default, copied value is stored on db (persistent)

RETURNS

0	on success
1	on error, e.g. target object does not have an attribute with given name

UpdateCtrl

Transfers the value of current measurement object to the controlled object (target object 'pObject' and target attribute 'variabName'). If target object is a command, it is automatically executed afterwards.
Note: Calculation results will not be reset by this value transfer.

```
int StaExtqmea.UpdateCtrl(int dbSync)
```

ARGUMENTS

dbSync (optional)

0	Value is copied in memory only
1	Default, copied value is stored on db (persistent)

RETURNS

0	on success
1	on error, e.g. target object does not have an attribute with given name

4.3.12 StaExtsmea

Overview

CopyExtMeaStatusToStatusTmp
GetStatus
GetStatusTmp
InitTmp
IsStatusBitSet
IsStatusBitSetTmp
ResetStatusBit
ResetStatusBitTmp
SetMeaValue
SetStatus
SetStatusBit
SetStatusBitTmp
SetStatusTmp
UpdateControl
UpdateCtrl

CopyExtMeaStatusToStatusTmp

Copies the (persistent) status of current measurement object to temporary (in memory) status.

```
None StaExtsmea.CopyExtMeaStatusToStatusTmp()
```

GetStatus

Returns the status flags. Please note, this value is interpreted as a bitfield. See StaExtsmea.SetStatus() for details on the status bits.

```
int StaExtsmea.GetStatuts()
```

RETURNS

Status bitfield as an integer value.

GetStatusTmp

Returns the temporary (in memory) status flags. Please note, this value is interpreted as a bitfield. See StaExtsmea.SetStatus() for details on the status bits.

```
int StaExtsmea.GetStatusTmp()
```

RETURNS

Status bitfield as an integer value.

InitTmp

Initialises the temporary (in memory) fields of the measurement object with the values currently stored in the corresponding persistent fields. This affects temporary measurement value and temporary status fields. The temporary measurement value is used internally for comparison of new and old values for deadband violation. The temporary status is used during calculation in order to not modify initial value.

This function should be called once after the link has been established and before the calculation loop is executed.

```
None StaExtsmea.InitTmp()
```

IsStatusBitSet

Checks if specific bit(s) are set in the status bitfield. See StaExtsmea.SetStatus() for details on the status bits.

```
int StaExtsmea.IsStatusBitSet(int mask)
```

RETURNS

0	if at least one bit in mask is not set
1	if all bit(s) in mask are set

IsStatusBitSetTmp

Checks if specific bit(s) are set in the temporary (in memory) status bitfield. See StaExtsmea.SetStatus() for details on the status bits.

```
int StaExtsmea.IsStatusBitSetTmp(int mask)
```

RETURNS

0	if at least one bit in mask is not set
1	if all bit(s) in mask are set

ResetStatusBit

Resets specific bits in the status bitfield. See StaExtsmea.SetStatus() for details on the status bits.

```
None StaExtsmea.ResetStatusBit(int mask, int dbSync)
```

ARGUMENTS

mask Mask of bits to set to 0. A bit is unchanged if already unset before.

dbSync (optional)

0	New status flags are applied in memory only
1	Default, new status flags are stored on db (persistent)

ResetStatusBitTmp

Resets specific bits in the temporary (in memory) status bitfield. See StaExtsmea.SetStatus() for details on the status bits.

```
None StaExtsmea.ResetStatusBitTmp(int mask)
```

ARGUMENTS

mask	Mask of bits to set to 0. A bit is unchanged if already unset before.

SetMeaValue

Sets the value stored in the measurement object.

```
int StaExtsmea.SetMeaValue(float value)
```

ARGUMENTS

value	New value.

RETURNS

Return value has no meaning. It is always 0.

SetStatus

Sets the status flags of the measurement object. Please note, this value is interpreted as a bitfield where the bits have the following meaning. An option is considered enabled if the corresponding bit is set to 1.

bit0 0x00000001	Manually entered data
bit1 0x00000002	Tele-Measurement
bit2 0x00000004	Disturbance
bit3 0x00000008	Protection
bit4 0x00000010	Marked suspect
bit5 0x00000020	Violated constraint
bit6 0x00000040	On Event
bit7 0x00000080	Event Block.
bit8 0x00000100	Alarm Block.
bit9 0x00000200	Update Block.
bit10 0x00000400	Control Block.
bit29 0x20000000	Read
bit30 0x40000000	Write
bit31 0x80000000	Neglected by SE

```
None StaExtsmea.SetStatus(int status, int dbSync)
```

ARGUMENTS

status	Bitfield for status flags, see above

dbSync (optional)

0	New status flags are applied in memory only
1	Default, new status flags are stored on db (persistent)

SetStatusBit

Sets specific bits in the status bitfield. See StaExtsmea.SetStatus() for details on the status bits.

```
None StaExtsmea.SetStatusBit(int mask, int dbSync)
```

ARGUMENTS

mask	Mask of bits to set to 1. A bit is unchanged if already set before.

dbSync (optional)

0	New status flags are applied in memory only
1	Default, new status flags are stored on db (persistent)

SetStatusBitTmp

Sets specific bits in the temporary (in memory) status bitfield. See StaExtsmea.SetStatus() for details on the status bits.

```
None StaExtsmea.SetStatusBitTmp(int mask)
```

ARGUMENTS

mask	Mask of bits to set to 1. A bit is unchanged if already set before.

SetStatusTmp

Sets the temporary (in memory) status flags of the measurement object. This temporary value is used during calculations so that changes do not lead to object modifications and initial value remains unchanged.

Please note, this value is interpreted as a bitfield. See StaExtsmea.SetStatus() for details on the status bits.

```
None StaExtsmea.SetStatusTmp(int status)
```

ARGUMENTS

status	Bitfield for status flags, see above

UpdateControl

Transfers the value of current measurement object to the controller object (target object 'pCtrl' and target attribute 'varName'). If target object is a command, it is automatically executed afterwards.
Note: Calculation results will not be reset by this value transfer.

```
int StaExtsmea.UpdateControl(int dbSync)
```

ARGUMENTS

dbSync (optional)

0	Value is copied in memory only
1	Default, copied value is stored on db (persistent)

RETURNS

0	on success
1	on error, e.g. target object does not have an attribute with given name

UpdateCtrl

Transfers the value of current measurement object to the controlled object (target object 'pObject' and target attribute 'variabName'). If target object is a command, it is automatically executed afterwards.
Note: Calculation results will not be reset by this value transfer.

```
int StaExtsmea.UpdateCtrl(int dbSync)
```

ARGUMENTS

dbSync (optional)

0	Value is copied in memory only
1	Default, copied value is stored on db (persistent)

RETURNS

0	on success
1	on error, e.g. target object does not have an attribute with given name

4.3.13 StaExttapmea

Overview

CopyExtMeaStatusToStatusTmp
GetMeaValue
GetStatus
GetStatusTmp
InitTmp
IsStatusBitSet
IsStatusBitSetTmp
ResetStatusBit
ResetStatusBitTmp
SetMeaValue
SetStatus

SetStatusBit
SetStatusBitTmp
SetStatusTmp
UpdateControl
UpdateCtrl

CopyExtMeaStatusToStatusTmp

Copies the (persistent) status of current measurement object to temporary (in memory) status.

```
None StaExttapmea.CopyExtMeaStatusToStatusTmp()
```

GetMeaValue

Returns the value for tap position and tap info currently stored in the measurement object.

```
[int error,
float value] StaExttapmea.GetMeaValue(int unused,
                                      int applyMultiplicator)
```

ARGUMENTS

value (out)
Value.

type type of value to return

0	tap position
1	operation mode
2	tap changer command
3	tap operation mode
4	tap operation mode command

useTranslationTable
Only supported if type=0 (tap step), if 1 (default) returned value will be translated according to given table. If 0 is passed, the raw value will be returned.

RETURNS

0	on success
1	on error, e.g. unsupported type

Returns 1 on erroReturn value has no meaning. It is always 0.

GetStatus

Returns the status flags. Please note, this value is interpreted as a bitfield. See StaExttapmea.SetStatus() for details on the status bits.

```
int StaExttapmea.GetStatuts()
```

RETURNS

Status bitfield as an integer value.

GetStatusTmp

Returns the temporary (in memory) status flags. Please note, this value is interpreted as a bitfield. See StaExttapmea.SetStatus() for details on the status bits.

```
int StaExttapmea.GetStatusTmp()
```

RETURNS

Status bitfield as an integer value.

InitTmp

Initialises the temporary (in memory) fields of the measurement object with the values currently stored in the corresponding persistent fields. This affects temporary measurement value and temporary status fields. The temporary measurement value is used internally for comparison of new and old values for deadband violation. The temporary status is used during calculation in order to not modify initial value.

This function should be called once after the link has been established and before the calculation loop is executed.

```
None StaExttapmea.InitTmp()
```

IsStatusBitSet

Checks if specific bit(s) are set in the status bitfield. See StaExttapmea.SetStatus() for details on the status bits.

```
int StaExttapmea.IsStatusBitSet(int mask)
```

RETURNS

0	if at least one bit in mask is not set
1	if all bit(s) in mask are set

IsStatusBitSetTmp

Checks if specific bit(s) are set in the temporary (in memory) status bitfield. See StaExttapmea.SetStatus() for details on the status bits.

```
int StaExttapmea.IsStatusBitSetTmp(int mask)
```

RETURNS

0	if at least one bit in mask is not set
1	if all bit(s) in mask are set

ResetStatusBit

Resets specific bits in the status bitfield. See StaExttapmea.SetStatus() for details on the status bits.

```
None StaExttapmea.ResetStatusBit(int mask, int dbSync)
```

ARGUMENTS

mask	Mask of bits to set to 0. A bit is unchanged if already unset before.

dbSync (optional)

0	New status flags are applied in memory only
1	Default, new status flags are stored on db (persistent)

ResetStatusBitTmp

Resets specific bits in the temporary (in memory) status bitfield. See StaExttapmea.SetStatus() for details on the status bits.

```
None StaExttapmea.ResetStatusBitTmp(int mask)
```

ARGUMENTS

mask	Mask of bits to set to 0. A bit is unchanged if already unset before.

SetMeaValue

Sets the value stored in the measurement object.

```
int StaExttapmea.SetMeaValue(float value)
```

ARGUMENTS

value	New value.

RETURNS

Return value has no meaning. It is always 0.

SetStatus

Sets the status flags of the measurement object. Please note, this value is interpreted as a bitfield where the bits have the following meaning. An option is considered enabled if the corresponding bit is set to 1.

bit0 0x00000001	Manually entered data
bit1 0x00000002	Tele-Measurement
bit2 0x00000004	Disturbance
bit3 0x00000008	Protection
bit4 0x00000010	Marked suspect
bit5 0x00000020	Violated constraint
bit6 0x00000040	On Event
bit7 0x00000080	Event Block.
bit8 0x00000100	Alarm Block.

bit9 0x00000200 Update Block.

bit10 0x00000400 Control Block.

bit29 0x20000000 Read

bit30 0x40000000 Write

bit31 0x80000000 Neglected by SE

```
None StaExttapmea.SetStatus(int status, int dbSync)
```

ARGUMENTS

status Bitfield for status flags, see above

dbSync (optional)

0	New status flags are applied in memory only
1	Default, new status flags are stored on db (persistent)

SetStatusBit

Sets specific bits in the status bitfield. See StaExttapmea.SetStatus() for details on the status bits.

```
None StaExttapmea.SetStatusBit(int mask, int dbSync)
```

ARGUMENTS

mask Mask of bits to set to 1. A bit is unchanged if already set before.

dbSync (optional)

0	New status flags are applied in memory only
1	Default, new status flags are stored on db (persistent)

SetStatusBitTmp

Sets specific bits in the temporary (in memory) status bitfield. See StaExttapmea.SetStatus() for details on the status bits.

```
None StaExttapmea.SetStatusBitTmp(int mask)
```

ARGUMENTS

mask Mask of bits to set to 1. A bit is unchanged if already set before.

SetStatusTmp

Sets the temporary (in memory) status flags of the measurement object. This temporary value is used during calculations so that changes do not lead to object modifications and initial value remains unchanged.

Please note, this value is interpreted as a bitfield. See StaExttapmea.SetStatus() for details on the status bits.

```
None StaExttapmea.SetStatusTmp(int status)
```

ARGUMENTS

status	Bitfield for status flags, see above

UpdateControl

Transfers the value of current measurement object to the controller object (target object 'pCtrl' and target attribute 'varName'). If target object is a command, it is automatically executed afterwards.
Note: Calculation results will not be reset by this value transfer.

```
int StaExttapmea.UpdateControl(int dbSync)
```

ARGUMENTS

dbSync (optional)

0	Value is copied in memory only
1	Default, copied value is stored on db (persistent)

RETURNS

0	on success
1	on error, e.g. target object does not have an attribute with given name

UpdateCtrl

Transfers the value of current measurement object to the controlled object (target object 'pObject' and target attribute 'variabName'). If target object is a command, it is automatically executed afterwards.
Note: Calculation results will not be reset by this value transfer.

```
int StaExttapmea.UpdateCtrl(int dbSync)
```

ARGUMENTS

dbSync (optional)

0	Value is copied in memory only
1	Default, copied value is stored on db (persistent)

RETURNS

0	on success
1	on error, e.g. target object does not have an attribute with given name

4.3.14 StaExtv3mea

Overview

CopyExtMeaStatusToStatusTmp

Copies the (persistent) status of current measurement object to temporary (in memory) status.

```
None StaExtv3mea.CopyExtMeaStatusToStatusTmp()
```

GetMeaValue

Returns the value for voltage currently stored in the measurement object.

```
[int error,
float value] StaExtv3mea.GetMeaValue(int unused,
                                     int applyMultiplicator)
```

ARGUMENTS

value (out)
Value for voltage, optionally multiplied by configured multiplicator

phase Index of desired phase. Index must be 0, 1 or 2.

applyMultiplicator
If 1 (default), returned value will be multiplied by the multiplicator stored in the measurement object. If 0, raw value will be returned.

RETURNS

0	on success
1	on error, e.g. phase index does not exist

GetStatus

Returns the status flags. Please note, this value is interpreted as a bitfield. See StaExtv3mea.SetStatus() for details on the status bits.

```
int StaExtv3mea.GetStatuts()
```

RETURNS

Status bitfield as an integer value.

GetStatusTmp

Returns the temporary (in memory) status flags. Please note, this value is interpreted as a bitfield. See StaExtv3mea.SetStatus() for details on the status bits.

```
int StaExtv3mea.GetStatusTmp()
```

RETURNS

Status bitfield as an integer value.

InitTmp

Initialises the temporary (in memory) fields of the measurement object with the values currently stored in the corresponding persistent fields. This affects temporary measurement value and temporary status fields. The temporary measurement value is used internally for comparison of new and old values for deadband violation. The temporary status is used during calculation in order to not modify initial value.

This function should be called once after the link has been established and before the calculation loop is executed.

```
None StaExtv3mea.InitTmp()
```

IsStatusBitSet

Checks if specific bit(s) are set in the status bitfield. See StaExtv3mea.SetStatus() for details on the status bits.

```
int StaExtv3mea.IsStatusBitSet(int mask)
```

RETURNS

0	if at least one bit in mask is not set
1	if all bit(s) in mask are set

IsStatusBitSetTmp

Checks if specific bit(s) are set in the temporary (in memory) status bitfield. See StaExtv3mea.SetStatus() for details on the status bits.

```
int StaExtv3mea.IsStatusBitSetTmp(int mask)
```

RETURNS

0	if at least one bit in mask is not set
1	if all bit(s) in mask are set

ResetStatusBit

Resets specific bits in the status bitfield. See StaExtv3mea.SetStatus() for details on the status bits.

```
None StaExtv3mea.ResetStatusBit(int mask, int dbSync)
```

ARGUMENTS

mask Mask of bits to set to 0. A bit is unchanged if already unset before.

dbSync (optional)

0	New status flags are applied in memory only
1	Default, new status flags are stored on db (persistent)

ResetStatusBitTmp

Resets specific bits in the temporary (in memory) status bitfield. See StaExtv3mea.SetStatus() for details on the status bits.

```
None StaExtv3mea.ResetStatusBitTmp(int mask)
```

ARGUMENTS

mask Mask of bits to set to 0. A bit is unchanged if already unset before.

SetMeaValue

Sets the value stored in the measurement object.

```
int StaExtv3mea.SetMeaValue(float value)
```

ARGUMENTS

value New value.

RETURNS

Return value has no meaning. It is always 0.

SetStatus

Sets the status flags of the measurement object. Please note, this value is interpreted as a bitfield where the bits have the following meaning. An option is considered enabled if the corresponding bit is set to 1.

bit0 0x00000001	Manually entered data
bit1 0x00000002	Tele-Measurement
bit2 0x00000004	Disturbance
bit3 0x00000008	Protection
bit4 0x00000010	Marked suspect
bit5 0x00000020	Violated constraint

bit6 0x00000040	On Event
bit7 0x00000080	Event Block.
bit8 0x00000100	Alarm Block.
bit9 0x00000200	Update Block.
bit10 0x00000400	Control Block.
bit29 0x20000000	Read
bit30 0x40000000	Write
bit31 0x80000000	Neglected by SE

```
None StaExtv3mea.SetStatus(int status, int dbSync)
```

ARGUMENTS

status Bitfield for status flags, see above

dbSync (optional)

0	New status flags are applied in memory only
1	Default, new status flags are stored on db (persistent)

SetStatusBit

Sets specific bits in the status bitfield. See StaExtv3mea.SetStatus() for details on the status bits.

```
None StaExtv3mea.SetStatusBit(int mask, int dbSync)
```

ARGUMENTS

mask Mask of bits to set to 1. A bit is unchanged if already set before.

dbSync (optional)

0	New status flags are applied in memory only
1	Default, new status flags are stored on db (persistent)

SetStatusBitTmp

Sets specific bits in the temporary (in memory) status bitfield. See StaExtv3mea.SetStatus() for details on the status bits.

```
None StaExtv3mea.SetStatusBitTmp(int mask)
```

ARGUMENTS

mask Mask of bits to set to 1. A bit is unchanged if already set before.

SetStatusTmp

Sets the temporary (in memory) status flags of the measurement object. This temporary value is used during calculations so that changes do not lead to object modifications and initial value remains unchanged.

Please note, this value is interpreted as a bitfield. See StaExtv3mea.SetStatus() for details on the status bits.

```
None StaExtv3mea.SetStatusTmp(int status)
```

ARGUMENTS

status	Bitfield for status flags, see above

UpdateControl

Transfers the value of current measurement object to the controller object (target object 'pCtrl' and target attribute 'varName'). If target object is a command, it is automatically executed afterwards.
Note: Calculation results will not be reset by this value transfer.

```
int StaExtv3mea.UpdateControl(int dbSync)
```

ARGUMENTS

dbSync (optional)

0	Value is copied in memory only
1	Default, copied value is stored on db (persistent)

RETURNS

0	on success
1	on error, e.g. target object does not have an attribute with given name

UpdateCtrl

Transfers the value of current measurement object to the controlled object (target object 'pObject' and target attribute 'variabName'). If target object is a command, it is automatically executed afterwards.
Note: Calculation results will not be reset by this value transfer.

```
int StaExtv3mea.UpdateCtrl(int dbSync)
```

ARGUMENTS

dbSync (optional)

0	Value is copied in memory only
1	Default, copied value is stored on db (persistent)

RETURNS

0	on success
1	on error, e.g. target object does not have an attribute with given name

4.3.15 StaExtvmea

Overview

CopyExtMeaStatusToStatusTmp

Copies the (persistent) status of current measurement object to temporary (in memory) status.

```
None StaExtvmea.CopyExtMeaStatusToStatusTmp()
```

GetMeaValue

Returns the value for voltage currently stored in the measurement object.

```
[int error,
float value] StaExtvmea.GetMeaValue(int unused,
                                    int applyMultiplicator)
```

ARGUMENTS

value (out)
Value for voltage, optionally multiplied by configured multiplicator

unused Not used.

applyMultiplicator
If 1 (default), returned value will be multiplied by the multiplicator stored in the measurement object. If 0, raw value will be returned.

RETURNS

Return value has no meaning. It is always 0.

GetStatus

Returns the status flags. Please note, this value is interpreted as a bitfield. See StaExtvmea.SetStatus() for details on the status bits.

```
int StaExtvmea.GetStatuts()
```

RETURNS

Status bitfield as an integer value.

GetStatusTmp

Returns the temporary (in memory) status flags. Please note, this value is interpreted as a bitfield. See StaExtvmea.SetStatus() for details on the status bits.

```
int StaExtvmea.GetStatusTmp()
```

RETURNS

Status bitfield as an integer value.

InitTmp

Initialises the temporary (in memory) fields of the measurement object with the values currently stored in the corresponding persistent fields. This affects temporary measurement value and temporary status fields. The temporary measurement value is used internally for comparison of new and old values for deadband violation. The temporary status is used during calculation in order to not modify initial value.

This function should be called once after the link has been established and before the calculation loop is executed.

```
None StaExtvmea.InitTmp()
```

IsStatusBitSet

Checks if specific bit(s) are set in the status bitfield. See StaExtvmea.SetStatus() for details on the status bits.

```
int StaExtvmea.IsStatusBitSet(int mask)
```

RETURNS

0	if at least one bit in mask is not set
1	if all bit(s) in mask are set

IsStatusBitSetTmp

Checks if specific bit(s) are set in the temporary (in memory) status bitfield. See StaExtvmea.SetStatus() for details on the status bits.

```
int StaExtvmea.IsStatusBitSetTmp(int mask)
```

RETURNS

0	if at least one bit in mask is not set
1	if all bit(s) in mask are set

ResetStatusBit

Resets specific bits in the status bitfield. See StaExtvmea.SetStatus() for details on the status bits.

```
None StaExtvmea.ResetStatusBit(int mask, int dbSync)
```

ARGUMENTS

mask Mask of bits to set to 0. A bit is unchanged if already unset before.

dbSync (optional)

0	New status flags are applied in memory only
1	Default, new status flags are stored on db (persistent)

ResetStatusBitTmp

Resets specific bits in the temporary (in memory) status bitfield. See StaExtvmea.SetStatus() for details on the status bits.

```
None StaExtvmea.ResetStatusBitTmp(int mask)
```

ARGUMENTS

mask Mask of bits to set to 0. A bit is unchanged if already unset before.

SetMeaValue

Sets the value stored in the measurement object.

```
int StaExtvmea.SetMeaValue(float value)
```

ARGUMENTS

value New value.

RETURNS

Return value has no meaning. It is always 0.

SetStatus

Sets the status flags of the measurement object. Please note, this value is interpreted as a bitfield where the bits have the following meaning. An option is considered enabled if the corresponding bit is set to 1.

bit0 0x00000001	Manually entered data
bit1 0x00000002	Tele-Measurement
bit2 0x00000004	Disturbance
bit3 0x00000008	Protection
bit4 0x00000010	Marked suspect
bit5 0x00000020	Violated constraint

bit6 0x00000040 On Event

bit7 0x00000080 Event Block.

bit8 0x00000100 Alarm Block.

bit9 0x00000200 Update Block.

bit10 0x00000400 Control Block.

bit29 0x20000000 Read

bit30 0x40000000 Write

bit31 0x80000000 Neglected by SE

```
None StaExtvmea.SetStatus(int status, int dbSync)
```

ARGUMENTS

status Bitfield for status flags, see above

dbSync (optional)

0	New status flags are applied in memory only
1	Default, new status flags are stored on db (persistent)

SetStatusBit

Sets specific bits in the status bitfield. See StaExtvmea.SetStatus() for details on the status bits.

```
None StaExtvmea.SetStatusBit(int mask, int dbSync)
```

ARGUMENTS

mask Mask of bits to set to 1. A bit is unchanged if already set before.

dbSync (optional)

0	New status flags are applied in memory only
1	Default, new status flags are stored on db (persistent)

SetStatusBitTmp

Sets specific bits in the temporary (in memory) status bitfield. See StaExtvmea.SetStatus() for details on the status bits.

```
None StaExtvmea.SetStatusBitTmp(int mask)
```

ARGUMENTS

mask Mask of bits to set to 1. A bit is unchanged if already set before.

SetStatusTmp

Sets the temporary (in memory) status flags of the measurement object. This temporary value is used during calculations so that changes do not lead to object modifications and initial value remains unchanged.

Please note, this value is interpreted as a bitfield. See StaExtvmea.SetStatus() for details on the status bits.

```
None StaExtvmea.SetStatusTmp(int status)
```

ARGUMENTS

status	Bitfield for status flags, see above

UpdateControl

Transfers the value of current measurement object to the controller object (target object 'pCtrl' and target attribute 'varName'). If target object is a command, it is automatically executed afterwards.
Note: Calculation results will not be reset by this value transfer.

```
int StaExtvmea.UpdateControl(int dbSync)
```

ARGUMENTS

dbSync (optional)

0	Value is copied in memory only
1	Default, copied value is stored on db (persistent)

RETURNS

0	on success
1	on error, e.g. target object does not have an attribute with given name

UpdateCtrl

Transfers the value of current measurement object to the controlled object (target object 'pObject' and target attribute 'variabName'). If target object is a command, it is automatically executed afterwards.
Note: Calculation results will not be reset by this value transfer.

```
int StaExtvmea.UpdateCtrl(int dbSync)
```

ARGUMENTS

dbSync (optional)

0	Value is copied in memory only
1	Default, copied value is stored on db (persistent)

RETURNS

0	on success
1	on error, e.g. target object does not have an attribute with given name

4.3.16 StaSwitch

Overview

Close
IsClosed
IsOpen
Open

Close

Closes the switch by changing its status to 'close'. This action will fail if the status is currently determined by an active running arrangement.

```
int StaSwitch.Close()
```

RETURNS

0	On success
$\neq$ **0**	On error

SEE ALSO

StaSwitch.Open()

IsClosed

Returns information about current switch state.

```
int StaSwitch.IsClosed()
```

RETURNS

1	switch is closed
0	switch is open

SEE ALSO

StaSwitch.IsOpen()

IsOpen

Returns information about current switch state.

```
int StaSwitch.IsOpen()
```

RETURNS

1	switch is open
0	switch is closed

SEE ALSO

StaSwitch.IsClosed()

Open

Opens the switch by changing its status to 'open'. This action will fail if the status is currently determined by an active running arrangement.

```
int StaSwitch.Open()
```

RETURNS

0	On success
$\neq$ **0**	On error

SEE ALSO

StaSwitch.Close()

4.4 Commands

Overview

Execute

Execute

Executes the command.

```
int Com*.Execute()
```

4.4.1 ComAddlabel

Overview

Execute

Execute

This function executes the Add Statistic Labels command itself for a given plot and curve.

```
int ElmRes.ComAddlabel(DataObject, int curveIndex)
```

ARGUMENTS

plot	The plot to modify.
curveIndex	The index of the curve inside the plot's table. The index is zero based, therefore the index of the first curve is 0.

RETURNS

0	The function executed without any errors.
1	The plot is visible on a single line graphic only.
2	The parameter plot is None.
3	The parameter plot is not a virtual instrument (classname should start with Vis).
4	The object plot was found in any open graphic.

5	The object plot is not a diagram.
6	An internal error occured (plot is incomplete).

4.4.2 ComAddon

Overview

CreateModule

Creates the calculation module of this AddOn. Volatile object parameters are created for all variable definitions stored inside this command. They are accessible like any other built in object parameter.

```
int ComAddon.CreateModule()
```

RETURNS

0	Ok, module was created.
1	An error occurred.

SEE ALSO

ComAddon.FinaliseModule() ComAddon.DeleteModule()

DefineDouble

Creates a new floating-point-number parameter for the given type of objects.

```
int ComAddon.DefineDouble(str class,
                          str name,
                          str desc,
                          str unit,
                          float initial)
```

ARGUMENTS

class	The type of objects for which the new parameter is to be created, e.g. ElmLne for the line.
name	Name of the new parameter
desc	Parameter description
unit	Parameter's unit
initial	Default value of new parameter

RETURNS

0 Ok, Parameter was created.

Other than 0 An error occurred, possible reasons:

- The module of this add on does not exist.
- An object with the given class does not exist in PowerFactory.
- The parameter name for the given class already exists in the module.

SEE ALSO

ComAddon.DefineDoublePerConnection()

DefineDoubleMatrix

Creates a new floating-point-matrix parameter for the given type of objects.

```
int ComAddon.DefineDoubleMatrix(string class,
                                str name,
                                str desc,
                                str unit,
                                int rows,
                                int columns,
                                float initial)
```

ARGUMENTS

class	The type of objects for which the new parameter is to be created, e.g. ElmLne for the line.
name	Name of the new parameter
desc	Parameter description
unit	Parameter's unit
rows	Number of initial rows. Number of rows will be 0 if a value smaller than 0 is given.
columns	Number of initial columns. Number of columns will be 0 if a value smaller than 0 is given.
initial	Default value for all entries of new parameter

RETURNS

0 Ok, Parameter was created.

Other than 0 An error occurred, possible reasons:

- The module of this add on does not exist.
- An object with the given class does not exist in PowerFactory.
- The parameter name for the given class already exists in the module.

DefineDoublePerConnection

Creates a new floating-point-number parameter for every connection for the given type of objects.

```
int ComAddon.DefineDoublePerConnection(str class,
                                       str name,
                                       str desc,
                                       str unit,
                                       float initial)
```

ARGUMENTS

class	The type of objects for which the new parameter is to be created, e.g. ElmLne for the line.
name	Name of the new parameter
desc	Parameter description
unit	Parameter's unit
initial	Default value of new parameter

RETURNS

0 Ok, Parameter was created.

Other than 0 An error occurred, possible reasons:

- The module of this add on does not exist.
- An object with the given class does not exist in PowerFactory.
- The parameter name for the given class already exists in the module.
- The elements of class are not branch elements. Therefore there is no connection count. DefineDouble shall be used instead.

SEE ALSO

ComAddon.DefineDouble()

DefineDoubleVector

Creates a new floating-point-number vector parameter for the given type of objects.

```
int ComAddon.DefineDoubleVector(str class,
                                str name,
                                str desc,
                                str unit,
                                float initial,
                                int size)
```

ARGUMENTS

class	The type of objects for which the new parameter is to be created, e.g. ElmLne for the line.
name	Name of the new parameter
desc	Parameter description
unit	Parameter's unit
initial	Default value of new parameter
size	Initial size of vector. Size will be 0 if a value smaller than 0 is given.

RETURNS

0 Ok, Parameter was created.

Other than 0 An error occurred, possible reasons:
- The module of this add on does not exist.
- An object with the given class does not exist in PowerFactory.
- The parameter name for the given class already exists in the module.

SEE ALSO

ComAddon.DefineDouble() ComAddon.DefineDoublePerConnection()

DefineDoubleVectorPerConnection

Creates a new floating-point-number vector parameter for the given type of objects for every connection of the object.

```
int ComAddon.DefineDoubleVectorPerConnection(str class,
                                             str name,
                                             str desc,
                                             str unit,
                                             float initial,
                                             int size)
```

ARGUMENTS

class	The type of objects for which the new parameter is to be created, e.g. ElmLne for the line.
name	Name of the new parameter
desc	Parameter description
unit	Parameter's unit
initial	Default value of new parameter
size	Initial size of vector. Size will be 0 if a value smaller than 0 is given.

RETURNS

0 Ok, Parameter was created.

Other than 0 An error occurred, possible reasons:
- The module of this add on does not exist.
- An object with the given class does not exist in PowerFactory.
- The parameter name for the given class already exists in the module.
- The elements of class are not branch elements. Therefore there is no connection count. DefineDoubleVector shall be used instead.

SEE ALSO

ComAddon.DefineDoubleVector()

DefineInteger

Creates a new integer parameter for the given type of objects.

```
int ComAddon.DefineInteger(str class,
                           str name,
                           str desc,
                           str unit,
                           int initial)
```

ARGUMENTS

class	The type of objects for which the new parameter is to be created, e.g. ElmLne for the line.
name	Name of the new parameter
desc	Parameter description
unit	Parameter's unit
initial	Default value of new parameter

RETURNS

0 Ok, Parameter was created.

Other than 0 An error occurred, possible reasons:

- The module of this add on does not exist.
- An object with the given class does not exist in PowerFactory.
- The parameter name for the given class already exists in the module.

SEE ALSO

ComAddon.DefineIntegerPerConnection()

DefineIntegerPerConnection

Creates a new integer parameter for every connection for the given type of objects.

```
int ComAddon.DefineIntegerPerConnection(str class,
                                        str name,
                                        str desc,
                                        str unit,
                                        int initial)
```

ARGUMENTS

class	The type of objects for which the new parameter is to be created, e.g. ElmLne for the line.
name	Name of the new parameter
desc	Parameter description
unit	Parameter's unit
initial	Default value of new parameter

RETURNS

0 Ok, Parameter was created.

Other than 0 An error occurred, possible reasons:

- The module of this add on does not exist.
- An object with the given class does not exist in PowerFactory.
- The parameter name for the given class already exists in the module.
- The elements of class are not branch elements. Therefore there is no connection count. DefineInteger shall be used instead.

SEE ALSO

ComAddon.DefineInteger()

DefineIntegerVector

Creates a new integer vector parameter for the given type of objects.

```
int ComAddon.DefineIntegerVector(str class,
                                 str name,
                                 str desc,
                                 str unit,
                                 int initial,
                                 int size)
```

ARGUMENTS

class	The type of objects for which the new parameter is to be created, e.g. ElmLne for the line.
name	Name of the new parameter
desc	Parameter description
unit	Parameter's unit
initial	Default value of new parameter
size	Initial size of vector. Size will be 0 if a value smaller than 0 is given.

RETURNS

0 Ok, Parameter was created.

Other than 0 An error occurred, possible reasons:

- The module of this add on does not exist.
- An object with the given class does not exist in PowerFactory.
- The parameter name for the given class already exists in the module.

SEE ALSO

ComAddon.DefineInteger() ComAddon.DefineIntegerPerConnection()

DefineIntegerVectorPerConnection

Creates a new integer vector parameter for the given type of objects for every connection of the object.

```
int ComAddon.DefineIntegerVectorPerConnection(str class,
                                              str name,
                                              str desc,
                                              str unit,
                                              int initial,
                                              int size)
```

ARGUMENTS

class	The type of objects for which the new parameter is to be created, e.g. ElmLne for the line.
name	Name of the new parameter

desc	Parameter description
unit	Parameter's unit
initial	Default value of new parameter
size	Initial size of vector. Size will be 0 if a value smaller than 0 is given.

RETURNS

0 Ok, Parameter was created.

Other than 0 An error occurred, possible reasons:

- The module of this add on does not exist.
- An object with the given class does not exist in PowerFactory.
- The parameter name for the given class already exists in the module.
- The elements of class are not branch elements. Therefore there is no connection count. DefineIntegerVector shall be used instead.

SEE ALSO

ComAddon.DefineIntegerVector()

DefineObject

Creates a new object parameter for the given type of objects.

```
int ComAddon.DefineObject(str class,
                          str name,
                          str desc,
                          DataObject initial)
```

ARGUMENTS

class	The type of objects for which the new parameter is to be created, e.g. ElmLne for the line.
name	Name of the new parameter
desc	Parameter description
initial	Default object of new parameter

RETURNS

0 Ok, Parameter was created.

Other than 0 An error occurred, possible reasons:

- The module of this add on does not exist.
- An object with the given class does not exist in PowerFactory.
- The parameter name for the given class already exists in the module.

SEE ALSO

ComAddon.DefineObjectPerConnection()

DefineObjectPerConnection

Creates a new object parameter for every connection for the given type of objects.

```
int ComAddon.DefineObjectPerConnection(str class,
                                       str name,
                                       str desc,
                                       DataObject initial)
```

ARGUMENTS

class	The type of objects for which the new parameter is to be created, e.g. ElmLne for the line.
name	Name of the new parameter
desc	Parameter description
initial	Default value of new parameter

RETURNS

0 Ok, Parameter was created.

Other than 0 An error occurred, possible reasons:

- The module of this add on does not exist.
- An object with the given class does not exist in PowerFactory.
- The parameter name for the given class already exists in the module.
- The elements of class are not branch elements. Therefore there is no connection count. DefineObject shall be used instead.

SEE ALSO

ComAddon.DefineObject()

DefineObjectVector

Creates a new object vector parameter for the given type of objects.

```
int ComAddon.DefineObjectVector(str class,
                                str name,
                                str desc,
                                str unit,
                                DataObject initial,
                                int size)
```

ARGUMENTS

class	The type of objects for which the new parameter is to be created, e.g. ElmLne for the line.
name	Name of the new parameter
desc	Parameter description
unit	Parameter's unit
initial	Default value of new parameter
size	Initial size of vector. Size will be 0 if a value smaller than 0 is given.

RETURNS

0 Ok, Parameter was created.

Other than 0 An error occurred, possible reasons:

- The module of this add on does not exist.
- An object with the given class does not exist in PowerFactory.
- The parameter name for the given class already exists in the module.

SEE ALSO

ComAddon.DefineObject() ComAddon.DefineObjectPerConnection()

DefineObjectVectorPerConnection

Creates a new object vector parameter for the given type of objects for every connection of the object.

```
int ComAddon.DefineObjectVectorPerConnection(str class,
                                             str name,
                                             str desc,
                                             str unit,
                                             DataObject initial,
                                             int size)
```

ARGUMENTS

class The type of objects for which the new parameter is to be created, e.g. ElmLne for the line.

name Name of the new parameter

desc Parameter description

unit Parameter's unit

initial Default value of new parameter

size Initial size of vector. Size will be 0 if a value smaller than 0 is given.

RETURNS

0 Ok, Parameter was created.

Other than 0 An error occurred, possible reasons:

- The module of this add on does not exist.
- An object with the given class does not exist in PowerFactory.
- The parameter name for the given class already exists in the module.
- The elements of class are not branch elements. Therefore there is no connection count. DefineObjectVector shall be used instead.

SEE ALSO

ComAddon.DefineObjectVector()

DefineString

Creates a new text parameter for the given type of objects.

```
int ComAddon.DefineString(str class,
                          str name,
                          str desc,
                          str unit,
                          str initial)
```

ARGUMENTS

class	The type of objects for which the new parameter is to be created, e.g. ElmLne for the line.
name	Name of the new parameter
desc	Parameter description
unit	Parameter's unit
initial	Default value of new parameter

RETURNS

0 Ok, Parameter was created.

Other than 0 An error occurred, possible reasons:

- The module of this add on does not exist.
- An object with the given class does not exist in PowerFactory.
- The parameter name for the given class already exists in the module.

SEE ALSO

ComAddon.DefineStringPerConnection()

DefineStringPerConnection

Creates a new text parameter for every connection for the given type of objects.

```
int ComAddon.DefineStringPerConnection(str class,
                                       str name,
                                       str desc,
                                       str unit,
                                       str initial)
```

ARGUMENTS

class	The type of objects for which the new parameter is to be created, e.g. ElmLne for the line.
name	Name of the new parameter
desc	Parameter description
unit	Parameter's unit
initial	Default value of new parameter

RETURNS

0 Ok, Parameter was created.

Other than 0 An error occurred, possible reasons:

- The module of this add on does not exist.
- An object with the given class does not exist in PowerFactory.
- The parameter name for the given class already exists in the module.
- The elements of class are not branch elements. Therefore there is no connection count. DefineString shall be used instead.

SEE ALSO

ComAddon.DefineString()

DeleteModule

Deletes the module of this add on.

```
int ComAddon.DeleteModule()
```

0	Success. The module is deleted completely.
1	Failure. The module does not exist and can therefore not be deleted.

SEE ALSO

ComAddon.CreateModule()

FinaliseModule

Finalises a user defined module which was created using the mthod CreateModule. All user defined variables defined for this module are read-only after the call of finalise module. The module is the one being used in the flexible data, single line graphic text boxes and colouring. It can be reset like any other built-in calculation using the reset button.

```
int ComAddon.FinaliseModule()
```

RETURNS

0	Ok, module was finalised.
1	An error occurred, this command is not the one being currently active.

SEE ALSO

ComAddon.CreateModule()

GetActiveModule

Gets the key of the module being currently active. An empty string is returned if there is no active module.

```
str ComAddon.GetActiveModule()
```

RETURNS

The key of the active module. an empty string is returned if there is no active module.

SEE ALSO

ComAddon.SetActiveModule()

ModuleExists

Checks if the module for this add-on was already created using the method CreateModule.

```
int ComAddon.ModuleExists()
```

RETURNS

0	The module was not created yet.
1	The module was already created.

SEE ALSO

ComAddon.CreateModule() ComAddon.FinaliseModule() ComAddon.DeleteModule()

SetActiveModule

Set this module as active module. This method is required only if several modules are created concurrently. In case that only one module is being used, there is no need to use this method, because CreateModule sets the created module automatically as active module.

```
int ComAddon.SetActiveModule()
```

RETURNS

0	Success. This command is set as active module.
1	Failure. This command is already the active module.

SEE ALSO

ComAddon.CreateModule() ComAddon.FinaliseModule() ComAddon.DeleteModule()

4.4.3 ComCapo

Overview

ConnectShuntToBus
LossCostAtBusTech
TotalLossCost

ConnectShuntToBus

Connects the equivalent shunt in the specified terminal and executes the load flow command. The shunt is not physically added in the database, just the susceptance is added for the calculation.

```
int ComCapo.ConnectShuntToBus(DataObject terminal,
                              float phtech,
                              float Q
                              )
```

ARGUMENTS

terminal The terminal to which the shunt will be connected

phtech Phase technology. Possible values are

0	three-phase
1	ph-ph a-b
2	ph-ph b-c
3	ph-ph a-c
4	ph-e a
5	ph-e b

6	ph-e c

Note: In balanced load flow, the technology will always be three-phase.

Q Reactive power value in Mvar

RETURNS

0	On succes.
1	An error occurred during load flow execution.

LossCostAtBusTech

Returns the losses cost of the selected terminal and configuration calculated during the sensitivity analysis or the optimization.

```
float ComCapo.LossCostAtBusTech(DataObject terminal,
                                float phtech
                                )
```

ARGUMENTS

terminal Specified bus

phtech Phase technology. Possible values are

0	three-phase
1	ph-ph a-b
2	ph-ph b-c
3	ph-ph a-c
4	ph-e a
5	ph-e b
6	ph-e c

RETURNS

Returns the losses cost

TotalLossCost

Returns the total cost calculated after the sensitivity analysis or the optimization.

```
float ComCapo.TotalLossCost([int iopt])
```

ARGUMENTS

iopt (optional)
Type of cost. Possible values are

0	Losses in MW (default)
1	Cost of losses
2	Cost of voltage violations
3	Cost of shunts

RETURNS

Returns losses in MW or cost value.

4.4.4 ComConreq

Overview

Execute

Execute

Performs a Connection Request Assessment according to the selected method. Results are provided for connection request elements in the single line graphic, and are summarised in a report in the output window.

```
int ComConreq.Execute()
```

RETURNS

0	OK
1	Error: calculation function
2	Error: settings/initialisation/load flow

4.4.5 ComContingency

Overview

ContinueTrace
CreateRecoveryInformation
GetGeneratorEvent
GetInterruptedPowerAndCustomersForStage
GetInterruptedPowerAndCustomersForTimeStep
GetLoadEvent
GetNumberOfGeneratorEventsForTimeStep
GetNumberOfLoadEventsForTimeStep
GetNumberOfSwitchEventsForTimeStep
GetNumberOfTimeSteps
GetObj
GetSwitchEvent
GetTimeOfStepInSeconds
GetTotalInterruptedPower
JumpToLastStep
RemoveEvents
StartTrace
StopTrace

ContinueTrace

Continues trace execution for this contingency.

```
int ComContingency.ContinueTrace()
```

RETURNS

0	On success.
1	On error.

CreateRecoveryInformation

Creates recovery information for a contingency. The recovery information can later be retrieved e.g. via ComContingency.GetInterruptedPowerAndCustomersForStage().
Can only save one contingency at the same time.

```
int ComContingency.CreateRecoveryInformation(DataObject resultFileInput)
```

ARGUMENTS

resultFileInput
Read from this result file.

RETURNS

0	On success.
1	On error.

GetGeneratorEvent

Gets generator event of a certain time step during recovery.
ComContingency.CreateRecoveryInformation() has to be called beforehand to collect the data.

```
[ DataObject generator,
double changedP,
double changedQ ) ]
ComContingency.GetGeneratorEvent(int currentTimeStep,
                                 int loadEvent)
```

ARGUMENTS

currentTimeStep
Input: Number of time steps are get via ComContingency.GetNumberOfTimeSteps()

switchEvent
Input: Number of generator events for a certain time step are get via ComContingency.GetNumberOfSwitchEventsForTimeStep()

generator (out)
Output: Generator that dispatched

changedP (out)
Output: Changed active power

changedQ (out)
Output: Changed reactive power

GetInterruptedPowerAndCustomersForStage

Gets recovery information of a contingency.
ComContingency.CreateRecoveryInformation() has to be called beforehand to collect the data.

```
[ int error,
float interruptedPower,
float newInterruptedPower,
float interruptedCustomers,
float newInterruptedCustomers ]
ComContingency.GetInterruptedPowerAndCustomersForStage(float timeOfStageInMinutes)
```

ARGUMENTS

timeOfStageInMinutes
Input: Get Information for this time.

interruptedPower (out)
Output: Interrupted Power at this time.

newInterruptedPower (out)
Output: New interrupted Power at this time.

interruptedCustomers (out)
Output: Interrupted Customers at this time.

newInterruptedCustomers (out)
Output: New interrupted Customers at this time.

RETURNS

0	On success.
1	On error.

GetInterruptedPowerAndCustomersForTimeStep

Gets recovery information of a contingency.
ComContingency.CreateRecoveryInformation() has to be called beforehand to collect the data.

```
[ float interruptedPower,
float newInterruptedPower,
float interruptedCustomers,
float newInterruptedCustomers ]
ComContingency.GetInterruptedPowerAndCustomersForTimeStep(int currentTimeStep)
```

ARGUMENTS

currentTimeStep
Input: Number of time steps are get via ComContingency.GetNumberOfTimeSteps()

interruptedPower (out)
Output: Interrupted Power at this time.

newInterruptedPower (out)
Output: New interrupted Power at this time.

interruptedCustomers (out)
Output: Interrupted Customers at this time.

newInterruptedCustomers (out)
Output: New interrupted Customers at this time.

GetLoadEvent

Gets load event of a certain time step during recovery.
ComContingency.CreateRecoveryInformation() has to be called beforehand to collect the data.

```
[ DataObject load,
double changedP,
double changedQ,
int& isTransfer ) ]
ComContingency.GetLoadEvent(int currentTimeStep,
                            int loadEvent)
```

ARGUMENTS

currentTimeStep
Input: Number of time steps are get via ComContingency.GetNumberOfTimeSteps()

switchEvent
Input: Number of load events for a certain time step are get via ComContingency.GetNumberOfSwitchEventsForTimeStep()

load (out) Output: Load that is shed or transfered

changedP (out)
Output: Changed active power

changedQ (out)
Output: Changed reactive power

isTransfer (out)
Output: = 0 : is load shedding event. >0 is load transfer event.

GetNumberOfGeneratorEventsForTimeStep

Returns the number of generator events of a certain step during recovery.
ComContingency.CreateRecoveryInformation() has to be called beforehand to collect the data.

```
list(int, ...) ComContingency.GetNumberOfGeneratorEventsForTimeStep([int currentTimeStep])
```

ARGUMENTS

currentTimeStep
Input: Number of time steps are get via ComContingency.GetNumberOfTimeSteps()

GetNumberOfLoadEventsForTimeStep

Returns the number of load events of a certain step during recovery.
ComContingency.CreateRecoveryInformation() has to be called beforehand to collect the data.

```
list(int, ...) ComContingency.GetNumberOfLoadEventsForTimeStep([int currentTimeStep])
```

ARGUMENTS

currentTimeStep
Input: Number of time steps are get via ComContingency.GetNumberOfTimeSteps()

GetNumberOfSwitchEventsForTimeStep

Returns the number of switch events of a certain step during recovery.
ComContingency.CreateRecoveryInformation() has to be called beforehand to collect the data.

```
list(int, ...) ComContingency.GetNumberOfSwitchEventsForTimeStep([int currentTimeStep])
```

ARGUMENTS

currentTimeStep
Input: Number of time steps are get via ComContingency.GetNumberOfTimeSteps()

GetNumberOfTimeSteps

Returns the number of time steps during recovery.
ComContingency.CreateRecoveryInformation() has to be called beforehand to collect the data.

```
int ComContingency.GetNumberOfTimeSteps()
```

GetObj

Gets interrupted element by index (zero based).

```
DataObject ComContingency.GetObj(int index)
```

ARGUMENTS

index Element order index, 0 for the first object.

RETURNS

object Interupted element for given index.
None Index out of range.

GetSwitchEvent

Gets switch event of a certain time step during recovery.
ComContingency.CreateRecoveryInformation() has to be called beforehand to collect the data.

```
[ DataObject switchToBeActuated,
int isClosed,
int sectionalizingStep ) ]
ComContingency.GetSwitchEvent(int currentTimeStep,
                              int switchEvent)
```

ARGUMENTS

currentTimeStep
Input: Number of time steps are get via ComContingency.GetNumberOfTimeSteps()

switchEvent
Input: Number of switch event for a certain time step are get via ComContingency.GetNumberOfSwitchEventsForTimeStep()

switchToBeActuated (out)
Output: Switch to be actuated

isClosed (out)
Output: $>$ 0 if switch is closed

sectionalizingStep (out)
Output: sectionalizing step when this switch is actuated

GetTimeOfStepInSeconds

Returns the time of the current step during recovery.
ComContingency.CreateRecoveryInformation() has to be called beforehand to collect the data.

```
list(float , ...) ComContingency.GetTimeOfStepInSeconds(int currentTimeStep)
```

ARGUMENTS

currentTimeStep
Input: Number of time steps are get via ComContingency.GetNumberOfTimeSteps()

GetTotalInterruptedPower

Gets the total interrupted power (in kW) during restoration. ComContingency.CreateRecoveryInformation() has to be called beforehand to collect the data.

```
float  ComContingency.GetTotalInterruptedPower()
```

JumpToLastStep

Gets the last trace execution for this contingency.

```
int ComContingency.JumpToLastStep([float timeDelay])
```

ARGUMENTS

timeDelay (optional)
time delay in seconds between trace steps

RETURNS

0	On success.
1	On error.

RemoveEvents

Removes events from this contingency.

```
None ComContingency.RemoveEvents([float emitMessage])
None ComContingency.RemoveEvents(str whichEvents)
None ComContingency.RemoveEvents(float emitMessage,
                                 str whichEvents
                                 )
None ComContingency.RemoveEvents(str whichEvents,
                                 float emitMessage
                                 )
```

ARGUMENTS

emitMessage(optional)
0: no info message shall be issued after event removal

whichEvents(optional)
'lod' removed load evenets, 'gen' removes generator events, 'switch' removes switching events

StartTrace

Starts trace execution for this contingency.

```
int ComContingency.StartTrace()
```

RETURNS

0	On success.
1	Error, e.g. Contingency is not in trace.
2	On error.

StopTrace

Stops trace execution for this contingency.

```
int ComContingency.StopTrace([int emitMessage])
```

ARGUMENTS

emitMessage (optional)
= 0: no trace-stop info messages shall be issued

RETURNS

0	On Success.
1	Contingency is not in Trace.

4.4.6 ComDiff

Overview

Start
Stop

Start

Starts comparisons of calculation results. See Application.SetDiffMode() for more information.

SEE ALSO

ComDiff.Stop(), Application.GetDiffMode(), Application.SetDiffMode()

Stop

Stops comparisons of calculation results. See Application.SetDiffMode() for more information.

SEE ALSO

ComDiff.Start(), Application.GetDiffMode(), Application.SetDiffMode()

4.4.7 ComDllmanager

Overview

Report

Report

Prints a status report of currently available external user-defined dlls (e.g. dpl, exdyn) to the output window. (Same as pressing the 'Report' button in the dialog.)

```
None ComDllmanager.Report()
```

4.4.8 ComDpl

Overview

CheckSyntax
Encrypt
Execute
GetExternalObject
GetInputParameterDouble
GetInputParameterInt
GetInputParameterString
IsEncrypted
SetExternalObject
SetInputParameterDouble
SetInputParameterInt
SetInputParameterString

CheckSyntax

Checks the syntax and input parameter of the DPL script and all its subscripts.

```
int ComDpl.CheckSyntax()
```

RETURNS

0	On success.
1	On error.

SEE ALSO

ComDpl.Execute()

Encrypt

Encrypts a script and all its subscripts. The password is needed only to decrypt the encrypted script. Execution of encrypted scripts work without password. If no password is given a 'Choose Password' dialog appears.

```
int ComDpl.Encrypt([str password], [int doRemoveHistoricRecords])
```

ARGUMENTS

password (optional)
password for decryption

doRemoveHistoricRecords (optional)
0: do not remove historic copies in database. 1: do remove 2: show dialog and ask.

RETURNS

0	On success.
1	On error.

SEE ALSO

ComDpl.IsEncrypted()

Execute

Executes the DPL script. It is not possible to call this function with input parameter. Use ComDpl.SetInputParameterInt(), ComDpl.SetInputParameterDouble() and ComDpl.SetInputParameterString() instead.

```
int ComDpl.Execute()
```

RETURNS

For scripts without the use of exit() the following values are returned:

0	On a successfull execution.
1	An error occured.
6	User hit the break button.

SEE ALSO

ComDpl.CheckSyntax()

GetExternalObject

Gets the external object defined in the ComDpl edit dialog.

```
[int error,
DataObject value] ComDpl.GetExternalObject(str name)
```

ARGUMENTS

name Name of the external object parameter.

value (out)
The external object.

RETURNS

0	On success.
1	On error.

SEE ALSO

ComDpl.SetExternalObject(), ComDpl.GetInputParameterInt(), ComDpl.GetInputParameterDouble(), ComDpl.GetInputParameterString()

GetInputParameterDouble

Gets the double input parameter value defined in the ComDpl edit dialog.

```
[int error,
float value] ComDpl.GetInputParameterDouble(str name)
```

ARGUMENTS

name Name of the double input parameter.

value (out)
Value of the double input parameter.

RETURNS

0	On success.
1	On error.

SEE ALSO

ComDpl.SetInputParameterDouble(), ComDpl.GetInputParameterInt(), ComDpl.GetInputParameterString(), ComDpl.GetExternalObject()

GetInputParameterInt

Gets the integer input parameter value defined in the ComDpl edit dialog.

```
[int error,
int value ] ComDpl.GetInputParameterInt(str name)
```

ARGUMENTS

name Name of the integer input parameter.

value (out)
Value of the integer input parameter.

RETURNS

0	On success.
1	On error.

SEE ALSO

ComDpl.SetInputParameterInt(), ComDpl.GetInputParameterDouble(), ComDpl.GetInputParameterString(), ComDpl.GetExternalObject()

GetInputParameterString

Gets the string input parameter value defined in the ComDpl edit dialog.

```
[int error,
str value ] ComDpl.GetInputParameterString(str name)
```

ARGUMENTS

name Name of the string input parameter.

value (out)
Value of the string input parameter.

RETURNS

0 On success.

1 On error.

SEE ALSO

ComDpl.SetInputParameterString(), ComDpl.GetInputParameterInt(), ComDpl.GetInputParameterDouble(), ComDpl.GetExternalObject()

IsEncrypted

Returns the encryption state of the script.

```
int ComDpl.IsEncrypted()
```

RETURNS

1 Script is encrypted.

0 Script is not encrypted.

SEE ALSO

ComDpl.Encrypt()

SetExternalObject

Sets the external object defined in the ComDpl edit dialog.

```
int ComDpl.SetExternalObject(str name,
                             DataObject value
                             )
```

ARGUMENTS

name Name of the external object parameter.

value The external object.

RETURNS

0 On success.

1 On error.

SEE ALSO

ComDpl.GetExternalObject(), ComDpl.SetInputParameterInt(), ComDpl.SetInputParameterDouble(), ComDpl.SetInputParameterString()

SetInputParameterDouble

Sets the double input parameter value defined in the ComDpl edit dialog.

```
int ComDpl.SetInputParameterDouble(str name,
                                   double value
                                   )
```

ARGUMENTS

name	Name of the double input parameter.
value	Value of the double input parameter.

RETURNS

0	On success.
1	On error.

SEE ALSO

ComDpl.GetInputParameterDouble(), ComDpl.SetInputParameterInt(), ComDpl.SetInputParameterString(), ComDpl.SetExternalObject()

SetInputParameterInt

Sets the integer input parameter value defined in the ComDpl edit dialog.

```
int ComDpl.SetInputParameterInt(str name,
                                int value
                                )
```

ARGUMENTS

name	Name of the integer input parameter.
value	Value of the integer input parameter.

RETURNS

0	On success.
1	On error.

SEE ALSO

ComDpl.GetInputParameterInt(), ComDpl.SetInputParameterDouble(), ComDpl.SetInputParameterString(), ComDpl.SetExternalObject()

SetInputParameterString

Sets the string input parameter value defined in the ComDpl edit dialog.

```
int ComDpl.SetInputParameterString(str name,
                                   str value
                                   )
```

ARGUMENTS

name	Name of the string input parameter.
value	Value of the string input parameter.

RETURNS

0	On success.
1	On error.

SEE ALSO

ComDpl.GetInputParameterString(), ComDpl.SetInputParameterInt(), ComDpl.SetInputParameterDouble(), ComDpl.SetExternalObject()

4.4.9 ComFlickermeter

Overview

Execute

Execute

Calculates the short- and long-term flicker according to IEC 61000-4-15.

```
int ComFlickermeter.Execute()
```

RETURNS

0	OK
1	Error: column not found in file; other internal errors
2	Error: empty input file
3	Error: cannot open file
4	Internal error: matrix empty

4.4.10 ComGenrelinc

Overview

GetCurrentIteration
GetMaxNumIterations

GetCurrentIteration

The command returns the current iteration number of the 'Run Generation Adequacy' command (ComGenrel).

```
int ComGenrelinc.GetCurrentIteration()
```

RETURNS

Returns the current iteration number.

GetMaxNumIterations

The command returns the maximume number of iterations specified in the 'Run Generation Adequacy' command (ComGenrel).

```
int ComGenrelinc.GetMaxNumIterations()
```

RETURNS

Returns the maximum number of iterations.

4.4.11 ComGridtocim

Overview

ConvertAndExport
SetAuthorityUri
SetBoundaries
SetGridsToExport

ConvertAndExport

Convert Grid to CIM into a temporary archive and save it as zip-file.

```
int ComGridtocim.ConvertAndExport(str fileName,
                                  int withValidation
                                  )
```

ARGUMENTS

fileName File name for zip-archive.

withValidation **0** Do not validate CIM-archive.
1 Validate CIM-archive.

SetAuthorityUri

Sets the authority uri for a specific grid.

```
None ComGridtocim.SetAuthorityUri(DataObject grid,
                                  str uri
                                  )
```

ARGUMENTS

grid Grid to set to set the URI for.

uri Model authority URI to be set.

SetBoundaries

Sets the grids as "Boundary Grid" and clears any previous setting.

```
None ComGridtocim.SetBoundaries(list grids)
```

ARGUMENTS

grids	The grids to be considered as boundaries.

SetGridsToExport

Sets the grids as ”Selected” and clears any previous setting.

```
None ComGridtocim.SetGridsToExport(list grids)
```

ARGUMENTS

grids	The grids to be selectted.

4.4.12 ComImport

Overview

GetCreatedObjects
GetModifiedObjects

GetCreatedObjects

Returns the newly created objects after execution of a DGS import.

```
list ComImport.GetCreatedObjects()
```

RETURNS

Collection of objects that have been created during DGS import.

GetModifiedObjects

Returns the modified objects after execution of a DGS import.

```
list ComImport.GetModifiedObjects()
```

RETURNS

Collection of objects that have been modified during DGS import.

4.4.13 ComLdf

Overview

CalcLdf
CalcParams
CheckControllers
DoNotResetCalc
EstimateLoading
EstimateOutage
Execute
IsAC

CalcLdf

Perform a load flow analysis with new topology rebuild, but without initialisation of calculation parameters.

```
int ComLdf.CalcLdf()
```

RETURNS

0	On success.
1	On error.

CalcParams

Initialise calculation parameters for all models.

```
int ComLdf.CalcParams()
```

RETURNS

Always return 1.

CheckControllers

Check the conditions of all controllers based on available load flow results. The report will be printed out in output window.

```
int ComLdf.CheckControllers()
```

RETURNS

Always return 1.

DoNotResetCalc

The load flow results will not be reset even the load flow calculation fails.

```
int ComLdf.DoNotResetCalc(int doNotReset)
```

ARGUMENTS

doNotReset
Specifies whether the results shall be reset or not.

0	Reset load flow results if load flow fails.
1	Load flow results will remain even load flow fails.

RETURNS

Always return 0.

EstimateLoading

Estimate the loading of all branch elements if the power injections of given set of terminals are changed. The changed power for each terminal is stored in dpl1 (active power) and dpl2 (reactive power).

```
int ComLdf.EstimateLoading(list nodes,
                           int init
                           )
```

ARGUMENTS

nodes The terminals whose power injections are changed.

init Initialisation of sensitivities.

0	No need to calculate sensitivities; it assumes that sensitivities have been calculated before hand.
1	Sensitivities will be newly calculated.

RETURNS

0	On success.
1	On error.

EstimateOutage

Estimate the loading of all branches with outages of given set of branch elements.

```
int ComLdf.EstimateOutage(list branches,
                          int  init
                          )
```

ARGUMENTS

branches The branch elements to be in outage.

init Initialisation of sensitivities.

0	No need to calculate sensitivities; it assumes that sensitivities have been calculated before hand.
1	Sensitivities will be newly calculated.

RETURNS

0	On success.
1	On error.

Execute

Performs a load flow analysis on a network. Results are displayed in the single line graphic and available in relevant elements.

```
int ComLdf.Execute()
```

RETURNS

0	OK
1	Load flow failed due to divergence of inner loops.
2	Load flow failed due to divergence of outer loops.

IsAC

Check whether this load flow is configured as AC method or not.

```
int ComLdf.IsAC()
```

RETURNS

0	Is a DC method.
1	Is an AC method.

IsBalanced

Check whether this load flow command is configured as balanced or unbalanced.

```
int ComLdf.IsBalanced()
```

RETURNS

Returns true if the load flow is balanced.

IsDC

Check whether this load flow is configured as DC method or not.

```
int ComLdf.IsDC()
```

RETURNS

0	Is an AC method.
1	Is a DC method.

PrintCheckResults

Shows the verification report in the output window.

```
int ComLdf.PrintCheckResults()
```

RETURNS

Always return 1.

SetOldDistributeLoadMode

Set the old scaling mode in case of Distributed Slack by loads.

```
None ComLdf.SetOldDistributeLoadMode(int iOldMode)
```

ARGUMENTS

iOldMode The flag showing if the old model is used.

0	Use standard mode.
1	Use old mode.

4.4.14 ComLink

Overview

LoadMicroSCADAFile
ReceiveData
SendData
SentDataStatus
SetOPCReceiveQuality
SetSwitchShcEventMode

LoadMicroSCADAFile

Reads in a MicroSCADA snapshot file.

```
int ComLink.LoadMicroSCADAFile(str filename,
                               [int populate]
                               )
```

ARGUMENTS

filename name of the file to read

populate (optional)
determines whether new values should be populated to the network elements (0=no, 1=yes)

RETURNS

0	On success.
1	On error.

ReceiveData

Reads and processes values for all (in PowerFactory configured) items from OPC server (OPC only).

```
int ComLink.ReceiveData([int force])
```

ARGUMENTS

force (optional)

0	(default) Processes changed values (asynchronously) received by PowerFactory via callback
1	Forces (synchronous) reading and processing of all values (independet of value changes)

RETURNS

Number of read items

SendData

Sends values from configured measurement objects to OPC server (OPC only).

```
int ComLink.SendData([int force])
```

ARGUMENTS

force (optional)

0	(default) Send only data that have been changed and difference between old and new value is greater than configured deadband
1	Forces writing of all values (independet of previous value)

RETURNS

Number of written items

SentDataStatus

Outputs status of all items marked for sending to output window.

```
int ComLink.SentDataStatus()
```

RETURNS

Number of items configured for sending.

SetOPCReceiveQuality

Allows to override the actual OPC receive quality by this value. (Can be used for testing.)

```
int ComLink.SetOPCReceiveQuality(int quality)
```

ARGUMENTS

quality new receive quality (bitmask)

RETURNS

0	On success.
1	On error.

SetSwitchShcEventMode

Configures whether value changes for switches are directly transferred to the object itself of whether shc switch events shall be created instead.

```
None ComLink.SetSwitchShcEventMode(int enabled)
```

ARGUMENTS

enabled

0	Values are directly written to switches
1	For each value change a switch event will be created

4.4.15 ComMerge

Overview

CheckAssignments
Compare
CompareActive
ExecuteRecording
ExecuteWithActiveProject
GetCorrespondingObject
GetModification
GetModificationResult
GetModifiedObjects
Merge
PrintComparisonReport
PrintModifications
Reset
SetAutoAssignmentForAll
SetObjectsToCompare
ShowBrowser
WereModificationsFound

CheckAssignments

Checks if all assignments are correct and merge can be done.

```
int ComMerge.CheckAssignments()
```

RETURNS

0	On success.
1	Canceled by user.
2	Missing assignments found.
3	Conflicts found.
4	On other errors.

Compare

Starts a comparison according to the settings in this ComMerge object. The merge browser is not shown.

```
int ComMerge.Compare()
```

CompareActive

Starts a comparison according to the settings in this ComMerge object. The merge browser is not shown. Can compare with the active project.

```
int ComMerge.CompareActive()
```

ExecuteRecording

Starts a comparison according to the settings in this ComMerge object and shows the merge browser. Records modifications in the active scenario and/or expansion stage of the target project.

```
int ComMerge.ExecuteRecording()
```

ExecuteWithActiveProject

Starts a comparison according to the settings in this ComMerge object and shows the merge browser. Can compare with the active project.

```
None ComMerge.ExecuteWithActiveProject()
```

GetCorrespondingObject

Searches corresponding object for given object.

```
DataObject ComMerge.GetCorrespondingObject(DataObject sourceObj,
                                           [int target]
                                           )
```

ARGUMENTS

sourceObj
Object for which corresponding object is searched.

target

0	Get corresponding object from “Base” (default)
1	Get corresponding object from “1st”
2	Get corresponding object from “2nd”

RETURNS

object	Corresponding object.
None	Corresponding object not found.

GetModification

Gets kind of modification between corresponding objects of “Base” and “1st” or “2nd”.

```
int ComMerge.GetModification(DataObject sourceObj,
                             [int taget]
                             )
```

ARGUMENTS

sourceObj
Object from any source for which modification is searched.

target

1	Get modification from “Base” to “1st” (default)
2	Get modification from “Base” to “2nd”

RETURNS

0	On error.
1	No modifications (equal).
2	Modified.
3	Added in “1st”/“2nd”.
4	Removed in “1st”/“2nd”.

GetModificationResult

Gets kind of modifications between compared objects in “1st” and “2nd”.

```
int ComMerge.GetModificationResult(DataObject obj)
```

ARGUMENTS

obj Object from any source for which modification is searched.

RETURNS

0	On error.
1	No modifications (equal).
2	Same modifications in “1st” and “2nd” (no conflict).
3	Different modifications in “1st” and “2nd” (conflict).

GetModifiedObjects

Gets all objects with a certain kind of modification.

```
list ComMerge.GetModifiedObjects(int modType,
                                 [int modSource]
                                 )
```

ARGUMENTS

modType

1	get unmodified objects
2	get modified objects
3	get added objects
4	get removed obejcts

modSource

1	consider modification between “Base” and “1st” (default)
2	consider modification between “Base” and “2nd”

RETURNS

Set with matching objects.

Unmodified, modified and added objects are always from given “modSource”, removed objects are always from “Base” .

Merge

Checks assignments, merges modifications according to assignments into target and prints merge report to output window.

```
None ComMerge.Merge(int printReport)
```

ARGUMENTS

printReport

1	print merge report (default)
0	do not print merge report

always set to 0 in paste and split mode

PrintComparisonReport

Prints the modifications of all compared objects as a report to the output window.

```
None ComMerge.PrintComparisonReport(int mode)
```

ARGUMENTS

mode

0	no report
1	only modified compare objects
2	all compare objects

PrintModifications

Prints modifications of given objects (if any) to the output window.

```
int ComMerge.PrintModifications(list objs)
int ComMerge.PrintModifications(DataObject obj)
```

ARGUMENTS

objs	Set of objects for which the modifications are printed.
obj	Object for which the modifications are printed.

RETURNS

0	On error: object(s) not found in comparison.
1	On success: modifications were printed.

Reset

Resets/clears and deletes all temp. object sets, created internally for the comparison.

```
None ComMerge.Reset()
```

SetAutoAssignmentForAll

Sets the assignment of all compared objects automatically.

```
None ComMerge.SetAutoAssignmentForAll(int conflictVal)
```

ARGUMENTS

conflictVal
Assignment of compared objects with undefined automatic values (e.g. conflicts)

0	no assignment
1	assign from “Base”
2	assign from 1st
3	assign from 2nd

SetObjectsToCompare

Sets top level objects for comparison.

```
None ComMerge.SetObjectsToCompare(DataObject base,
                                  [DataObject first,]
                                  [DataObject second]
                                  )
```

ARGUMENTS

base	Top level object to be set as “Base”
first	Top level object to be set as “1st”
second	Top level object to be set as “2nd”

ShowBrowser

Shows merge browser with initialized settings and all compared objects. Can only be called after a comparison was executed.

```
int ComMerge.ShowBrowser()
```

RETURNS

0	The browser was left with ok button.
1	The browser was left with cancel button.
2	On error.

WereModificationsFound

Checks, if modifications were found in comparison.

```
int ComMerge.WereModificationsFound()
```

RETURNS

0	All objects in comparison are equal.
1	Modifications found in comparison.

4.4.16 ComMot

Overview

GetMotorConnections

Finds the cables connecting the motor to the switch.

```
list ComMot.GetMotorConnections(DataObject motor)
```

ARGUMENTS

motor	The motor element

RETURNS

Returns the set of cables connecting the motor to the switch.

GetMotorSwitch

Finds the switch which will connect the motor to the network.

```
DataObject ComMot.GetMotorSwitch(DataObject motor)
```

ARGUMENTS

motor	The motor element

RETURNS

Returns the switch element.

GetMotorTerminal

Finds the terminal to which the motor will be connected.

```
DataObject ComMot.GetMotorTerminal(DataObject motor)
```

ARGUMENTS

motor	The motor element

RETURNS

Returns the terminal element.

4.4.17 ComNmink

Overview

AddRef
Clear
GetAll

AddRef

Adds shortcuts to the objects to the existing selection.

```
None ComNmink.AddRef(DataObject O)
None ComNmink.AddRef(list S)
```

ARGUMENTS

O(optional)
an object

S(optional)
a Set of objects

Clear

Delete all contents, i.e. to empty the selection.

```
None ComNmink.Clear()
```

GetAll

Returns all objects which are of the class 'ClassName'.

```
list ComNmink.GetAll(str className)
```

ARGUMENTS

className
The object class name.

RETURNS

The set of objects

4.4.18 ComOmr

Overview

GetFeeders
GetOMR
GetRegionCount

GetFeeders

Get all feeders for which optimal manual switches have been determined. This function can be used after execution of an Optimal Manual Restoration command only.

```
list ComOmr.GetFeeders()
```

RETURNS

The set of all feeders used for optimisation.

GetOMR

Get terminal and connected optimal manual switches determined by the optimisation for the given feeder and its region(pocket) of the given index. For a detailed description of a pocket, please consult the manual. This function can be used after execution of an Optimal Manual Restoration command only.

```
list ComOmr.GetOMR(DataObject arg0,
                   int arg1
                   )
```

ARGUMENTS

arg0	The feeder to derive the resulting optimal terminal with its connected (optimal) manual switches for.
arg1	The index of the region(pocket) inside the given feeder to derive the resulting optimal terminal with its connected (optimal) manual switches for.

RETURNS

The resulting optimal terminal with its connected (optimal) manual switches for the region in the feeder.

GetRegionCount

Get total number of regions(pockets) separated by infeeding point, feeder ends and certain switches for the provided feeder. For a detailed description of a pocket, please consult the manual. This function can be used after execution of an Optimal Manual Restoration command only.

```
int ComOmr.GetRegionCount(DataObject feeder)
```

ARGUMENTS

feeder	Feeder to derive number of regions(pockets) for.

RETURNS

Number of regions(pockets) for the feeder.

4.4.19 ComOpc

Overview

ReceiveData
SendData

ReceiveData

Reads and processes values for all (in PowerFactory configured) items from OPC server (OPC only).

```
int ComOpc.ReceiveData([int force])
```

ARGUMENTS

force (optional)

1	Forces (synchronous) reading and processing of all values (independet of value changes)

RETURNS

1 if successfully received data -1 if an error occured -2 if the link is not connected

SendData

Sends values from configured measurement objects to OPC server (OPC only).

```
int ComOpc.SendData([int force])
```

ARGUMENTS

force (optional)

0	(default) Send only data that have been changed and difference between old and new value is greater than configured deadband
1	Forces writing of all values (independet of previous value)

RETURNS

1 if successfully received data -1 if an error occured -2 if the link is not connected

4.4.20 ComOutage

Overview

ContinueTrace
ExecuteTime
GetObject
RemoveEvents
SetObjs
StartTrace
StopTrace

ContinueTrace

Continue the next step of the trace.

```
int ComOutage.ContinueTrace()
```

RETURNS

0	On success.
$\neq 0$	On error.

ExecuteTime

Execute contingency (with multiple time phase) for the given time.

```
int ComOutage.ExecuteTime(float time)
```

ARGUMENTS

time	the given time to be executed.

RETURNS

$= 0$	On success.
$\neq 0$	On error.

GetObject

Get the element stored in line number “line” in the table of ComOutage. The line index starts with 0.

```
DataObject ComOutage.GetObject(int line)
```

ARGUMENTS

line	line index, if index exceeds the range None is returned

RETURNS

the element of line “line” in the table.

RemoveEvents

Remove all events defined in this contingency.

```
None ComOutage.RemoveEvents ([int info])
None ComOutage.RemoveEvents (str type)
None ComOutage.RemoveEvents (int info, str type)
None ComOutage.RemoveEvents (str type, int info)
```

ARGUMENTS

type

none	Hidden objects are ignored and not added to the set
'Lod'	remove all EvtLod
'Gen'	remove all EvtGen
'Switch'	remove all EvtSwitch

info(optional)

1	show info message in output window (default)
0	do not show info message

SetObjs

To fill up the "interrupted components" with given elements.

Sets the list of objects according to S.

```
int ComOutage.SetObjs(list S)
```

ARGUMENTS

S the set of objects

RETURNS

0	On success.
1	On error.

StartTrace

Start trace all post fault events of this contingency.

```
int ComOutage.StartTrace()
```

RETURNS

$= 0$	On success.
$\neq 0$	On error.

StopTrace

To stop the trace.

```
int ComOutage.StopTrace([int msg])
```

ARGUMENTS

msg (optional)
Emit messages or not.

0	Suppress messages.
1	Emit messages.

RETURNS

$= 0$	On success.
$\neq 0$	On error.

4.4.21 ComPython

Overview

GetExternalObject
GetInputParameterDouble
GetInputParameterInt
GetInputParameterString
SetExternalObject
SetInputParameterDouble
SetInputParameterInt
SetInputParameterString

GetExternalObject

Gets the external object defined in the ComPython edit dialog.

```
[int error,
DataObject value] ComPython.GetExternalObject(str name)
```

ARGUMENTS

name	Name of the external object parameter.
value (out)	The external object.

RETURNS

0	On success.
1	On error.

SEE ALSO

ComPython.SetExternalObject(), ComPython.GetInputParameterInt(), ComPython.GetInputParameterDouble(), ComPython.GetInputParameterString()

GetInputParameterDouble

Gets the double input parameter value defined in the ComPython edit dialog.

```
[int error,
float value] ComPython.GetInputParameterDouble(str name)
```

ARGUMENTS

name	Name of the double input parameter.
value (out)	Value of the double input parameter.

RETURNS

0	On success.
1	On error.

SEE ALSO

ComPython.SetInputParameterDouble(), ComPython.GetInputParameterInt(), ComPython.GetInputParameterString(), ComPython.GetExternalObject()

GetInputParameterInt

Gets the integer input parameter value defined in the ComPython edit dialog.

```
[int error,
int value ] ComPython.GetInputParameterInt(str name)
```

ARGUMENTS

name Name of the integer input parameter.

value (out)
Value of the integer input parameter.

RETURNS

0	On success.
1	On error.

SEE ALSO

ComPython.SetInputParameterInt(), ComPython.GetInputParameterDouble(), ComPython.GetInputParameterString(), ComPython.GetExternalObject()

GetInputParameterString

Gets the string input parameter value defined in the ComPython edit dialog.

```
[int error,
str value ] ComPython.GetInputParameterString(str name)
```

ARGUMENTS

name Name of the string input parameter.

value (out)
Value of the string input parameter.

RETURNS

0	On success.
1	On error.

SEE ALSO

ComPython.SetInputParameterString(), ComPython.GetInputParameterInt(), ComPython.GetInputParameterDouble(), ComPython.GetExternalObject()

SetExternalObject

Sets the external object defined in the ComPython edit dialog.

```
int ComPython.SetExternalObject(str name,
                                DataObject value
                                )
```

ARGUMENTS

name	Name of the external object parameter.
value	The external object.

RETURNS

0	On success.
1	On error.

SEE ALSO

ComPython.GetExternalObject(), ComPython.SetInputParameterInt(), ComPython.SetInputParameterDouble(), ComPython.SetInputParameterString()

SetInputParameterDouble

Sets the double input parameter value defined in the ComPython edit dialog.

```
int ComPython.SetInputParameterDouble(str name,
                                      double value
                                      )
```

ARGUMENTS

name	Name of the double input parameter.
value	Value of the double input parameter.

RETURNS

0	On success.
1	On error.

SEE ALSO

ComPython.GetInputParameterDouble(), ComPython.SetInputParameterInt(), ComPython.SetInputParameterString(), ComPython.SetExternalObject()

SetInputParameterInt

Sets the integer input parameter value defined in the ComPython edit dialog.

```
int ComPython.SetInputParameterInt(str name,
                                   int value
                                   )
```

ARGUMENTS

name	Name of the integer input parameter.
value	Value of the integer input parameter.

RETURNS

0	On success.
1	On error.

SEE ALSO

ComPython.GetInputParameterInt(), ComPython.SetInputParameterDouble(), ComPython.SetInputParameterString(), ComPython.SetExternalObject()

SetInputParameterString

Sets the string input parameter value defined in the ComPython edit dialog.

```
int ComPython.SetInputParameterString(str name,
                                      str value
                                      )
```

ARGUMENTS

name	Name of the string input parameter.
value	Value of the string input parameter.

RETURNS

0	On success.
1	On error.

SEE ALSO

ComPython.GetInputParameterString(), ComPython.SetInputParameterInt(), ComPython.SetInputParameterDouble(), ComPython.SetExternalObject()

4.4.22 ComRel3

Overview

AnalyseElmRes
ExeEvt
OvlAlleviate
RemoveEvents
RemoveOutages
ValidateConstraints

AnalyseElmRes

Evaluate the results object created by the last calculation. Performs exactly the same as pressing the button 'Perform Evaluation of Result File' in the dialogue box of the command.

```
int ComRel3.AnalyseElmRes([int error])
```

ARGUMENTS

error (optional)

0	do not display an error message (default)
1	display error messages in case of errors

RETURNS

$= 0$	On success.
$\neq 0$	On error.

ExeEvt

Executes a given event.

```
None ComRel3.ExeEvt([DataObject event])
```

ARGUMENTS

event	The event that shall be executed.

OvlAlleviate

Performs an overload alleviation for given events.

```
int ComRel3.OvlAlleviate([list preCalcEvents])
```

ARGUMENTS

preCalcEvents (optional)
The events which will be executed before the calculation.

RETURNS

0	On success.
1	Failure in load flow.
2	No overloading detected.
> 2	On error.

RemoveEvents

Removes all events stored in all contingencies (*.ComContingency) inside the reliability command.

```
None ComRel3.RemoveEvents()
```

RemoveOutages

Removes all contingency definitions (*.ComContingencies) stored inside the reliability command.

```
None ComRel3.RemoveOutages([int msg])
```

ARGUMENTS

msg(optional)

1	Show info message in output window (default value).
0	Do not emit messages.

ValidateConstraints

Checks if the restoration of a contingency violates any constraint according to the current settings of the reliability calculation. These do not necessarily have to be the settings used during calculation. Of course the selected calculation method of ComRel3 has to be 'Load flow analysis' to check for constraint violations.

```
int ComRel3.ValidateConstraints(DataObject contingency)
```

ARGUMENTS

contingency
The contingency which will be checked for constraint violations.

RETURNS

0	No constraint violations, or all constraint violations could be solved.
1	Constraints are violated.
-1	Contingency not valid.

4.4.23 ComRelpost

Overview

CalcContributions

Calculates the contributions to load interruptions of the loads that are passed to this function. The loads can be e.g. inside a feeder or a zone as well. If nothing is passed as input all loads will be analysed.

```
int ComRelpost.CalcContributions([list elements])
```

ARGUMENTS

elements (optional)
Elements (Loads) for which the contributions shall be calculated (default: all loads, if no argument is passed).

RETURNS

0	Calculation successful.
1	On error.

GetContributionOfComponent

Gets the contributions of a component to a certain reliability indice.

```
float ComRelpost.GetContributionOfComponent(int componentNr,
                                            str indice)
```

ARGUMENTS

componentNr
1. Lines
2. Cables
3. Transformers
4. Busbars
5. Generators
6. Common Modes
7. Double Earth Faults

indice Avalaible indices are: ’SAIFI’, ’SAIDI’, ’ASIFI’, ’ASIDI’, ’ENS’, ’EIC’

RETURNS

The contribution of this component to this reliability indice.

4.4.24 ComRelreport

Overview

GetContingencies
GetContributionOfComponent

GetContingencies

Gets all contingencies of reliability for reporting.

```
list ComRelpost.GetContingencies()
```

RETURNS

All contingencies of reliability for reporting.

GetContributionOfComponent

Is described in ComRelpost.GetContributionOfComponent().

```
float ComRelreport.GetContributionOfComponent(int componentNr,
                                              str indice)
```

4.4.25 ComRes

Overview

ExportFullRange
FileNmResNm

ExportFullRange

Executes the export command for the whole data range.

```
None ComRes.ExportFullRange()
```

FileNmResNm

Sets the filename for the data export to the name of the result object being exported (classes: ElmRes, IntComtrade)

```
None ComRes.FileNmResNm()
```

4.4.26 ComShc

Overview

ExecuteRXSweep
GetFaultType
GetOverLoadedBranches
GetOverLoadedBuses

ExecuteRXSweep

Calculates RX Sweep. If no impedance passed, the value from the command shall be used. If argument passed then the impedance changes are stored to the command (Rf, Xf).

```
int ComShc.ExecuteRXSweep()
int ComShc.ExecuteRXSweep(float Zr,
                          float Zi
                          )
```

ARGUMENTS

Zr	Impedance real part
Zi	Impedance imaginary part

RETURNS

$= 0$	On success.
$\neq 0$	On error.

GetFaultType

Returns the short-circuit fault type.

```
int ComShc.GetFaultType()
```

RETURNS

0	three phase fault
1	single phase to ground
2	two phase fault
3	two phase to ground fault
4	three phase unbalanced fault
5	single phase to neutral fault
6	single phase, neutral to ground fault
7	two phase to neutral fault
8	two phase, neutral to ground fault
9	three phase to neutral fault
10	three phase, neutral to ground fault
20	DC fault

GetOverLoadedBranches

Get overloaded branches after a short-circuit calculation.

```
[int error,
list branches] ComShc.GetOverLoadedBranches(float ip,
                                            float ith)
```

ARGUMENTS

ip	Max. peak-current loading, in %
ith	Max. thermal loading, in %
branches (out)	Set of branches which are checked

RETURNS

$= 0$	On error or 0 branches found.
$\neq 0$	Number of branches.

EXAMPLE

GetOverLoadedBuses

Get overloaded buses after a short-circuit calculation.

```
[int error,
list buses] ComShc.GetOverLoadedBuses(float ip,
                                      float ith)
```

ARGUMENTS

ip	Max. peak-current loading, in %
ith	Max. thermal loading, in %
buses (optional, out)	Set of buses which are checked

RETURNS

$= 0$	On error or 0 buses found.
$\neq 0$	Number of buses.

EXAMPLE

4.4.27 ComShctrace

Overview

BlockSwitch

Blocks a switch from operating for the remainder of the trace.

```
int ComShctrace.BlockSwitch(DataObject switchDevice)
```

ARGUMENTS

switchDevice
Switch device to block.

RETURNS

0	Switch can not be blocked (e.g. because it already operated).
1	Switch is blocked.

ExecuteAllSteps

Executes all steps of the short circuit trace. This function requires the trace to be already running

```
int ComShctrace.ExecuteAllSteps()
```

RETURNS

0	No error occourred, trace is complete.
!=0	An error occurred, calculation was reset.

SEE ALSO

ComShctrce.ExecuteInitialStep()

ExecuteInitialStep

Executes the first step of the short circuit trace.

```
int ComShctrace.ExecuteInitialStep()
```

RETURNS

0	No error occourred, the short-circuit trace is now running.
!=0	An error occurred, calculation was reset.

ExecuteNextStep

Executes the next step of the short circuit trace. This function requires the trace to be already running

```
int ComShctrace.ExecuteNextStep()
```

RETURNS

0	No error occourred, step was executed .
!=0	An error occurred, calculation was reset.

SEE ALSO

ComShctrce.ExecuteInitialStep()

GetBlockedSwitches

Returns all switches which are currently blocked.

```
list ComShctrace.GetBlockedSwitches()
```

RETURNS

All blocked switches.

GetCurrentTimeStep

Returns the current time step of the trace in seconds.

```
float ComShctrace.GetCurrentTimeStep()
```

RETURNS

The current time step in [s].

GetDeviceSwitches

Returns all switches operated by a protection device.

```
list ComShctrace.GetDeviceSwitches(DataObject device)
```

ARGUMENTS

device	Protection device to get the switches for.

RETURNS

All switches devices operated by the protection device.

GetDeviceTime

Returns the time a protection device operated or will operate at.

```
float ComShctrace.GetDeviceTime(DataObject device)
```

ARGUMENTS

device	Protection device to get the time for.

RETURNS

The tripping time of the device itself, if the device already tripped, or the prospective tripping time.

GetNonStartedDevices

Returns all protection devices which are not started.

```
list ComShctrace.GetNonStartedDevices()
```

RETURNS

All protection devices which are not started.

GetStartedDevices

Returns all started but not yet tripped protection devices.

```
list ComShctrace.GetStartedDevices()
```

RETURNS

All started but not yet tripped protection devices.

GetSwitchTime

Returns the time a switch device operated or will operate at.

```
float ComShctrace.GetSwitchTime(DataObject device,
                                DataObject switchDevice
                                )
```

ARGUMENTS

device	Reference protection device for the switch.
device	Switch device to get the time for.

RETURNS

The tripping time of the switch device, based on the tripping time of the reference protection device. If the switch already operated, the time of operation will be returned.

GetTrippedDevices

Returns all protection devices already tripped.

```
list ComShctrace.GetTrippedDevices()
```

RETURNS

All protection devices already tripped.

NextStepAvailable

Indicates whether or not a next time step can be executed.

```
int ComShctrace.NextStepAvailable()
```

RETURNS

0	Next step is not available, the trace is completed.
1	A next step is available.

4.4.28 ComSimoutage

Overview

AddCntcy

Executes an (additional) ComOutage, without resetting results. The results of the outage analysis will be added to the intermediate results. Object "O" must be a ComOutage object. If the outage definition has already been analyzed, it will be ignored. The ComOutage will be renamed to "name" when "name" is given.

```
int ComSimoutage.AddCntcy(DataObject O,
                          [str name]
                          )
```

ARGUMENTS

O	The ComOutage object
name	A name for the outage

RETURNS

0	On success.
1	On error.

AddContingencies

Adds contingencies for fault cases/groups selected by the user to the command. Shows a modal window with the list of available fault cases and groups. Functionality as “Add Cases/Groups” button in dialog.

```
None ComSimoutage.AddContingencies()
```

ClearCont

Reset existing contingency analysis results and delete existing contingency cases.

```
int ComSimoutage.ClearCont()
```

RETURNS

0	On success.
1	On error.

CreateFaultCase

Create fault cases from the given elements.

```
int ComSimoutage.CreateFaultCase(list elms,
                                 int mode,
                                 [int createEvt],
                                 [object folder]
                                 )
```

ARGUMENTS

elms Selected elements to create fault cases.

mode How the fault cases are created:

0	Single fault case containing all elements.
1	n-1 (multiple cases).
2	n-2 (multiple cases).
3	Collecting coupling elemnts and create fault cases for line couplings.

createEvt (optional)
Switch event:

0	Do NOT create switch events.
1	Create switch events.

folder (optional)
Folder in which the fault case is stored.

RETURNS

0	On success.
1	On error.

Execute

Execute contingency analysis.

```
int ComSimoutage.Execute()
```

RETURNS

0	On success.
1	On error.

ExecuteCntcy

Execute additional contingency analysis without resetting results.

```
int ComSimoutage.ExecuteCntcy()
```

RETURNS

0	On success.
1	On error.

GetNTopLoadedElms

To get certain number of top loaded components (most close to its limit).

```
list ComSimoutage.GetNTopLoadedElms(int number)
```

ARGUMENTS

number The number of elements to be found.

elements (out)
The top loaded elements.

MarkRegions

To execute Region marker for certain system status (like prefault, post fault etc.), which will indentifies energizing mode for each element.

```
int ComSimoutage.MarkRegions(int stage)
```

ARGUMENTS

stage which system stage to be analyzed, 0¡=stage¡=2

RETURNS

0	On success.
1	On error.

RemoveContingencies

Removes all contingencies from the command. Functionality as ”Remove All” button in dialog.

```
None ComSimoutage.RemoveContingencies()
```

Reset

Resets the intermediate results of the outage simulation.

```
int ComSimoutage.Reset()
```

RETURNS

0	On success.
1	On error.

SetLimits

Sets the limits for the outage simulation.

```
int ComSimoutage.SetLimits(float vlmin,
                           float vlmax,
                           float ldmax)
```

ARGUMENTS

vlmin	The minimum voltage
vlmax	The maximum voltage
ldmax	The maximum loading

Update

To update contingency cases via topology search. It will find interrupted elements, required switch actions for each contingency.

```
int ComSimoutage.Update()
```

RETURNS

0	On success.
1	On error.

4.4.29 ComSvgexport

Overview

SetFileName
SetObject
SetObjects

SetFileName

Sets SVG file for export.

```
None ComSvgexport.SetFileName(str path)
```

ARGUMENTS

path	Path of target SVG file

SetObject

Sets annotation layer or group for export.

```
None ComSvgexport.SetObject(DataObject obj)
```

ARGUMENTS

obj	Annotation layer (IntGrflayer) or group (IntGrfgroup) to be exported

SetObjects

Sets annotation layers and groups for export.

```
None ComSvgexport.SetObjects(set objs)
```

ARGUMENTS

objs	Set of annotation layers (IntGrflayer) and/or groups (IntGrfgroup) to be exported

4.4.30 ComSvgimport

Overview

SetFileName
SetObject

SetFileName

Sets source SVG file for import.

```
None ComSvgimport.SetFileName(str path);
```

ARGUMENTS

path	Path of SVG file to be imported

SetObject

Sets target annotation layer or group for import.

```
None ComSvgimport.SetObject(DataObject obj);
```

ARGUMENTS

obj	Target annotation layer (IntGrflayer) or group (IntGrfgroup)

4.4.31 ComTasks

Overview

AppendCommand
AppendStudyCase
RemoveCmdsForStudyCaseRow
RemoveStudyCases
SetResultsFolder

AppendCommand

Appends a command for calculation.

```
int ComTasks.AppendCommand(DataObject command,
                           [int studyCaseRow]
                           )
```

RETURNS

0	Command could not be added for calculation.
1	Command has been successfully added for calculation.

ARGUMENTS

command
Command to add for calculation.

studyCaseRow

$<= 0$	Command is added to the list of commands for its study case.
> 0	Optionally, the row in the study case table containing the study case for which this command shall be added can be passed. This is helpful, e.g., if a study case has been added multiple times for calculation with different command lists.

AppendStudyCase

Appends a study case to the list of study cases for calculation.

```
int ComTasks.AppendStudyCase(DataObject studyCase)
```

RETURNS

0	Study case could not be added for calculation.
1	Study case has been successfully added for calculation.

ARGUMENTS

studyCase
Study case to add for calculation.

RemoveCmdsForStudyCaseRow

Removes all commands selected for calculation for a given row in the study case table.

```
int ComTasks.RemoveCmdsForStudyCaseRow(int studyCaseRow)
```

RETURNS

0	Commands could not be removed from calculation.
1	All commands of study case row were successfully removed from calculation.

ARGUMENTS

studyCaseRow
The row in the study case table containing the study case for which all commands shall be removed.

RemoveStudyCases

Removes all selected study cases from calculation.

```
None ComTasks.RemoveStudyCases()
```

SetResultsFolder

Set a folder to store results for a given row in the study case table.

```
int ComTasks.SetResultsFolder(DataObject folder, int studyCaseRow)
```

RETURNS

0	New folder could not be set as results folder for the given row in the study case table.
1	Folder was successfully set as resuls folder for given row in the study case table.

ARGUMENTS

folder	The new folder to store results in.
studyCaseRow	The row in the study case table containing the study case for which results folder shall be set.

4.4.32 ComTececo

Overview

UpdateTablesByCalcPeriod

UpdateTablesByCalcPeriod

Update all calculation points with respect to a new start- and end year

```
int ComTececo.UpdateTablesByCalcPeriod(float start,
                                       float end
                                       )
```

ARGUMENTS

start	Start year of the study period
end	End year of the study period

RETURNS

0	Calculation points have been successfully set.
1	Invalid input data: end year of study period must be greater or equal to start year.

4.4.33 ComTransfer

Overview

GetTransferCalcData
IsLastIterationFeasible

GetTransferCalcData

The function returns the calculated transfer capacity and the total number of iteration after the transfer capacity command has been executed.

```
[float transferCapacity,
int totalIterations    ] ComTransfer.GetTransferCalcData()
```

ARGUMENTS

transferCapacity (out)
Transfer capacity value at the last feasible iteration.

totalIterations (out)
Total iteration number.

IsLastIterationFeasible

The function verifies if the last transfer calculation iteration resulted in the feasible solution or not.

```
int ComTransfer.IsLastIterationFeasible()
```

4.4.34 ComUcte

Overview

SetBatchMode

SetBatchMode

The batch mode allows to suppress all messages except error and warnings. This can be useful when used in scripts where additional output might be confusing.

```
None ComUcte.SetBatchMode(int enabled)
```

ARGUMENTS

enabled

0	disables batch mode, all messages are printed to output window (default).
1	enables batch mode, only error and warning messages are printed to output window.

4.4.35 ComUcteexp

Overview

BuildNodeNames

Builds the node names as used in UCTE export and makes them accessible via :UcteNode-Name attribute. The node names will only be available as long as topology has not been changed. They must be re-build after any topology relevant modification.

Furthermore, the method fills the quick access cache given by the cache index for node names and branch topologies as used in UCTE export. The quick access cache endures also topology changes. The cache index is optional. If no cache index is given the default quick access cache is used.

```
int ComUcteexp.BuildNodeNames([int cacheIndex])
```

ARGUMENTS

cacheIndex (optional)
Index of the quick access cache (must be greater than or equals to 0)

RETURNS

0	On success.
1	0n error (e.g. load flow calculation failed).

DeleteCompleteQuickAccess

Deletes all quick access caches.

```
None ComUcteexp.DeleteCompleteQuickAccess()
```

ExportAndInitQuickAccess

Performs an UCTE export and fills the quick access cache given by the cache index.

```
None ComUcteexp.ExportAndInitQuickAccess(int cacheIndex)
```

ARGUMENTS

cacheIndex
Index of the quick access cache (must be greater than or equals to 0)

GetConnectedBranches

Determines the connected branches for the given terminal from the quick access cache given by the optional cache index. If no cache index is given the default quick access cache is used.

```
list ComUcteexp.GetConnectedBranches(DataObject terminal
                                     [int cacheIndex])
```

ARGUMENTS

terminal Terminal to determine the connected branches from

connectedBranches (out)
Connected branches for the given terminal

cacheIndex (optional)
Index of the quick access cache (must be greater than or equals to 0)

GetFromToNodeNames

Determines the UCTE node names of the branch ends from the quick access cache given by the optional cache index. If no cache index is given the default quick access cache is used.

```
[str nodeNameFrom,
str nodeNameTo   ] ComUcteexp.GetFromToNodeNames(DataObject branch,
                                                 [int cacheIndex])
```

ARGUMENTS

branch Branch to find the UCTE node names from

nodeNameFrom (out)
UCTE node name of start node

nodeNameTo (out)
UCTE node name of end node

cacheIndex (optional)
Index of the quick access cache (must be greater than or equals to 0)

GetOrderCode

Determines the order code of the given branch element as used for UCTE export from the quick access cache given by the optional cache index. If no cache index is given the default quick access cache is used.

```
str ComUcteexp.GetOrderCode(DataObject branch,
                            [int cacheIndex])
```

ARGUMENTS

branch Branch element to get the UCTE order code from

orderCode (out)
Order code of the given branch element

cacheIndex (optional)
Index of the quick access cache (must be greater than or equals to 0)

GetUcteNodeName

Determines the node name of the given terminal as used for UCTE export from the quick access cache given by the optional cache index. If no cache index is given the default quick access cache is used.

```
str ComUcteexp.GetOrderCode(DataObject terminal,
                            [int cacheIndex])
```

ARGUMENTS

terminal Terminal to get the UCTE node name from

ucteNodeName (out)
UCTE node name of the given terminal

cacheIndex (optional)
Index of the quick access cache (must be greater than or equals to 0)

InitQuickAccess

Initializes the quick access cache given by the optional cache index. The quick acess cache contains node names and branch topologies as used in UCTE export and endures topology changes. *InitQuickAccess()* requires a successful executed UCTE export as pre-condition. The cache index is optional. If no cache index is given the default quick access cache is used.

```
None ComUcteexp.InitQuickAccess([int cacheIndex])
```

ARGUMENTS

cacheIndex (optional)
Index of the quick access cache (must be greater than or equals to 0)

QuickAccessAvailable

Checks if the quick access cache given by the optional cache index is available. If no cache index is given the default quick access cache is checked for availability.

```
None ComUcteexp.QuickAccessAvailable([int cacheIndex])
```

ARGUMENTS

cacheIndex (optional)
Index of the quick access cache (must be greater than or equals to 0)

ResetQuickAccess

Resets the given quick access cache for node names and branch topologies as used in UCTE export. The cache index is optional. If no cache index is given the default quick access cache is reset.

```
None ComUcteexp.ResetQuickAccess([int cacheIndex])
```

ARGUMENTS

cacheIndex (optional)
Index of the quick access cache (must be greater than or equals to 0)

SetGridSelection

Configures the selected grids in the UCTE export command.

```
None ComUcteexp.SetGridSelection(list gridsToExport)
```

ARGUMENTS

gridsToExport
Grids (instances of class ElmNet) to be selected for export. All not contained grids will be de-selected.

4.5 Settings

4.5.1 SetCluster

Overview

CalcCluster
GetNumberOfClusters

CalcCluster

Performs a load flow calculation for the cluster index passed to the function. To execute properly this function requires that a valid load flow result is already calculated before calling it.

```
int SetCluster.CalcCluster(int clusterIndex,
                           [int messageOn]
                           )
```

ARGUMENTS

clusterIndex
The cluster index. Zero based value, the first cluster has index 0.

messageOn (optional)
Possible values:

0	Do not emit a message in the output window.
1	Emit a message in the output window in case that the function does not execute properly.

RETURNS

0	On success.
1	There are no clusters, the number of clusters is 0.
2	The cluster index exceeds the number of clusters.
3	There is no load flow in memory before running CalcCluster.

GetNumberOfClusters

Get the number of clusters.

```
int SetCluster.GetNumberOfClusters()
```

RETURNS

The number of clusters.

4.5.2 SetColscheme

Overview

CreateFilter
SetColouring
SetFilter

CreateFilter

Creates filter used to determine objects to be colored.

```
int SetColscheme.CreateFilter([int pageNr])
```

ARGUMENTS

pageNr	**empty**	Create filter for currently valid calculation
	set	Dialog page number for which filter is created (see table below)

Table 4.5.3

Dialog Page Name	**”pageNr” value**
Basic Data	101
Load Flow	102
AC Load Flow Sensitivities	120
AC Contingency Analysis	121
AC Quasi-dynamic Simulation	137
DC Load Flow	122
DC Load Flow Sensitivities	123
DC Contingenciy Analysis	124
DC Quasi-dynamic Simulation	138
VDE/IEC Short-Circuit	103
Complete Short-Circuit	111
ANSI Short-Circuit	112
IEC 61363	114
DC Short-Circuit	117
RMS-Simulation	104
Modal Analysis	128
EMT-Simulation	105
Harmonics/Power Quality	106
Frequency Sweep	127
D-A-CH-CZ Connection Request	139
BDEW/VDE Connection Request	142
Optimal Power Flow	108
DC Optimal Power Flow	130
DC OPF with Contingencies	135
State Estimation	113
Reliability	109
General Adequacy	115
Tie Open Point Opt.	116
Motor Starting Calculation	133
Arc Flash Calculation	129
Optimal Capacitor Placement	126
Voltage Profile Optimisation	125
Backbone Calculation	131
Optimal RCS Placement	132
Optimal Manual Restoration	136
Phase Balance Optimisation	141
User defined calculation	142

RETURNS

0	On success.
1	On error.

SetColouring

Sets colouring for given or currently valid calculation.

```
int SetColscheme.SetColouring(str page,
                              int energizing,
                              [int alarm,]
                              [int normal]
                              )
```

ARGUMENTS

page

empty	set for currentlx valid calculation
set	page for which modes are set (see table below)

energizing
Colouring for Energizing Status

-2	enable (set to previously selected mode),
-1	do not change
0	disable
>**0**	set to this mode (see table below)

alarm Colouring for Alarm

-2	enable (set to previously selected mode),
-1	do not change (default)
0	disable
>**0**	set to this mode (see table below)

normal Other Colouring

-2	enable (set to previously selected mode),
-1	do not change (default)
0	disable
>**0**	set to this mode (see table below)

Table 4.5.4

Dialog Page Name	**”page” value**
Basic Data	basic
Load Flow	ldf
AC Load Flow Sensitivities	acsens
AC Contingency Analysis	accont
AC Quasi-dynamic Simulation	acldfsweep
DC Load Flow	dcldf
DC Load Flow Sensitivities	dcsens
DC Contingenciy Analysis	dccont
DC Quasi-dynamic Simulation	dcldfsweep
VDE/IEC Short-Circuit	shc
Complete Short-Circuit	shcfull
ANSI Short-Circuit	shcansi
IEC 61363	shc61363
DC Short-Circuit	shcdc
RMS-Simulation	rms
Modal Analysis	modal
EMT-Simulation	emt
Harmonics/Power Quality	harm
Frequency Sweep	fsweep
D-A-CH-CZ Connection Request	dachcz
BDEW/VDE Connection Request	bdewvde
Optimal Power Flow	opf
DC Optimal Power Flow	dcopf
DC OPF with Contingencies	dccontopf
State Estimation	est
Reliability	rel
General Adequacy	genrel
Tie Open Point Opt.	topo
Motor Starting Calculation	motstart
Arc Flash Calculation	arcflash
Optimal Capacitor Placement	optcapo
Voltage Profile Optimisation	mvplan
Backbone Calculation	backbone
Optimal RCS Placement	optrcs
Optimal Manual Restoration	omr
Phase Balance Optimisation	balance
User defined calculation	usercalc

Table 4.5.5

Energizing State Name	**”energizing” value**
De-energized	33
Out of Calculation	37
De-energised, Planned Outage	61

Table 4.5.6

Alarm Name	**”alarm” value**
Voltage Violations / Overloading	29
Outages	31
Overloading of Thermal / Peak Short Circuit Current	32
Feeder Radiality Check	38

Table 4.5.7

Other Colouring Name	**Group**	**”normal” value**
Voltages / Loading	Results	1
Voltage Levels	Topology	2
Individual	Individual	4
Connected Grid Components	Topology	5
According to Filter	User-defined	see notes below table
Grids	Groupings	7
Modifications in Variations / System Stages	Variations / System Stages	8
Loading of Thermal / Peak Short-Circuit Current	Results	9
Paths	Groupings	10
System Type AC/DC and Phases	Topology	11
Relays, Current and Voltage Transformers	Secondary Equipment	12
Fault Clearing Times	Results	13
Feeders	Topology	14
Switches, Type of Usage	Secondary Equipment	15
Measurement Locations	Secondary Equipment	16
Missing graphical connections	Topology	17
Zones	Groupings	18
State Estimation	Results	19
Boundaries (Interior Region)	Topology	20
Station Connectivity	Topology	21
Outage Check	Topology	22
Energizing Status	Topology	23
Modifications in Recording Expansion Stage	Variations / System Stages	24
Areas	Groupings	25
Owners	Groupings	26
Routes	Groupings	27
Operators	Groupings	28
Original Locations	Variations / System Stages	30
Boundaries (Definition)	Topology	34
Meteo Stations	Groupings	35
Station Connectivity (Beach Balls only)	Topology	36
Power Restoration	Secondary Equipment	43
Connected Components	Topology	39
Connected Components, Voltage Level	Topology	40
Year of Construction	Primary Equipment	41
Cross Section	Primary Equipment	42
Forced Outage Rate	Primary Equipment	44
Forced Outage Duration	Primary Equipment	45
Loads: Yearly interruption frequency	Results	46
Loads: Yearly interruption time	Results	47
Loads: Average Interruption Duration	Results	48
Loads: Load Point Energy Not Supplied	Results	49
Supplied by Substation	Topology	50
Supplied by Secondary Substation	Topology	51
Incident Energy	Results	52
PPE-Category	Results	53
Optimal Manual Restoration	Results	54
Connection Request: Approval Status	Results	55
Voltage Angle	Results	56
Contributions to SAIDI	Results	57
Contributions to SAIFI	Results	58
Contributions to ENS	Results	59
Contributions to EIC	Results	60

Note: User-defined filters can be set with a “normal” value of 1000 or higher. The first filter in the list has the value 1000, the next one has 1001 and so on.

RETURNS

0	error (at least one of the given colourings cannot be set, e.g. not available for given page). Nothing is changed.
1	ok

SetFilter

Sets filter for given or currently valid calculation.

```
int SetColscheme.SetFilter(int filter,
                           [int page]
                           )
int SetColscheme.SetFilter(DataObject obj,
                           [int page]
                           )
```

ARGUMENTS

filter number of filter to be set

obj user-defined filter to be set

page (optional)
Dialog page number for which filter is set (for numbers see table listed in SetColscheme.CreateFilter())

RETURNS

0	ok
1	error (filter or page not found)

SEE ALSO

SetColscheme.CreateFilter()

4.5.3 SetDesktop

Overview

AddPage
DoAutoScaleX
GetPage
SetAdaptX
SetAutoScaleX
SetResults
SetScaleX
SetXVar
Show
WriteWMF

AddPage

Adds an existing page to a graphics and activates it

- Opens the graphics board if not already open.
- Adds the page if it is not already part of the graphics board.

```
DataObject SetDesktop.AddPage(DataObject page2add)
```

ARGUMENTS

page2add
The page to add to the desktop.

- Page is a SetVipage (virtual instrument panel): A copy of the page is added.
- Page is an IntGrfnet (Single line graphic, block diagram): The graphic is added.

RETURNS

The page displayed or None if the desktop was not changed.

DoAutoScaleX

Scales the x-axes of all plots in the graphics board which use the x-axis scale defined in the graphics board.

```
None SetDesktop.DoAutoScaleX()
```

GetPage

Searches, activates and returns a graphics page in the currently open graphics board. If “create” is true, then a new virtual instrument panel will be created and added to the graphics board if no page with name was found.

```
DataObject SetDesktop.GetPage(str name,
                              [int create]
                              )
```

ARGUMENTS

name Name of the page.

create (optional)
Possible values:

0	do not create new virtual instrument panel
1	create panel if it does not exist already

RETURNS

Virtual instrument panel (SetVipage)

SetAdaptX

Sets the Adapt Scale option of the x-scale.

```
None SetDesktop.SetAdaptX(int mode,
                          [float trigger]
                          )
```

ARGUMENTS

mode Possible values:

0	off
1	on

trigger (optional)
Trigger value, unused if mode is off or empty

SetAutoScaleX

Sets automatic scaling mode of the x-scale. A warning is issued if an invalid mode is passed to the function.

```
None SetDesktop.SetAutoScaleX(int mode)
```

ARGUMENTS

mode Possible values:

0	never
1	after simulation
2	during simulation

SetResults

Sets default results object of graphics board.

```
None SetDesktop.SetResults(DataObject res)
```

ARGUMENTS

res Result object to set or None to reset. Valid result object is any of class ElmRes, IntComtrade and IntComtradeset.

SetScaleX

Sets x-axis scale. A function call without arguments sets the Auto Scale setting to On without changing the scale itself.

```
None SetDesktop.SetScaleX()
None SetDesktop.SetScaleX(float min,
                          float max,
                          [int log]
                          )
```

ARGUMENTS

min (optional)
Minimum of x-scale.

max (optional)
Maximum of x-scale.

log (optional)
Possible values:

0	linear
1	logarithmic

SetXVar

Sets x-axis variable. If The default x-axis variable (time) is set if no argument is passed.

```
None SetDesktop.SetXVar()
None SetDesktop.SetXVar(DataObject obj,]
                        str varname
                        )
```

ARGUMENTS

obj (optional)
x-axis object

varname (optional)
variable of obj

Show

Shows the virtual instrument panel with the same name as 'pageObject' or the page with name 'pageName' in the graphics board. The object 'pageObject' is typically a object of class 'SetVipage' (virtual instrument panel) but, as only its name is used, it may be any other type of object. Calling the function without an argument opens the graphics board.

```
int SetDesktop.Show()
int SetDesktop.Show(str pageName)
int SetDesktop.Show(DataObject pageObject)
```

ARGUMENTS

pageName (optional)
Name of graphics page.

pageObject (optional)
A graphics page oject.

RETURNS

0	on success
1	on error

WriteWMF

Writes the currently open graphic to a windows metafile file (*.wmf).

```
int SetDesktop.WriteWMF(str filename)
```

ARGUMENTS

filename Filename without extension.

4.5.4 SetDistrstate

Overview

CalcCluster

CalcCluster

Calculates a load flow with a given load distribution state applied.

```
list(int, ...) SetDistrstate.CalcCluster(float arg0,
                                         [float arg1]
                                         )
```

ARGUMENTS

clusterIndex
The number of the load cluster - 1

emitMessage (optional)
Emit messages if not equal to zero

RETURNS

0 if ok. -1 if load flow of cluster did not converge.

4.5.5 SetFilt

Overview

Get

Get

Returns a container with the filtered objects.

```
list SetFilt.Get()
```

RETURNS

The set of filtered objects.

4.5.6 SetLevelvis

Overview

AdaptWidth

This function resizes the in the object specified group of text boxes regarding their text contents.

```
None SetLevelvis.AdaptWidth()
```

Align

This function aligns the text within a text box.

```
None SetLevelvis.Align(int iPos)
```

ARGUMENTS

iPos Alignment position

0	left
1	middle
2	right

ChangeFont

This function sets the font number for the specified group of text boxes.

```
None SetLevelvis.ChangeFont(int iFont)
```

ARGUMENTS

iFont Font number (default fonts range from 0 to 13)

ChangeFrameAndWidth

This method is not available anymore. Please use SetLevelvis.ChangeWidthVisibilityAndColour() instead.

```
list(None, ...) SetLevelvis.ChangeFrameAndWidth([int iFrame,]
                                                [int iWidth,]
                                                [int iVisibility,]
                                                [int iColour]
                                                )
```

ChangeLayer

This function sets the specified group of text boxes to a given layer.

```
None SetLevelvis.ChangeLayer(str sLayer)
```

ARGUMENTS

sLayer	Layer name (e.g. 'Object Names', 'Results', 'Invisible Objects',..)

ChangeRefPoints

This function sets the reference points between a text box (second parameter) and its parent object (first parameter), e.g. if the result box of a busbar shall be shown on top of a drawn bar instead of below the bar the values change from (6,4) to (4,6). The first number specifies the reference number of the text box. The integer values describe the position of the reference points within a rectangle (0=centre, 1=middle right, 2=top right,..):
4 3 2
5 0 1
6 7 8

```
None SetLevelvis.ChangeRefPoints(int iParRef,
                                 int iTBRef
                                 )
```

ARGUMENTS

iParRef	Defines the reference point on the parent object (e.g. busbar)
iTBRef	Defines the reference point on the text box

ChangeWidthVisibilityAndColour

This function sets the visibility of the frame, the width (in number of letters), the visibility and the colour of text boxes.

```
None SetLevelvis.ChangeWidthVisibilityAndColour([int iWidth,]
                                                [int iVisibility,]
                                                [int iColour]
                                                )
```

ARGUMENTS

iWidth Sets the width in number of letters

0..n	width

iVisibility Sets the visibility

0	not visible
1	visible

iColour Sets the colour

0..255	colour

Mark

Marks the specified group of text boxes in the currently shown diagram.

```
None SetLevelvis.Mark()
```

Reset

This function resets the individually modified text box settings.

```
None SetLevelvis.Reset(int iMode)
```

ARGUMENTS

iMode

0	Reset to default (changed reference points are not reset)
1	Only font
2	Shift to original layer (result boxes to layer 'Results', object names to layer 'Object Names')

4.5.7 SetParalman

Overview

GetNumSlave
SetNumSlave
SetTransfType

GetNumSlave

To get the number of slaves which is currently configured.

```
int SetParalman.GetNumSlave()
```

RETURNS

the number of slaves which is currently configured.

SetNumSlave

To configue the number of slaves to be used for parallel computing.

```
int SetParalman.SetNumSlave(int numSlaves)
```

ARGUMENTS

numSlaves
Number of slaves to be used for parallel computing

-1	All cores available will be used.
> 0	The number of slaves to be used.

RETURNS

Always return 0.

SetTransfType

To change the data transfer type: via file or via socket communication.

```
int SetParalman.SetTransfType(int viaFile)
```

ARGUMENTS

viaFile

0	The data will be transferred via socket communication.
1	The data will be transferred via file.

RETURNS

0	the data will be transferred via socket communication.
1	the data will be transferred via file.

4.5.8 SetSelect

Overview

AddRef
All
AllAsm
AllBars
AllBreakers
AllClosedBreakers
AllElm
AllLines
AllLoads
AllOpenBreakers
AllSym
AllTypLne
Clear
GetAll

AddRef

Adds a reference to the objects to the existing selection.

```
None SetSelect.AddRef(DataObject O)
None SetSelect.AddRef(list S)
```

ARGUMENTS

O	An object.
S	A set of objects.

All

Returns all objects in the selection.

```
list SetSelect.All()
```

RETURNS

The set of objects

AllAsm

Returns all asynchronous machines in the selection.

```
list SetSelect.AllAsm()
```

RETURNS

The set of objects

AllBars

Returns all busbars and terminals in the selection.

```
list SetSelect.AllBars()
```

RETURNS

The set of objects

AllBreakers

Returns all breakers in the selection.

```
list SetSelect.AllBreakers()
```

RETURNS

The set of objects

AllClosedBreakers

Returns all closed breakers in the selection.

```
list SetSelect.AllClosedBreakers()
```

RETURNS

The set of objects

AllElm

Returns all elements (Elm*) in the selection.

```
list SetSelect.AllElm()
```

RETURNS

The set of containing objects

AllLines

Returns all lines and line routes in the selection.

```
list SetSelect.AllLines()
```

RETURNS

The set of objects

AllLoads

Returns all loads in the selection.

```
list SetSelect.AllLoads()
```

RETURNS

The set of objects

AllOpenBreakers

Returns all open breakers in the selection.

```
list SetSelect.AllOpenBreakers()
```

RETURNS

The set of objects

AllSym

Returns all synchronous machines in the selection.

```
list SetSelect.AllSym()
```

RETURNS

The set of objects

AllTypLne

Returns all line types in the selection.

```
list SetSelect.AllTypLne()
```

RETURNS

The set of objects

Clear

Clears (deletes) the selection.

```
None SetSelect.Clear()
```

GetAll

Returns all objects in the selection which are of the class 'ClassName'.

```
list SetSelect.GetAll(str ClassName)
```

ARGUMENTS

ClassName
The object class name.

RETURNS

The set of objects

4.5.9 SetTboxconfig

Overview

Check
GetAvailableButtons
GetDisplayedButtons
Purge
SetDisplayedButtons

Check

Checks buttons to be displayed for invalid or duplicate ids and prints error messages.

```
int SetTboxconfig.Check()
```

RETURNS

0	No errors found.
1	Errors found.

GetAvailableButtons

Gets buttons available for selected tool bar.

```
str SetTboxconfig.GetAvailableButtons()
```

RETURNS

String ids of all buttons available for selected tool bar; ids are separated by '\n'.

GetDisplayedButtons

Gets buttons configured to be displayed in selected tool bar.

```
str SetTboxconfig.GetDisplayedButtons()
```

RETURNS

String ids of all buttons configured to be displayed in selected tool bar; ids are separated by '\n'.

Purge

Purges buttons to be displayed from invalid or duplicate ids.

```
int SetTboxconfig.Purge()
```

RETURNS

0	No problems found.
1	Configuration was adapted.

SetDisplayedButtons

Sets buttons to be displayed in selected tool bar. Purges given buttons from invalid or duplicate buttons (duplicate separators or breaks are kept).

```
int SetTboxconfig.SetDisplayedButtons(str buttonIds)
```

ARGUMENTS

buttonIds String ids of all buttons to be set as displayed buttons; ids have to be separated by '\n'

RETURNS

0	Given buttons were stored without modification.
1	Given buttons were purged from invalid or duplicate ids.

4.5.10 SetTime

Overview

Date
SetTime
SetTimeUTC
Time

Date

Sets date component to current system date.

```
None SetTime.Date()
```

SEE ALSO

SetTime.Time(), SetTime.SetTimeUTC()

SetTime

Sets the time in the current year. There is no restriction to the values for H, M and S, except for the fact that negative values are interpreted as zero. Values higher than 24 or 60 will be processed normally by adding the hours, minutes and seconds into an absolute time, from which a new hour-of-year, hour-of-day, minutes and seconds are calculated.

```
None SetTime.SetTime(float H,
                     [float M,]
                     [float S]
                     )
```

Arguments

H The hours

M (optional)
The minutes

S (optional)
The seconds

SetTimeUTC

Sets date and time to given time. The time must be given in UTC format as seconds since 01.01.1970 00:00 GMT.

```
None SetTime.SetTimeUTC(int time)
```

Arguments

time UTC time in seconds since 01.01.1970 00:00 GMT

See also

SetTime.Date(), SetTime.Time()

Time

Sets time component to current system time.

```
None SetTime.Time()
```

See also

SetTime.Date(), SetTime.SetTimeUTC()

4.5.11 SetVipage

Overview

CreateVI
DoAutoScaleX
DoAutoScaleY
GetVI
SetAdaptX
SetAutoScaleX
SetResults
SetScaleX
SetStyle
SetTile
SetXVar

CreateVI

Creates a copy of the virtual instrument passed and displays the copy on this panel.

```
DataObject SetVipage.CreateVI(DataObject vi)
```

ARGUMENTS

vi	The virtual instrument which will be copied. Only virtual instruments are allowed (classname 'Vis*').

RETURNS

Returns the created virtual instrument.

DoAutoScaleX

Scales the x-axes of all plots on the virtual instrument panel automatically.

```
None SetVipage.DoAutoScaleX()
```

DoAutoScaleY

Scales the y-axes of all plots on the virtual instrument panel automatically.

```
None SetVipage.DoAutoScaleY()
```

GetVI

Get or create a virtual instruments of the virtual instrument panel.

```
DataObject SetVipage.GetVI (str name,
                            [str class,]
                            [int create]
                            )
```

ARGUMENTS

name	Name of virtual instrument

class='VisPlot' (optional)
classname of virtual instrument.

create (optional)
Possible values:

0	do not create new virtual instrument
1	create virtual instrument if it does not exist already

RETURNS

Virtual instrument

SetAdaptX

Sets the Adapt Scale option of the x-scale.

```
None SetVipage.SetAdaptX(int mode,
                         [float trigger]
                         )
```

ARGUMENTS

mode Possible values:

0	off
1	on

trigger (optional)
Trigger value, unused if mode is off or empty

SetAutoScaleX

Sets automatic scaling mode of the x-scale. A warning is issued if an invalid mode is passed to the function.

```
None SetVipage.SetAutoScaleX(int mode)
```

ARGUMENTS

mode Possible values:

0	never
1	after simulation
2	during simulation

SetResults

Sets default results object of virtual instrument panel.

```
None SetVipage.SetResults(DataObject res)
```

ARGUMENTS

res	Result object to set or None to reset. Valid result object is any of class ElmRes, IntComtrade and IntComtradeset.

SetScaleX

Sets x-axis scale. A function call without arguments sets the Auto Scale setting to On without changing the scale itself.

```
None SetVipage.SetScaleX()
None SetVipage.SetScaleX(float min,
                         float max,
                         [int log]
                         )
```

ARGUMENTS

min (optional)
Minimum of x-scale.

max (optional)
Maximum of x-scale.

log (optional)
Possible values:

0	linear
1	logarithmic

SetStyle

Sets style of virtual instrument panel. A warning message is issued in the case that a style with the given name does not exist.

```
None SetVipage.SetStyle(str name)
```

ARGUMENTS

name	Style Name

SetTile

Rearranges the virtual instrument on the panel.

```
None SetVipage.SetTile([int tile])
```

ARGUMENTS

tile=1 (optional)

tile =0	arrange virtual instruments automatically (like tiles)
tile=1	arrange them horizontally

SetXVar

Sets x-axis variable. If The default x-axis variable (time) is set if no argument is passed.

```
None SetVipage.SetXVar()
None SetVipage.SetXVar(DataObject obj,]
                        str varname
                        )
```

ARGUMENTS

obj (optional)
: x-axis object

varname (optional)
: variable of obj

4.6 Others

4.6.1 BlkDef

Overview

Compile
Encrypt
GetCheckSum
Pack
PackAsMacro

Compile

Compiles the model to a DLL. Can be called on an already compiled model.

```
int BlkDef.Compile([string modelPath])
```

ARGUMENTS

modelPath (optional)
: Full path to a location where the model should be stored. Leave empty to use default.

Encrypt

Encrypts this block definition. It has to be packed as macro before.

```
int BlkDef.Encrypt([int doRemoveHistoricRecords])
```

ARGUMENTS

doRemoveHistoricRecords (optional)
: 0: do not remove historic copies in database. 1: do remove 2: show dialog and ask.

RETURNS

0	On success.
1	On error.

SEE ALSO

BlkDef.PackAsMacro()

GetCheckSum

```
str BlkDef.GetCheckSum()
```

DEPRECATED NAMES

CalculateCheckSum

RETURNS

The checksum of the block definition (0000-0000-0000-0000 for frames).

Pack

Copies all used macros (i.e. referenced BlkDef) to this block.

```
int BlkDef.Pack()
```

RETURNS

0	On success.
1	On error.

PackAsMacro

Collects all equations, stores them to this model and deletes block diagram and all macro references.

```
int BlkDef.PackAsMacro()
```

RETURNS

0	On success.
1	On error.

SEE ALSO

BlkDef.Encrypt()

4.6.2 BlkSig

Overview

GetFromSigName
GetToSigName

GetFromSigName

```
str BlkSig.GetFromSigName()
```

RETURNS

The name of the output from which the signal is connected. In cases of no connection, an empty string.

GetToSigName

```
str BlkSig.GetToSigName()
```

RETURNS

The name of the input to which the signal is connected. In cases of no connection, an empty string.

4.6.3 ChaVecfile

Overview

Update

Update

Reloads the file from disk. Same behaviour like button update.

```
int ChaVecfile.Update([int msgOn = 0])
```

ARGUMENTS

msgOn (optional)
Reporting of errors:

0	No error message is shown in case that the file can not be loaded (default).
1	Emit an error message in case that the file can not be loaded.

RETURNS

The number of samples (rows) read from the file.

4.6.4 CimModel

Overview

DeleteParameterAtIndex
GetAttributeEnumerationType
GetParameterCount
GetParameterNamespace
GetParameterValue
HasParameter
RemoveParameter
SetAssociationValue

SetAssociationValue
SetAttributeEnumeration
SetAttributeEnumeration
SetAttributeValue
SetAttributeValue

DeleteParameterAtIndex

Removes the parameter (attribute, or association) value at the given index.

```
None CimModel.DeleteParameterAtIndex(str parameter, int index)
```

ARGUMENTS

parameter
Full-name specifier of the attribute, or association (e.g. ”Model.profile”)

index Index of the parameter

GetAttributeEnumerationType

Returns the enumeration type of the attribute.

```
str CimModel.GetAttributeEnumerationType(str attribute)
```

ARGUMENTS

attribute Full-name specifier of the attribute (e.g. ”GeneratingUnit.genControlSource”)

GetParameterCount

Returns the number of parameters (attribute, or association) of given type.

```
int CimModel.GetParameterCount(str parameter)
```

ARGUMENTS

parameter
Full-name specifier of the attribute, or association (e.g. ”Model.profile”)

GetParameterNamespace

Returns the namesace of the parameter (attribute, or association).

```
str CimModel.GetParameterNamespace(str parameter)
```

ARGUMENTS

parameter
Full-name specifier of the attribute, or association (e.g. ”Model.profile”)

GetParameterValue

Returns the value of the parameter (attribute, or association) at the given index if available. If the parameter (attribute, or association) is not available, or the index is out of bounds the function returns an empty string.

```
str CimModel.GetParameterValue(str parameter, [int index])
```

Arguments

parameter
Full-name specifier of the attribute, or association (e.g. "Model.modelingAuthoritySet")

index Index of the parameter:

0	Default index

HasParameter

Checks whether the CimModel has the parameter (attribute, or association) specified.

```
int CimModel.HasParameter(str parameter)
```

Arguments

parameter
Full-name specifier of the attribute, or association (e.g. "Model.modelingAuthoritySet")

Returns

1	if parameter is specified
0	if parameter is not specified

RemoveParameter

Removes all occurences of the parameter (attribute, or association).

```
None CimModel.RemoveParameter(str parameter)
```

Arguments

parameter
Full-name specifier of the attribute, or association (e.g. "Model.modelingAuthoritySet")

SetAssociationValue

Adds the association if not available yet, and sets its value at the given index. If the association is already added, the function sets a new value at the given index only.

```
None CimModel.SetAssociationValue(str association,
                                  str value,
                                  [int index])
```

ARGUMENTS

association
Full-name specifier of the association (e.g. ”Model.DependentOn”)

value Value of the association

index Index of the association:

0 Default index

SetAssociationValue

Adds the association if not available yet, and sets its namespace and value. If the association is already added, the function sets its namespace and value only.

```
None CimModel.SetAssociationValue(str association,
                                  str value,
                                  str nspace)
```

ARGUMENTS

attribute Full-name specifier of the association (e.g. ”Model.DependentOn”)

value Value of the association

nspace Namespace of the association (e.g. ”md”)

SetAttributeEnumeration

Adds the attribute if not available yet, and sets its enumeration type and value. If the attribute is already added, the function sets its enumeration type and value only.

```
None CimModel.SetAttributeEnumeration(str attribute,
                                      str enumerationType,
                                      str value)
```

ARGUMENTS

attribute Full-name specifier of the attribute (e.g. ”GeneratingUnit.genControlSource”)

enumerationType
Enumeration type of the attribute (e.g. ”GeneratorControlSource”)

value Value of the enumeration (e.g. ”offAGC”)

SetAttributeEnumeration

Adds the attribute if not available yet, and sets its namespace, enumeration type and value. If the attribute is already added, the function sets its namespace, enumeration type and value only.

```
None CimModel.SetAttributeEnumeration(str attribute,
                                      str enumerationType,
                                      str value,
                                      str nspace)
```

ARGUMENTS

attribute	Full-name specifier of the attribute (e.g. "GeneratingUnit.genControlSource")
enumerationType	Enumeration type of the attribute (e.g. "GeneratorControlSource")
value	Value of the attribute (e.g. "offAGC")
nspace	Namespace of the attribute (e.g. "cim")

SetAttributeValue

Adds the attribute if not available yet, and sets its value at the given index. If the attribute is already added, the function sets a new value at the given index only.

```
None CimModel.SetAttributeValue(str attribute,
                                str value,
                                [int index])
```

ARGUMENTS

attribute	Full-name specifier of the attribute (e.g. "Model.modelingAuthoritySet")
value	Value of the attribute
index	Index of the attribute:

0	Default index

SetAttributeValue

Adds the attribute if not available yet, and sets its namespace and value. If the attribute is already added, the function sets its namespace and value only.

```
None CimModel.SetAttributeValue(str attribute,
                                str value,
                                str nspace)
```

ARGUMENTS

attribute	Full-name specifier of the attribute (e.g. "Model.modelingAuthoritySet")
value	Value of the attribute
nspace	Namespace of the attribute (e.g. "md")

4.6.5 CimObject

Overview

DeleteParameterAtIndex

Removes the parameter (attribute, or association) value at the given index.

```
None CimObject.DeleteParameterAtIndex(str parameter, int index)
```

ARGUMENTS

parameter
Full-name specifier of the attribute, or association (e.g. ”Model.profile”)

index Index of the parameter

GetAttributeEnumerationType

Returns the enumeration type of the attribute.

```
str CimObject.GetAttributeEnumerationType(str attribute)
```

ARGUMENTS

attribute Full-name specifier of the attribute (e.g. ”GeneratingUnit.genControlSource”)

GetParameterCount

Returns the number of parameters (attribute, or association) of given type.

```
int CimObject.GetParameterCount(str parameter)
```

ARGUMENTS

parameter
Full-name specifier of the attribute, or association (e.g. ”Model.profile”)

GetParameterNamespace

Returns the namesace of the parameter (attribute, or association).

```
str CimObject.GetParameterNamespace(str parameter)
```

ARGUMENTS

parameter
Full-name specifier of the attribute, or association (e.g. ”IdentifiedObject.name”)

GetParameterValue

Returns the value of the parameter (attribute, or association) at the given index if available. If the parameter (attribute, or association) is not available, or the index is out of bounds the function returns an empty string.

```
str CimObject.GetParameterValue(str parameter, [int index])
```

ARGUMENTS

parameter
Full-name specifier of the attribute, or association (e.g. ”IdentifiedObject.name”)

index Index of the parameter:

0	Default index

HasParameter

Checks whether the CimObject has the parameter (attribute, or association) specified.

```
int CimObject.HasParameter(str parameter)
```

ARGUMENTS

parameter
Full-name specifier of the attribute, or association (e.g. ”IdentifiedObject.name”)

RETURNS

1	if parameter is specified
0	if parameter is not specified

RemoveParameter

Removes all occurences of the parameter (attribute, or association).

```
None CimObject.RemoveParameter(str parameter)
```

ARGUMENTS

parameter
Full-name specifier of the attribute, or association (e.g. ”IdentifiedObject.name”)

SetAssociationValue

Adds the association if not available yet, and sets its value at the given index. If the association is already added, the function sets a new value at the given index only.

```
None CimObject.SetAssociationValue(str association,
                                   str value,
                                   [int index])
```

ARGUMENTS

association
Full-name specifier of the association (e.g. ”Equipment.EquipmentContainer”)

value Value of the association

index Index of the association:

0 Default index

SetAssociationValue

Adds the association if not available yet, and sets its namespace and value. If the association is already added, the function sets its namespace and value only.

```
None CimObject.SetAssociationValue(str association,
                                   str value,
                                   str nspace)
```

ARGUMENTS

attribute Full-name specifier of the association (e.g. ”Equipment.EquipmentContainer”)

value Value of the association

nspace Namespace of the association (e.g. ”cim”)

SetAttributeEnumeration

Adds the attribute if not available yet, and sets its enumeration type and value. If the attribute is already added, the function sets its enumeration type and value only.

```
None CimObject.SetAttributeEnumeration(str attribute,
                                       str enumerationType,
                                       str value)
```

ARGUMENTS

attribute Full-name specifier of the attribute (e.g. ”GeneratingUnit.genControlSource”)

enumerationType
Enumeration type of the attribute (e.g. ”GeneratorControlSource”)

value Value of the enumeration (e.g. ”offAGC”)

SetAttributeEnumeration

Adds the attribute if not available yet, and sets its namespace, enumeration type and value. If the attribute is already added, the function sets its namespace, enumeration type and value only.

```
None CimObject.SetAttributeEnumeration(str attribute,
                                       str enumerationType,
                                       str value,
                                       str nspace)
```

ARGUMENTS

attribute Full-name specifier of the attribute (e.g. ”GeneratingUnit.genControlSource”)

enumerationType
Enumeration type of the attribute (e.g. ”GeneratorControlSource”)

value Value of the attribute (e.g. ”offAGC”)

nspace Namespace of the attribute (e.g. ”cim”)

SetAttributeValue

Adds the attribute if not available yet, and sets its value at the given index. If the attribute is already added, the function sets a new value at the given index only.

```
None CimObject.SetAttributeValue(str attribute,
                                 str value,
                                 [int index])
```

ARGUMENTS

attribute Full-name specifier of the attribute (e.g. ”IdentifiedObject.name”)

value Value of the attribute

index Index of the attribute:

0 Default index

SetAttributeValue

Adds the attribute if not available yet, and sets its namespace and value. If the attribute is already added, the function sets its namespace and value only.

```
None CimObject.SetAttributeValue(str attribute,
                                 str value,
                                 str nspace)
```

ARGUMENTS

attribute Full-name specifier of the attribute (e.g. ”IdentifiedObject.name”)

value Value of the attribute

nspace Namespace of the attribute (e.g. ”cim”)

4.6.6 IntCase

Overview

Activate
ApplyNetworkState
ApplyStudyTime
Consolidate
Deactivate
SetStudyTime

Activate

Activates the study case. Deactivates other study cases first.

```
int IntCase.Activate()
```

RETURNS

0	on success
1	on error

ApplyNetworkState

For a study case in a combined project, copy the network state from another case.

Copies the active grids, scenarios and network variations configuration to the current case. The data will be added to any already existing configuration.

```
int IntCase.ApplyNetworkState(IntCase other)
```

ARGUMENTS

IntCase	The source Study Case to copy data from

RETURNS

0	On success
1	Source object is not an IntCase object
2	Case where function is called on is not the active case
3	Source case is not from active project
4	Source Study Case is not from a source project in a combined project
5	Other error. Details are given in an error message

ApplyStudyTime

For a study case in a combined project, apply the study time from another study case.

```
int IntCase.ApplyStudyTime(IntCase other)
```

ARGUMENTS

IntCase	The source study case to copy study time from

RETURNS

0	On success
1	Source object is not an IntCase object
2	Study case where function is called on is not the active case
3	Source case is not from active project
4	Source case is not from a project part of a combined project

Consolidate

Changes that are recorded in a project's active Variations are permanently applied to the Network Data folder (like right mouse button Consolidate Network Variation)
Note: Modified scenarios are not saved!

Works only:

- For active study cases
- If a network variation is active

```
int IntCase.Consolidate()
```

RETURNS

0	On success
1	If an error has occured

SEE ALSO

IntScheme.Consolidate()

Deactivate

De-activates the study case.

```
int IntCase.Deactivate()
```

RETURNS

0	on success
1	on error

SetStudyTime

Sets the Study Case time to seconds since 01.01.1970 00:00:00.

```
None IntCase.SetStudyTime(float dateTime)
```

ARGUMENTS

dateTime	Seconds since 01.01.1970 00:00:00

4.6.7 IntComtrade

Overview

ConvertToASCIIFormat

Creates new comtrade configuration and data files in ASCII format in the file system directory of the original files. The new configuration file is linked automatically to a new IntComtrade object created in the same *PowerFactory* folder like this object. An existing IntComtrade object is already in ASCII format when its parameter 'Binary' is set to 0.

```
int IntComtrade.ConvertToASCIIFormat()
```

RETURNS

0	File successfully converted.
1	Error occured, e.g. file is already in ASCII format.

ConvertToBinaryFormat

Creates new comtrade configuration and data files in binary format in the file system directory of the original files. The new configuration file is linked automatically to a new IntComtrade object created in the same *PowerFactory* folder like this object. An existing IntComtrade object is already in binary format when its parameter 'Binary' is set to 1.

```
int IntComtrade.ConvertToBinaryFormat()
```

RETURNS

0	File successfully converted.
1	Error occured, e.g. file is already in binary format.

FindColumn

Returns the first column matching the variable name.

```
int IntComtrade.FindColumn(str variable,
                           [int startCol]
                           )
```

ARGUMENTS

variable The variable name to look for.

startCol (optional)
The index of the column at which to start the search.

RETURNS

≥ 0	The column index found.
< 0	The column with name variable was not found.

FindMaxInColumn

Find the maximum value of the variable in the given column.

```
[int row,
float value] IntComtrade.FindMaxInColumn(int column)
```

ARGUMENTS

column The column index.

value (optional, out)
The maximum value found. The value is 0. in case that the maximum value was not found.

RETURNS

< 0	The maximum value of column was not found.
≥ 0	The row with the maximum value of the column.

FindMinInColumn

Find the minimum value of the variable in the given column.

```
[int row,
float value] IntComtrade.FindMinInColumn(int column)
```

ARGUMENTS

column The column index.

value (optional, out)
The minimum value found. The value is 0. in case that the minimum value was not found.

RETURNS

< 0	The minimum value of column was not found.
≥ 0	The row with the minimum value of the column.

GetDescription

Get the description of a column.

```
str IntComtrade.GetDescription([int column],
                               [int short]
                               )
```

ARGUMENTS

column (optional)
: The column index. The description name of the default variable is returned if the parameter is nor passed to the function.

short (optional)

0	long desc. (default)
1	short description

RETURNS

Returns the description which is empty in case that the column index is not part of the data.

GetNumberOfColumns

Returns the number of variables (columns) in result file excluding the default variable (e.g. time for time domain simulation).

```
int IntComtrade.GetNumberOfColumns()
```

RETURNS

Number of variables (columns) in result file.

GetNumberOfRows

Returns the number of values per column (rows) stored in result object.

```
int IntComtrade.GetNumberOfRows()
```

RETURNS

Returns the number of values per column stored in result object.

GetUnit

Get the unit of a column.

```
str IntComtrade.GetUnit([int column])
```

ARGUMENTS

column (optional)
: The column index. The unit of the default variable is returned if the parameter is nor passed to the function.

RETURNS

Returns the unit which is empty in case that the column index is not part of the data.

GetValue

Returns a value from a result object for row iX of curve col.

```
[int error,
float d   ] IntComtrade.GetValue(int iX,
                                 [int col])
```

ARGUMENTS

d (out)	The value retrieved from the data.
iX	The row.
col (optional)	The curve number, which equals the variable or column number, first column value (time,index, etc.) is returned when omitted.

RETURNS

0	when ok
1	when iX out of bound
2	when col out of bound
3	when invalid value is returned from a sparse file. Sparse files are written e.g. by the contingency, the value is invalid in case that it was not written, bcause it was below the recording limit. Result files created using DPL/Python are always full and will not return invalid values.

GetVariable

Get variable name of column

```
str IntComtrade.GetVariable([int column])
```

ARGUMENTS

column (optional)
The column index. The variable name of the default variable is returned if the parameter is nor passed to the function.

RETURNS

Returns the variable name which is empty in case that the column index is not part of the data.

Load

Loads the data of a result object (**IntComtrade**) in memory for reading.

```
None IntComtrade.Load()
```

Release

Releases the data loaded to memory. This function should be used whenever several result objects are processed in a loop. Data is always released from memory automatically after execution of the current script.

```
None IntComtrade.Release()
```

SortAccordingToColumn

Sorts all rows in the data loaded according to the given column. The IntComtrade itself remains unchanged.

```
int IntComtrade.SortAccordingToColumn(int column)
```

ARGUMENTS

col The column number.

RETURNS

0	The function executed correctly, the data was sorted correctly according to the given column.
1	The column with index column does not exist.

4.6.8 IntComtradeset

Overview

FindColumn
FindMaxInColumn
FindMinInColumn
GetDescription
GetNumberOfColumns
GetNumberOfRows
GetUnit
GetValue
GetVariable
Load
Release
SortAccordingToColumn

FindColumn

Returns the first column matching the variable name.

```
int IntComtradeset.FindColumn(str variable,
                              [int startCol]
                              )
```

ARGUMENTS

variable The variable name to look for.

startCol (optional)
The index of the column at which to start the search.

RETURNS

≥ 0	The column index found.
< 0	The column with name variable was not found.

FindMaxInColumn

Find the maximum value of the variable in the given column.

```
[int row,
float value] IntComtradeset.FindMaxInColumn(int column)
```

ARGUMENTS

column The column index.

value (optional, out)
: The maximum value found. The value is 0. in case that the maximum value was not found.

RETURNS

< 0	The maximum value of column was not found.
≥ 0	The row with the maximum value of the column.

FindMinInColumn

Find the minimum value of the variable in the given column.

```
[int row,
float value] IntComtradeset.FindMinInColumn(int column)
```

ARGUMENTS

column The column index.

value (optional, out)
: The minimum value found. The value is 0. in case that the minimum value was not found.

RETURNS

< 0	The minimum value of column was not found.
≥ 0	The row with the minimum value of the column.

GetDescription

Get the description of a column.

```
str IntComtradeset.GetDescription([int column],
                                  [int short]
                                  )
```

ARGUMENTS

column (optional)
: The column index. The description name of the default variable is returned if the parameter is nor passed to the function.

short (optional)

0	long desc. (default)
1	short description

RETURNS

Returns the description which is empty in case that the column index is not part of the data.

GetNumberOfColumns

Returns the number of variables (columns) in result file excluding the default variable (e.g. time for time domain simulation).

```
int IntComtradeset.GetNumberOfColumns()
```

RETURNS

Number of variables (columns) in result file.

GetNumberOfRows

Returns the number of values per column (rows) stored in result object.

```
int IntComtradeset.GetNumberOfRows()
```

RETURNS

Returns the number of values per column stored in result object.

GetUnit

Get the unit of a column.

```
str IntComtradeset.GetUnit([int column])
```

ARGUMENTS

column (optional)
The column index. The unit of the default variable is returned if the parameter is nor passed to the function.

RETURNS

Returns the unit which is empty in case that the column index is not part of the data.

GetValue

Returns a value from a result object for row iX of curve col.

```
[int error,
float d   ] IntComtradeset.GetValue(int iX,
                                    [int col])
```

ARGUMENTS

d (out) The value retrieved from the data.

iX The row.

col (optional)
The curve number, which equals the variable or column number, first column value (time,index, etc.) is returned when omitted.

RETURNS

0	when ok
1	when iX out of bound
2	when col out of bound
3	when invalid value is returned from a sparse file. Sparse files are written e.g. by the contingency, the value is invalid in case that it was not written, bcause it was below the recording limit. Result files created using DPL/Python are always full and will not return invalid values.

GetVariable

Get variable name of column

```
str IntComtradeset.GetVariable([int column])
```

ARGUMENTS

column (optional)
The column index. The variable name of the default variable is returned if the parameter is nor passed to the function.

RETURNS

Returns the variable name which is empty in case that the column index is not part of the data.

Load

Loads the data of a result object (**IntComtradeset**) in memory for reading.

```
None IntComtradeset.Load()
```

Release

Releases the data loaded to memory. This function should be used whenever several result objects are processed in a loop. Data is always released from memory automatically after execution of the current script.

```
None IntComtradeset.Release()
```

SortAccordingToColumn

Sorts all rows in the data loaded according to the given column. The IntComtradeset itself remains unchanged.

```
int IntComtradeset.SortAccordingToColumn(int column)
```

ARGUMENTS

col	The column number.

RETURNS

0	The function executed correctly, the data was sorted correctly according to the given column.
1	The column with index column does not exist.

4.6.9 IntDataset

Overview

AddRef
All
Clear
GetAll

AddRef

Adds new reference(s) for passed object(s) as children to the dataset object. Nothing happens if there exists already a reference for the passed object.

```
None IntDataset.AddRef(DataObject object)
None IntDataset.AddRef(list objects)
```

Arguments

obj/objects
: Object(s) for which references should be created and added to the dataset object

All

Returns all children of the dataset object.

```
list IntDataset.All()
```

Returns

All objects contained in dataset object.

Clear

Deletes all children of the dataset object.

```
None IntDataset.Clear()
```

GetAll

Returns all children of the dataset filtered according to given class name.

```
list IntDataset.GetAll(str className)
```

Arguments

className
: class name filter, e.g. ElmTerm

Returns

All objects of given class stored in dataset object.

4.6.10 IntEvt

Overview

CreateCBEvents
RemoveSwitchEvents

CreateCBEvents

Create boundary breaker events for all shc locations which occur simultaneously in this fault case.

```
None IntEvt.CreateCBEvents([int iRemoveExisting])
```

ARGUMENTS

iRemoveExisting (optional)

-1	Query user if circuit breaker events exist.
0	Do not create circuit breaker events if circuit breaker events are already defined events exist (default)
1	Remove existing circuit breaker events.

RemoveSwitchEvents

Remove all switch events of this fault case.

```
None IntEvt.RemoveSwitchEvents([int onlyContingency])
```

ARGUMENTS

onlyContingency (optional)
Condition to remove.

0	Remove all switch events regardless of the calculation type.
1	Remove all switch events only when this fault case is used for contingency analysis.

4.6.11 IntExtaccess

Overview

CheckUrl

CheckUrl

Checks whether access to given url will be granted or not according to the security settings. See also IntUrl.View() for accessing that url.

```
int IntExtaccess.CheckUrl(str url)
```

ARGUMENTS

url url to check

RETURNS

0	access granted
1	access denied

4.6.12 IntGrf

Overview

MoveToLayer

MoveToLayer

Moves an annotation element stored as *IntGrf* object to an annotation layer (*IntGrflayer*) or group (*IntGrfgroup*).

```
None IntGrf.MoveToLayer(DataObject layer)
```

ARGUMENTS

layer	Target *IntGrflayer* or *IntGrfgroup* object.

4.6.13 IntGrfgroup

Overview

ClearData
Export
Import

ClearData

Removes all annotation elements from this group.

```
None IntGrfgroup.ClearData()
```

Export

Exports all objects of a group into svg-file.

```
None IntGrfgroup.Export(str path,
                        [int OpenDialog])
```

ARGUMENTS

path Full export file path

OpenDialog (optional)
Prompt for export path in dialog

0	Export directly and do not show any dialog (default)
1	Show dialog with path before exporting

Import

Imports svg-file into group object.

```
None IntGrfgroup.Import(str path)
```

ARGUMENTS

path	Path of file to be imported.

4.6.14 IntGrflayer

Overview

ClearData
Export
Import

ClearData

Removes all annotation elements on this layer (keeps contained groups and annotation elements).

```
None IntGrflayer.ClearData()
```

Export

Exports all objects of a layer into svg-file, inclusive annotation objects of contained group objects.

```
None IntGrflayer.Export(str path,
                        [int OpenDialog])
```

ARGUMENTS

path Full export file path

OpenDialog (optional)
Prompt for export path in dialog

0	Export directly and do not show any dialog (default)
1	Show dialog with path before exporting

Import

Imports svg file into layer.

```
None IntGrflayer.Import(str path)
```

ARGUMENTS

path	Path of file to be imported.

4.6.15 IntGrfnet

Overview

SetLayerVisibility
Show

SetLayerVisibility

Sets a layer visible or invisible.

```
None IntGrfnet.SetLayerVisibility(str sLayer,
                                  int iVis)
```

ARGUMENTS

sLayer Layer to be modified.

iVis Visiblity

0	Make layer invisible.
1	Make layer visible.

Show

Opens a diagram.

```
int IntGrfnet.Show()
```

RETURNS

0	On success, no error occurred.
1	Otherwise

4.6.16 IntMat

Overview

ColLbl
Get
GetColumnLabel
GetNumberOfColumns
GetNumberOfRows
GetRowLabel
Init
Invert
Multiply
Resize
RowLbl
Save
Set
SetColumnLabel
SetRowLabel
SortToColumn

ColLbl

Deprecated function to get or set the label of the given column. Please use IntMat.GetColumnLabel() or IntMat.SetColumnLabel() instead.

```
str IntMat.ColLbl(int column)
str IntMat.ColLbl(str label,
                  int column
```

```
                    )
```

Get

Returns the value at the position (row, column) of the matrix. A run-time error will occur when 'row' or 'column' is out of range.

```
float IntMat.Get(int row,
                 int column
                 )
```

ARGUMENTS

row	Row in matrix: 1 ... GetNumberOfRows().
column	column in matrix: 1 ... GetNumberOfColumn()

RETURNS

Value in matrix.

SEE ALSO

IntMat.Set()

GetColumnLabel

Returns the label of a column.

```
str IntMat.GetColumnLabel(int column)
```

ARGUMENTS

column	Column index (first column has index 1).

RETURNS

Column label of given column.

DEPRECATED NAMES

ColLbl

SEE ALSO

IntMat.SetColumnLabel(), IntMat.GetRowLabel()

GetNumberOfColumns

Returns the number of columns in the matrix.

```
int IntMat.GetNumberOfColumns()
```

RETURNS

The number of columns of the matrix.

DEPRECATED NAMES

NCol, SizeY

SEE ALSO

IntMat.GetNumberOfRows()

GetNumberOfRows

Returns the number of rows in the matrix.

```
int IntMat.GetNumberOfRows()
```

RETURNS

The number of rows.

DEPRECATED NAMES

NRow, SizeX

SEE ALSO

IntMat.GetNumberOfColumns()

GetRowLabel

Returns the label of a row.

```
str IntMat.GetRowLabel(int row)
```

ARGUMENTS

row	Row index (first row has index 1).

RETURNS

Row label of given row.

DEPRECATED NAMES

RowLbl

SEE ALSO

IntMat.SetRowLabel(), IntMat.GetColumnLabel()

Init

Initializes the matrix with given size and values, regardless of the previous size and data.

This operation is performed in memory only. Use IntMat.Save() to save the modified matrix to database.

```
int IntMat.Init(int numberOfRows,
                int numberOfColumns,
                [float initialValue = 0.0]
                )
```

ARGUMENTS

numberOfRows
The number of rows.

numberOfColumns
The number of columns.

initialValue (optional)
Initial values: All matrix entries are initialised with this value. Matrix is initialized with 0 if ommited.

RETURNS

Always 1 and can be ignored.

SEE ALSO

IntMat.Resize()

Invert

Inverts the matrix.

This operation is performed in memory only. Use IntMat.Save() to save the modified matrix to database.

```
int IntMat.Invert()
```

RETURNS

0	Success, the matrix is replaced by its inversion.
1	Error, inversion not possible. Original matrix was not changed.

Multiply

Multiplies 2 matrixes and stores the result in this matrix.

This operation is performed in memory only. Use IntMat.Save() to save the modified matrix to database.

```
int IntMat.Multiply(DataObject M1,
                    DataObject M2
                    )
```

ARGUMENTS

object M1
Matrix 1 to be multiplied.

object M2
Matrix 2 to be multiplied.

RETURNS

Always 0 and can be ignored.

Resize

Resizes the matrix to a given size. Existing values will not be changed. Added values will be set to the optional value, otherwise to 0.

This operation is performed in memory only. Use IntMat.Save() to save the modified matrix to database.

```
int IntMat.Resize(int numberOfRows,
                  int numberOfColumns,
                  [float initialValue = 0.0]
                  )
```

Arguments

numberOfRows
The number of rows.

numberOfColumns
The number of columns.

initialValue (optional)
Initial values: Additional matrix entries are initialised with this value. Additional values are initialized with 0. if ommited.

Returns

Always 1 and can be ignored.

See also

IntMat.Init()

RowLbl

Deprecated function to get or set the label of the given row. Please use IntMat.GetRowLabel() or IntMat.SetRowLabel() instead.

```
str IntMat.RowLbl(int row)
str IntMat.RowLbl(str label,
                  int row
                  )
```

Save

Saves the current state of this matrix to database.

```
None IntMat.Save()
```

Set

Sets a value at the position (row, column) of the matrix. The matrix is resized automatically if the given coordinates exceed the size.

This operation is performed in memory only. Use IntMat.Save() to save the modified matrix to database.

```
int IntMat.Set(int row,
               int column,
               float value
               )
```

Arguments

row Row index, 1 based. The first row has index 1. Invalid index ($leq0$) leads to scripting error.

column	Column index, 1 based. The first column has index 1. Invalid index ($leq0$) leads to scripting error.
value	Value to assign.

RETURNS

Always 1 and can be ignored.

SEE ALSO

IntMat.Get()

SetColumnLabel

Sets the label of a column.

This operation is performed in memory only. Use IntMat.Save() to save the modified matrix to database.

```
None IntMat.SetColumnLabel(int column,
                           str label
                           )
```

ARGUMENTS

column	Column index (first column has index 1).
label	Label to set.

SEE ALSO

IntMat.GetColumnLabel(), IntMat.SetRowLabel()

SetRowLabel

Sets the label of a row.

This operation is performed in memory only. Use IntMat.Save() to save the modified matrix to database.

```
str IntMat.SetRowLabel(int row,
                       str label
                       )
```

ARGUMENTS

row	Row index (first row has index 1).
label	Label to set.

SEE ALSO

IntMat.GetRowLabel(), IntMat.SetColumnLabel()

SortToColumn

Sorts the matrix alphanumerically according to a column, which is specified by the input parameter. The row labels are sorted accordingly (if input parameter storeInDB is 1).

```
int IntMat.SortToColumn(int columnIndex,
                        double epsilon = 0.0,
                        int storeInDB = 1)
```

DEPRECATED NAMES

SortToColum

ARGUMENTS

columnIndex
The column index, 1 based. The first column has index 1.

epsilon (optional)
Accuracy for comparing equal values. Values which differ less than epsilon are treated as being equal. Default value is 0.

storeInDb (optional)
Possible Values:

0	Non-persistent change. Values are not stored in database.
1	Values are stored in database. (default)

RETURNS

0	On success.
1	Error. Original matrix was not changed.

4.6.17 IntMon

Overview

AddVar
ClearVars
GetVar
NVars
PrintAllVal
PrintVal
RemoveVar

AddVar

Appends the variable “name” to the list of selected variable names.

```
None IntMon.AddVar(str name)
```

ARGUMENTS

name The variable name to add.

ClearVars

Clears the list of selected variable names.

```
None IntMon.ClearVars()
```

GetVar

Returns the variable name on the given row of the variable selection text on the second page of the IntMon dialogue, which should contain one variable name per line.

```
str IntMon.GetVar(int row)
```

ARGUMENTS

row	Given row

RETURNS

The variable name in line row.

NVars

Returns the number of selected variables or, more exact, the number of lines in the variable selection text on the second page of the IntMon dialogue, which usually contains one variable name per line.

```
int IntMon.NVars()
```

RETURNS

The number of variables selected.

PrintAllVal

Writes all calculation results of the object assigned in obj_id to the output window. The output includes the variable name followed by the value, its unit and the description.
It should be noted that the variable set itself is modified by this method.

```
None IntMon.PrintAllVal()
```

PrintVal

Prints the values of the selected variables to the output window.

```
None IntMon.PrintVal()
```

RemoveVar

Removes the variable “name” from the list of selected variable names.

```
int IntMon.RemoveVar(str name)
```

ARGUMENTS

name	The variable name.

RETURNS

0	If variable with name was found and removed.
1	If the variable name was not found.

4.6.18 IntOutage

Overview

Apply

```
None IntOutage.Apply([int reportSwitches])
```

Applies the outage object. The functionality corresponds to pressing the ’Apply’ button in edit dialog with the difference that the scripting function can also be used without an active scenario.

ARGUMENTS

reportSwitches (optional)
Flag to enable the reporting of changed switches to the output window.

0	No output (default)
1	Print switches to output window

ApplyAll

```
None IntOutage.ApplyAll([int reportSwitches])
```

Applies all currently relevant (=in study time and not out-of-service) outage objects of current project. The functionality corresponds to pressing the ’ApplyAll’ button in edit dialog with the difference that the scripting function can also be used without an active scenario. It applies all relevant outages independent of the one it was called on.

ARGUMENTS

reportSwitches (optional)
Flag to enable the reporting of changed switches to the output window.

0	No output (default)
1	Print switches to output window

Check

```
int IntOutage.Check([int outputMessage])
```

This function checks if the outage is correctly reflected by the network elements.

ARGUMENTS

outputMessage (optional)
Flag to enable detailed output to the output window.

0	No output (default)
1	Detailed report of mismatch to output window

RETURNS

0	Ok, outage is correctly reflected
1	Not ok, status of network elements does not reflect outage

CheckAll

This function checks if all outages are correctly reflected by the network components for current study time. It checks all outages independent of the one it was called on.

```
[list notOutaged,
list wronglyOutaged] IntOutage.CheckAll([int emitMsg,]
                                        [DataObject gridfilter,])
```

ARGUMENTS

int emitMsg (optional)
whether to report inconistencies to the output window

-1	No output
0	(Default) print inconsistencies but without start / end message
1	Full output, including start / end message

gridfilter (optional)
Possibility to restrict checking for accidentally outaged elements to given object (e.g. grid) and its children (by default, all elements for all active grids are checked).

notOutaged (optional, out)(optional)
If given, all network components that should be outaged but are not are filled into this set.

wronglyOutaged (optional, out)(optional)
If given, all network components that should be outaged but are not are filled into this set.

IsInStudyTime

```
int IntOutage.IsInStudyTime()
```

Checks if outage is relevant for current study time, i.e. the study time lies within the outage's validity period.

RETURNS

0	Outage is not relevant for current study time (outside validity period)
1	Outage is relevant for current study time (inside validity period)

DEPRECATED NAMES

IsInStudytime

ResetAll

```
None IntOutage.ResetAll([int reportSwitches])
```

Resets all currently relevant (=in study time and not out-of-service) outage objects of current project. The functionality corresponds to pressing the 'Reset' button in all outage objects with difference that the scripting function can also be used without an active scenario. It resets all relevant outages independent of the one it was called on.

ARGUMENTS

reportSwitches (optional)
Flag to enable the reporting of changed switches to the output window.

0	No output (default)
1	Print switches to output window

4.6.19 IntPlot

Overview

SetAdaptY
SetAutoScaleY
SetScaleY

SetAdaptY

Sets the Adapt Scale option of the x-scale.

```
None IntPlot.SetAdaptY(int mode,
                       [float offset]
                       )
```

ARGUMENTS

mode Possible values:

0	off
1	on

offset (optional)
Offset, unused if mode is off or empty

SetAutoScaleY

Sets automatic scaling mode of the y-scale. A warning is issued if an invalid mode is passed to the function.

```
None IntPlot.SetAutoScaleY(int mode)
```

ARGUMENTS

mode Possible values:

0	never
1	after simulation
2	during simulation

SetScaleY

Sets y-axis scale limits. A function call without arguments sets the Auto Scale setting to On without changing the scale itself.

```
None IntPlot.SetScaleY()
None IntPlot.SetScaleY(float min,
                       float max,
                       [int log]
                       )
```

ARGUMENTS

min (optional)
Minimum of y-scale.

max (optional)
Maximum of y-scale.

log (optional)
Possible values:

0	linear
1	logarithmic

4.6.20 IntPrj

Overview

- Activate
- AddProjectToCombined
- AddProjectToRemoteDatabase
- Archive
- CanAddProjectToRemoteDatabase
- CanSubscribeProjectReadOnly
- CanSubscribeProjectReadWrite
- CreateVersion
- Deactivate
- GetDerivedProjects
- GetLatestVersion
- GetVersions
- HasExternalReferences
- LoadData
- Migrate
- Purge
- RemoveProjectFromCombined
- Restore
- SubscribeProjectReadOnly
- SubscribeProjectReadWrite
- UnsubscribeProject
- UpdateStatistics

Activate

Activates the project. If another project is already activated it will be deactivated first.

```
int IntPrj.Activate()
```

RETURNS

0	on success
1	on error

AddProjectToCombined

Adds a project to this using the Project Combination logic. The passed object must be an IntVersion. The receiving project must be activated but not have a Study Case active, otherwise this method will fail.

```
int IntPrj.AddProjectToCombined(object projectVersion)
```

ARGUMENTS

projectVersion
The verson of a project to add

RETURNS

0	operation was successful
1	an error occurred

AddProjectToRemoteDatabase

Adds a project to the online database if possible.

Can only be used if the database driver is set to Offline Mode.

```
int IntPrj.AddProjectToRemoteDatabase()
```

Archive

Archives the project if the functionality is configured and activated. Does nothing otherwise.

```
int IntPrj.Archive()
```

RETURNS

0	project has been archived
1	project has not been archived

CanAddProjectToRemoteDatabase

Checks if the project can be pushed to the remote database.

The project must be subscribable as read and write and it must be unsubscribed. Can only be used if the database driver is set to Offline Mode.

```
int IntPrj.CanAddProjectToRemoteDatabase()
```

RETURNS

0	project cannot be added to the remote database
1	project can be added to the remote database

CanSubscribeProjectReadOnly

Checks if a project can be subscribed read-only by the user executing the script.

```
int IntPrj.CanSubscribeProjectReadOnly()
```

RETURNS

0	no permission to subscribe project
1	project can be subscribed

CanSubscribeProjectReadWrite

Checks if a project can be subscribed read-write by the user executing the script.

```
int IntPrj.CanSubscribeProjectReadWrite()
```

RETURNS

0	no permission to subscribe project
1	project can be subscribed

CreateVersion

Creates a new version of project it was called on.

Optionally allows to pass a bool to enforce project approval and user notifications, and version name.

```
DataObject IntPrj.CreateVersion([int notifyUsersAndApprovalRequired [, str name]])
```

ARGUMENTS

notifyUsersAndApprovalRequired
Project approval required and user notifications activated:

0	Create version without approval and do not notify users (default).
1	Require approval and notify users.

name Version name

RETURNS

DataObject	Newly created *IntVersion* object.
None	On failure e.g. missing permission rights.

Deactivate

De-activates the project if it is active. Does nothing otherwise.

```
int IntPrj.Deactivate()
```

RETURNS

0	on success
1	on error

GetDerivedProjects

Return a set holding all versions created in the project.

```
list IntPrj.GetDerivedProjects()
```

RETURNS

Set holding all versions of a project.

GetLatestVersion

Returns the most recent version available in the project which has the notify users option set.

Optionally allows to consider all versions, regardless of notify users option.

```
DataObject IntPrj.GetLatestVersion([int onlyregular])
```

ARGUMENTS

onlyregular (optional)

1	consider only regular version (default)
0	consider all versions

RETURNS

Latest version of the project

GetVersions

Returns a set containing all versions of the project.

```
list IntPrj.GetVersions()
```

RETURNS

Set that contains all versions of the project

HasExternalReferences

Checks if any object inside the project references external non-system objects and prints a report to the Output Window.

```
int IntPrj.HasExternalReferences([int iCheckGlobal,]
                                 [int iCheckRemoteVariants]
                                 )
```

ARGUMENTS

iCheckGlobal (optional)

0	global (non-system) references are ok
1	gloabal (non-system) references are not ok (default)

iCheckRemoteVariants (optional)

0	remote variants are ok (default)
1	remote variants are not ok

RETURNS

0	no forbidden external references found
1	some forbidden external references were found

LoadData

Loads all objects of the project from the data base.

This function is useful to optimise searches which would traverse deep into an inactive project.

```
None IntPrj.LoadData()
```

Migrate

Migrates a project from version V13 to V14. Migration is only executed if project has been created in build 400 or earlier (and is not yet migrated).

```
None IntPrj.Migrate([int createCopy])
```

ARGUMENTS

createCopy (optional)

1	Creates a copy of current project (original copy is maintained) (default)
0	Does an ”in-place” migration of the project (original is overwritten)

Purge

Purges project storage and updates storage statistics.

Requires write access to the project; the functions does nothing when the project is locked by another user.

```
None IntPrj.Purge()
```

RemoveProjectFromCombined

Removes a project from a combined project. For the removal the mapping key must be specified. Mapping keys are stored in the project, parameter project_mapped. The project this method is called on must be activated but not have a Study Case active, otherwise this method will fail.

```
int IntPrj.RemoveProjectFromCombined(str mappingKey)
```

ARGUMENTS

mappingKey
The mapping key for the project that should be removed

RETURNS

0	operation was successful
1	an unknown error occurred
2	an error occurred and is documented in the output window

Restore

Restores an archived project so it can be used again. Does nothing if the project is not an archived one.

```
int IntPrj.Restore()
```

RETURNS

0	project has not been restored
1	project has been restored

SubscribeProjectReadOnly

Subscribes a project read only if the permission is granted.

Can only be used if the database driver is set to Offline Mode.

```
None IntPrj.SubscribeProjectReadOnly()
```

SubscribeProjectReadWrite

Subscribes a project read/write if the permission is granted.

Can only be used if the database driver is set to Offline Mode.

```
None IntPrj.SubscribeProjectReadWrite()
```

UnsubscribeProject

Unsubscribes a project.

Can only be used if the database driver is set to Offline Mode.

```
None IntPrj.UnsubscribeProject()
```

UpdateStatistics

Updates the storage statistics for a project. The statistics are displayed on the page Storage of a project.

Note: This function requires write access to the project otherwise the update is not executed and an error message is printed to the output window.

```
None IntPrj.UpdateStatistics()
```

4.6.21 IntPrjfolder

Overview

GetProjectFolderType
IsProjectFolderType

GetProjectFolderType

Returns the type of the project folder stored in attribute “iopt_type”.
The following types are currently available (language independent):

- blk - User Defined Models
- cbrat - CB Ratings
- chars - Characteristics
- cim - CIM Model
- common - Common Mode Failures
- demand - Demand Transfers
- dia - Diagrams
- equip - Equipment Type Library
- fault - Faults
- gen - Generic
- lib - Library
- mvar - Mvar Limit Curves
- netdat - Network Data
- netmod - Network Model
- oplib - Operational Library
- outage - Outages
- qpc - QP-Curves
- ra - Running Arrangements
- report - Table Reports
- scen - Operation Scenarios
- scheme - Variations
- script - Scripts
- study - Study Cases
- sw - StationWare
- tariff - Tariffs
- templ - Templates
- therm - Thermal Ratings

```
str IntPrjfolder.GetProjectFolderType()
```

RETURNS

The type of the project folder as string. For possible return values see list above.

SEE ALSO

Application.GetProjectFolder()

IsProjectFolderType

This function checks if a project folder is of given type.

```
int IntPrjfolder.IsProjectFolderType(str type)
```

ARGUMENTS

type	Folder type; for possible type values see IntPrjfolder.GetProjectFolderType()

RETURNS

1	true, is of given type
0	false, is not of given type

SEE ALSO

Application.GetProjectFolder(), IntPrjfolder.GetProjectFolderType()

4.6.22 IntQlim

Overview

GetQlim

GetQlim

Returns either the current maximum or the minimum reactive power limit, given the specified active power and voltage.
The active power must be given in the same units as the input mode definition of the capability curve object (parameter ”inputmod” is 0 for MW/Mvar and 1 for p.u.).

```
float IntQlim.GetQlim(float p,
                      float v,
                      [float minmax]
                      )
```

ARGUMENTS

p	the current value of active power in MW or p.u.
v	the current value of voltage in p.u.

minmax (optional)
Returns either the maximum or minimum value. Possible values are:

0	minimum value. This is the default value
1	maximum value

RETURNS

Returns the minimum/maximum limit. The units might be Mvar or p.u., depending on the input mode of the capability curve. Also, the limits are calculated for a single machine.

4.6.23 IntRunarrange

Overview

GetSwitchStatus

GetSwitchStatus

Determines the status of the given switch in the running arrangement, without assigning or applying the running arrangement.

```
int GetSwitchStatus(DataObject switch)
```

ARGUMENTS

switch	*ElmCoup* or *StaSwitch* from which to get the status stored in running arrangement

RETURNS

Status of the switch in the running arrangement. Possible values are

-1	Switch is not part of the running arrangment
0	Switch is open
1	Switch is closed

4.6.24 IntScenario

Overview

Activate
Apply
ApplySelective
Deactivate
DiscardChanges
GetObjects
GetOperationValue
ReleaseMemory
Save
SetOperationValue

Activate

Activates a scenario. If there is currently another scenario active that one will be deactivated automatically.

```
int IntScenario.Activate()
```

RETURNS

0	successfully activated
1	error, e.g. already activate, no project and study case active

Apply

Copies the values stored in a scenario to the corresponding network elements. The value transfer is identical to scenario activation, however, the scenario will not be activated. In case of having an active variation or another scenario, the values will be recorded there.

```
int IntScenario.Apply([int requestUserConfirmation])
int IntScenario.Apply(int requestUserConfirmation,
                      DataObject parentfilter
                      )
```

ARGUMENTS

requestUserConfirmation(optional)

0	silent, just apply the data without further confirmation requests
1	request a user confirmation first (default)

parentfilter (optional)
If given, scenario data is only applied for given object and all of its children (hierarchical filter)

RETURNS

0 on success

ApplySelective

Similar to function Apply() but copies only the set of attributes listed in the given apply configuration. An apply configuration is a folder consisting of variable selection objects (IntMon), one per class. For each class the attributes to be copied can be selected.

```
int IntScenario.ApplySelective(DataObject configuration)
int IntScenario.ApplySelective(int requestUserConfirmation,
                               DataObject applyConfiguration
                               )
```

ARGUMENTS

applyConfiguration
folder containing variable selection objects

requestUserConfirmation(optional)

0	silent, just apply the data without further confirmation requests
1	request a user confirmation first (default)

RETURNS

0 on succes

Deactivate

Deactivates the currently active scenario.

```
int IntScenario.Deactivate([int saveOrUndo])
```

ARGUMENTS

saveOrUndo(optional)
Determines whether changes in active scenario will be saved or discarded before the scenario is deactivated. If this argument is omitted, the user will be asked.

0	discard changes
1	save changes

RETURNS

0 on success

DiscardChanges

Discards all unsaved changes made to a scenario.

```
int IntScenario.DiscardChanges()
```

RETURNS

0	on success
1	error, scenario was not modified or not active

GetObjects

Returns a set of all objects for which operational data are stored in scenario.

```
list IntScenario.GetObjects()
```

RETURNS

Set of all objects for which operational data are stored in scenario

GetOperationValue

This function offers read access to the operation data values stored in the scenario.

```
[int error,
int|float|str|DataObject value] IntScenario.GetOperationValue(DataObject obj,
                                                              str attribute,
                                                              [int fromObject])
```

ARGUMENTS

value (out)
variable that holds the value after call

obj object for which the operation to be retrieved

attribute name of the operation data attribute

fromObject
only if current scenario is active:

0	value is taken from scenario (as stored on db)
1	(default), value is taken from object (reflects un-saved modifications)

RETURNS

0 on success

ReleaseMemory

Releases the memory used by a scenario. Any further access to the scenario will reload the data from database. The function can be called on inactive scenarios only. Use this function with care!

```
int IntScenario.ReleaseMemory()
```

RETURNS

0	on success
1	error, scenario is active

Save

Saves the current active value of all operational attributes for all active network elements to database.

```
int IntScenario.Save()
```

RETURNS

0	successfully saved
1	error, scenario was not modified or not active

SetOperationValue

Offers write access to operational data stored in a scenario.

```
int IntScenario.SetOperationData(float newvalue,
                                 DataObject obj,
                                 str attribute,
                                 [int toObject]
                                 )
int IntScenario.SetOperationData(int newvalue,
                                 DataObject obj,
                                 str attribute,
                                 [int toObject]
                                 )
int IntScenario.SetOperationData(str newvalue,
                                 DataObject obj,
                                 str attribute,
                                 [int toObject]
                                 )
int IntScenario.SetOperationData(DataObject newvalue,
                                 DataObject obj,
                                 str attribute,
                                 [int toObject]
                                 )
```

ARGUMENTS

newvalue new value to store in the scenario

obj	object for which the operation data to store
attribute	name of the operation data attribute
toObject	only if current scenario is active:

0	value is only stored to scenario on db
1	(default), as 0 but value is also updated on object in memory

RETURNS

0 on success

4.6.25 IntScensched

Overview

Activate
Deactivate
DeleteRow
GetScenario
GetStartEndTime
SearchScenario

Activate

Activates a scenario scheduler.

```
int IntScensched.Activate()
```

RETURNS

0	successfully activated
1	error, e.g. already activate, no project and study case active

Deactivate

Deactivates a scenario scheduler.

```
int IntScensched.Deactivate()
```

RETURNS

0	successfully deactivated
1	error, e.g. already deactivates, no project and study case active

DeleteRow

Delete row(s) of the scenario scheduler.

```
None IntScensched.DeleteRow(int row, [int numberOfRows])
```

ARGUMENTS

row	row number (begin with 0)

numberOfRows (optional)
number of rows to delete (default = 1)

GetScenario

Get the scenario for corresponding time 'iTime'.

```
DataObject IntScensched.GetScenario(int iTime)
```

ARGUMENTS

iTime Time (UCTE) to get the corresponding scenario.

RETURNS

None No scenario at time 'iTime'defined

IntScenario Scenario will be activated at time 'iTime'

GetStartEndTime

Gets the start and end time of the corresponding scenario.

```
[int error
int startTime,
int endTime  ] IntScensched.GetStartEndTime(DataObject scenario)
```

ARGUMENTS

scenario A scenario (*IntScenario*).

startTime (out)
Start time (time when the scenario is activated)).

endTime (out)
End time (time until the scenario is still activated).

RETURNS

-1 Scenario not found (not part of scenario scheduler)

≥ 0 Vector index (index of scenario)

SearchScenario

Search at which table index (row) the corresponding scenario is defined in the scheduler.

```
int IntScensched.SearchScenario(DataObject scenarioObject)
```

ARGUMENTS

scenarioObject
scenario object

RETURNS

-1 Scenario not found (not part of scenario scheduler).

≥ 0 Vector index (row, index of scenario).

4.6.26 IntScheme

Overview

Activate
Consolidate
Deactivate
GetActiveScheduler
NewStage

Activate

Activates a variation and inserts a variation reference in a 'Variation Configuration Folder'stored in the study case.

```
int IntScheme.Activate()
```

RETURNS

0	successfully activated
1	error, e.g. already activate, no project and study case active

Consolidate

Changes that are recorded in this variation will be permanently applied to the original location. Note: Modified scenarios are not saved.

Works only:

- for non network variation e.g. used for Mvar Limit Curves, Thermal Ratings ...
- and the variation must be activated.

```
int IntScheme.Consolidate()
```

RETURNS

0	On success.
1	If an error has occured.

Deactivate

Deactivates a variation and removes the variation reference in the 'Variation Configuration Folder'stored in the study case.

```
int IntScheme.Deactivate()
```

RETURNS

0	successfully deactivated
1	error, e.g. already deactivated, no project and study case active

GetActiveScheduler

Returns the corresponding active variation scheduler or None if no scheduler is active for this variation (IntScheme).

```
DataObject IntScheme.GetActiveScheduler()
```

NewStage

Adds a new expansion stage into the variation (name = sname).

```
int IntScheme.NewStage(str sname,
                       int iUTCtime,
                       int iActivate
                       )
```

ARGUMENTS

sname Name of the new expansion stage.

iUTCtime Activation time of the new expansion stage.

iActivate

1	The actual study time is changed to the parameter iUTCtime and the varation will be activated. If the variation is a network variation, the new created expansion stage is used as 'recording 'expansion stage. If the variation (this) is not active, the variation will be automatically activated.
0	Expansion stage and/or variation will not be activated.

4.6.27 IntSscheduler

Overview

Activate
Deactivate
Update

Activate

Activates a variation scheduler. An already activated scheduler for same variation will be deactivated automatically.

```
int IntSscheduler.Activate()
```

RETURNS

$= 0$	On success
$\neq 0$	If an error has occurred

Deactivate

Deactivates a variation scheduler.

```
int IntSscheduler.Deactivate()
```

RETURNS

$= 0$	on success
$\neq 0$	If an error has occurred especially if scheduler was not active (to be consistent with scenario scheduler deactivate()).

Update

Update variation scheduler (updates internal reference stages).

```
int IntSscheduler.Update()
```

RETURNS

$= 0$	On success
$\neq 0$	If an error has occurred

4.6.28 IntSstage

Overview

Activate
CreateStageObject
EnableDiffMode
GetVariation
IsExcluded
PrintModifications
ReadValue
WriteValue

Activate

Activates the expansion stage and sets the 'recording' expansion stage. The study time will be automatically set to the correponsing time of the stage.

```
int IntSstage.Activate([int iQueryOption])
```

ARGUMENTS

iQueryOption

0	(default) The user must confirm the query.
1	The “Yes” button is automatically applied.
2	The “No” button is automatically applied.

RETURNS

0	Successfully activated.
1	Error, e.g. scheme is not active.

CreateStageObject

Creates a stage object (delta or delete object) inside corresponding *IntSstage*.

```
DataObject IntSstage.CreateStageObject(int type,
                                        DataObject rootObject
                                        )
```

Arguments

type Kind of object to create

1	Delete object
2	Delta object

rootObject
(Original) object for which the stage object should be created.

Returns

Stage object on success.

EnableDiffMode

Enables the comparison mode for the variation management system. If the mode is enabled a DELTA object is only created when the object is different.

```
None IntSstage.EnableDiffMode(int enable)
```

Arguments

enable

0	disables the difference/comparison mode
1	enables the difference/comparison mode

GetVariation

Returns variation of expansion stage.

```
DataObject IntSstage.GetVariation()
```

Returns

Variation object corresponding to stage.

Deprecated Names

GetScheme

IsExcluded

Returns if expansion stage flag 'Exclude from Activation' is switched on (return value = 1) or not (return value = 0). The function checks also if the stage is excluded regarding the restricted validity period of the corresponding variation and considers also the settings of an variation scheduler when defined.

```
float IntSstage.IsExcluded()
```

RETURNS

1	if stage is excluded
0	if stage is considered

PrintModifications

Reports in the the output window the modification of the corresponding expansion stage. Works only if the expansion stage is the active 'recording 'expansion stage.

```
int IntSstage.PrintModifications([int onlyNetworkData,]
                                 [str ignoredParameter]
                                 )
```

ARGUMENTS

onlyNetworkData (optional)

1	(default) Show only network data modifications. Graphical modifications are not report when the diagrams folder are recored.
0	Show all modifications

ignoredParameter (optional)
Comma separated list of parameters which are ignored for reporting.

RETURNS

0	on success
1	if the actual expansion stage is not the 'recording 'expansion stage.

ReadValue

Get the value for an attribute of an ADD or DELTA object which modifies “rootObj” (root object).

```
[int error,
float|str|DataObject value] IntSstage.ReadValue(DataObject rootObj,
                                                str attributeName)
```

RETURNS

$= 0$	On success.
$\neq 0$	Error e.g. wrong data type.

WriteValue

Writes a value for an attribute to an ADD or DELTA object which modifies rootObj (root object).

```
int IntSstage.WriteValue(float|str|DataObject value,
                         DataObject rootObj,
                         str attributeName)
```

RETURNS

$= 0$	On success.
$\neq 0$	Error e.g. wrong data type.

4.6.29 IntSubset

Overview

Apply

Copies the values stored in a scenario to the corresponding network elements. The value transfer is identical to scenario activation, however, the scenario will not be activated. In case of having an active variation or another scenario, the values will be recorded there.

```
int IntSubset.Apply([int requestUserConfirmation])
```

ARGUMENTS

requestUserConfirmation(optional)

0	silent, just apply the data without further confirmation requests
1	request a user confirmation first (default)

RETURNS

0 on success

ApplySelective

Similar to function Apply() but copies only the set of attributes listed in the given apply configuration. An apply configuration is a folder consisting of variable selection objects (IntMon), one per class. For each class the attributes to be copied can be selected.

```
int IntSubset.ApplySelective(DataObject applyConfiguration)
int IntSubset.ApplySelective(int requestUserConfirmation,
                             DataObject applyConfiguration
                             )
```

ARGUMENTS

applyConfiguration
folder containing variable selection objects

requestUserConfirmation(optional)

0	silent, just apply the data without further confirmation requests
1	request a user confirmation first (default)

RETURNS

0 on succes

Clear

Clears all values stored in the subset.
Please note that this function can only be called on subsets of currently in-active scenarios.

```
int IntSubset.Clear()
```

RETURNS

0	On success.
1	On error, e.g. subset belongs to a currently active scenario.

4.6.30 IntThrating

Overview

GetCriticalTimePhase
GetRating

GetCriticalTimePhase

This function returns the smallest duration (time-phase) for which the power flow is beyond the rating.

```
float IntThrating.GetCriticalTimePhase(float Flow,
                                       float Loading
                                       )
```

ARGUMENTS

Flow	Power from the load flow calculation, in MVA.
Loading	Element loading, in %.

RETURNS

$>$**0**	Smallest time-phase for which the flow is beyond the rating.
-1	In case that no rating is violated.

GetRating

This function returns the rating in MVA according to the thermal rating table, considering element overloading and its duration (time-phase).

```
float IntThrating.GetRating(float Loading,
                            float Duration
                            )
```

ARGUMENTS

Loading	Element loading, in %.
Duration	Duration or time phase for which the loading is considered, in minutes

RETURNS

Rating in MVA or 0 if not found.

4.6.31 IntUrl

Overview

View

View

Requests the operating system to open given URL for viewing. The performed action depends on the default action configured in the system. For example, by default 'http://www.google.com' would be opened in standard browser.

Please note, the action is only executed if access to given URL is enabled in the 'External Access' configuration of PowerFactory (IntExtaccess).

```
int IntUrl.View()
```

RETURNS

The returned value reports the success of the operation:

0	Success, URL was opened
1	Error, URL was not opened (because of invalid address or security reasons)

4.6.32 IntUser

Overview

Purge
SetPassword
TerminateSession

Purge

Purges project storage and updates storage statistics for all projects of the user.

Requires write access to the project; the functions does nothing when the project is locked by another user.

```
None IntUser.Purge()
```

SetPassword

Sets the password for the user the function is called on.

Note: A normal user is allowed to set the password for himself only. The administrator user is allowed to set passwords for every user.

```
None IntUser.SetPassword(str newpassword)
```

ARGUMENTS

newpassword
Case sensitive user password to set

TerminateSession

Allows the Administrator to log out another user. Prints an error if the current user is not the Administrator.

```
None IntUser.TerminateSession()
```

4.6.33 IntUserman

Overview

CreateGroup

Creates a new user group of given name. If a group with given name already exists the existing one is returned instead.

Note: Only Administrator user is allowed to call this function.

```
DataObject IntUserman.CreateGroup(str name)
```

ARGUMENTS

name	Given name of the user group

RETURNS

Created user group (IntGroup)

CreateUser

Creates a new user with given name. If the user already exists the existing one is returned instead.

Note: Only Administrator user is allowed to call this function.

```
DataObject IntUserman.CreateUser(str name)
```

ARGUMENTS

name	Given name of the user

RETURNS

Created user (IntUser)

GetGroups

Returns a container with all user groups.

Note: Only the administrator user is allowed to call this function.

```
list IntUserman.GetGroups()
```

RETURNS

Set of all available users

GetUsers

Returns a container with all users as they are currently visible in the Data Manager tree.

Note: Only the administrator user is allowed to call this function.

```
list IntUserman.GetUsers()
```

RETURNS

Set of all available users

UpdateGroups

Updates the Everybody group so it contains all currently existing users and cleans it of removed users.

```
None IntUserman.UpdateGroups()
```

4.6.34 IntVec

Overview

Get

Get the value in row index. Index is one based, therefore the index of the first entry is 1.

```
float IntVec.Get(int index)
```

ARGUMENTS

index	Index in vector, one based.

SEE ALSO

IntVec.Set()

Init

Initializes the vector. Resizes the vector and initializes all values to 0.

This operation is performed in memory only. Use IntVec.Save() to save the modified vector to database.

```
None IntVec.Init(int size)
```

ARGUMENTS

size	The new size of the vector.

Max

Gets the maximum value stored in the vector.

```
float IntVec.Max()
```

RETURNS

The maximum value stored in the vector. Empty vectors return 0 as maximum value.

Mean

Calculates the average value of the vector.

```
float IntVec.Mean()
```

RETURNS

The average value of the vector. A value of 0. is returned for empty vectors.

Min

Gets the minimum value stored in the vector.

```
float IntVec.Min()
```

RETURNS

The minimum value stored in the vector. Empty vectors return 0 as minimum value.

Resize

Resizes the vector. Inserted values are initialized to 0.

This operation is performed in memory only. Use IntVec.Save() to save the modified vector to database.

```
None IntVec.Resize(int size)
```

ARGUMENTS

size	The new size.

Save

Saves the current state of this vector to database.

```
None IntVec.Save()
```

Set

Set the value in row index. Index is one based, therefore the index of the first entry is 1. The vector is resized automatically to size index in case that the index exceeds the current vector size. Values inserted are automatically initialized to a value of 0.

This operation is performed in memory only. Use IntVec.Save() to save the modified vector to database.

```
None IntVec.Set(int index,
                float value)
```

ARGUMENTS

index	Index in vector.
value	Value to assign in row index.

SEE ALSO

IntVec.Get()

Size

Returns the size of the vector.

```
int IntVec.Size()
```

RETURNS

The size of the vector.

Sort

Sorts the vector.

This operation is performed in memory only. Use IntVec.Save() to save the modified vector to database.

```
None IntVec.Sort([int descending = 0]
```

ARGUMENTS

descending
Sort order:

0	Smallest value first (ascending, default).
1	Highest value first (descending).

4.6.35 IntVersion

Overview

CreateDerivedProject
GetDerivedProjects
GetHistoricalProject
Rollback

CreateDerivedProject

Creates a derived project from the version.

```
DataObject IntVersion.CreateDerivedProject(str name,
                                           [DataObject parent]
                                           )
```

ARGUMENTS

name The name of the project which will be created.

parent(optional)
The parent of the project which will be created. Default is the current user.

RETURNS

Returns the created project.

GetDerivedProjects

list of projects derived from this version

```
list IntVersion.GetDerivedProjects()
```

RETURNS

list of derived projects

GetHistoricalProject

Returns historic project within version

```
DataObject IntVersion.GetHistoricalProject()
```

RETURNS

Returns the historic project object

Rollback

Roll backs the project to this version. No project have to be active. Furthermore no script from the project of the version have to be running.

```
int IntVersion.Rollback()
```

RETURNS

0	on success
1	otherwise

4.6.36 IntViewbookmark

Overview

JumpTo
UpdateFromCurrentView

JumpTo

Opens the referenced diagram (if not already open) and sets the viewing area.

```
None IntViewbookmark.JumpTo()
```

UpdateFromCurrentView

Updates the bookmark's diagram and view area from the current drawing window.

```
None IntViewbookmark.UpdateFromCurrentView()
```

4.6.37 RelZpol

Overview

AssumeCompensationFactor
AssumeReRl
AssumeXeXl

AssumeCompensationFactor

Triggers a calculation of the complex compensation factor and stores the result.

```
int RelZpol.AssumeCompensationFactor()
```

RETURNS

0	The compensation factor was sucessfully calculated.
1	An error occourred (e.g. conencted branch was not found).

AssumeReRl

Triggers a calculation of the real part of the decoupled compensation factor and stores the result.

```
int RelZpol.AssumeReRl()
```

RETURNS

0	The compensation factor was sucessfully calculated.

1 An error occourred (e.g. conencted branch was not found).

AssumeXeXl

Triggers a calculation of the imaginary part of the decoupled compensation factor and stores the result.

```
int RelZpol.AssumeXeXl()
```

RETURNS

0 The compensation factor was sucessfully calculated.

1 An error occourred (e.g. conencted branch was not found).

4.6.38 StoMaint

Overview

SetElms

SetElms

Sets the maintenance elements.

```
None StoMaint.SetElms(DataObject singleElement)
None StoMaint.SetElms(list multipleElements)
```

ARGUMENTS

singleElement
single Element for Maintenance

multipleElements
multiple Elements for Maintenance

4.6.39 TypAsmo

Overview

CalcElParams

CalcElParams

Function calculates the electrical parameters from the input data. Behaves identically as the calculate button on the basic data page was pressed. Shall be applied only if the 'Slip-Torque/Current Characteristic' chosen.

```
int TypAsmo.CalcElParams()
```

RETURNS

0 Calculated successfully.

1 Error.

4.6.40 TypLne

Overview

IsCable

IsCable

Checks if the line type is a cable type.

```
int TypLne.IsCable()
```

RETURNS

1	Type is a cable
0	Type is not a cable

4.6.41 TypTr2

Overview

GetZeroSequenceHVLVT

GetZeroSequenceHVLVT

Returns the calculated star equivalent of the zero sequence impedances.

```
[int error,
float hvReal,
float hvImag,
float lvReal,
float lvImag,
float tReal ,
float tImag  ] TypTr2.GetZeroSequenceHVLVT()
```

ARGUMENTS

hvReal (out)
Real part of the HV impedance in %.

hvImag (out)
Imaginary part of the HV impedance in %.

lvReal (out)
Real part of the LV impedance in %.

lvImag (out)
Imaginary part of the LV impedance in %.

tReal (out)
Real part of the tertiary (delta) impedance in %.

tImag (out)
Imaginary part of the tertiary (delta) impedance in %.

RETURNS

0	No error occurred.
1	An error occurred; the values are invalid.

4.6.42 VisBdia

Overview

AddObjs

Adds objects to elements column in table 'Bars'.

```
None VisBdia.AddObjs(list elements)
```

ARGUMENTS

elements Elements to add in table.

AddResObjs

Adds objects to elements column in table 'Bars' (similar to AddObjs). Additionally a result file is assigned to all rows added in the 'Result File' column.

```
None VisBdia.AddResObjs(DataObject resultFileObj,
                        list elements
                        )
```

ARGUMENTS

resultFileObj
The result file to assign. Must be an object of class ElmRes.

elements Elements to add in table.

Clear

Removes all elements from plot by erasing all rows from the table named 'Bars'.

```
None VisBdia.Clear()
```

SetScaleY

Sets y-axis scale limits.

```
None VisBdia.SetScaleY(float min,
                       float max,
                       [int log]
                       )
```

ARGUMENTS

min (optional)
Minimum of y-scale.

max (optional)
Maximum of y-scale.

log (optional)
Possible values:

0	linear
1	logarithmic

SetXVariable

Set the x-axis Variable of the Distortion Analysis Diagram

```
int VisBdia.SetXVariable(str variable)
```

ARGUMENTS

variable x-axis variable to set.Length of variable must not exceed 37 characters.

RETURNS

0 if ok, 1 if variable length exceeds 37 characters,

SetYVariable

Set the y-axis variable of the Distortion Analysis Diagram

```
int VisBdia.SetYVariable(str variable)
```

ARGUMENTS

variable y-axis variable to set.Length of variable must not exceed 37 characters.

RETURNS

0 if ok, 1 if variable length exceeds 37 characters,

4.6.43 VisDraw

Overview

AddRelay
AddRelays
CentreOrigin
Clear
DoAutoScaleX
DoAutoScaleY

AddRelay

Adds a relay to the plot and sets optionally the drawing style.

```
None VisDraw.AddRelay(DataObject relay,
                      [float colour,]
                      [float style,]
                      [float width])
```

ARGUMENTS

relay The protection device (ElmRelay or RelFuse) to be added.

colour (optional)
The colour to be used.

style (optional)
The line style to be used.

width (optional)
The line width to be used.

AddRelays

Adds relays to the plot.

```
None VisDraw.AddRelays(list relays)
```

ARGUMENTS

relays The protection devices (ElmRelay or RelFuse) to be added.

CentreOrigin

Centre the origin of the plot

```
None VisDraw.CentreOrigin()
```

Clear

Removes all protection devices from the plot.

```
None VisDraw.Clear()
```

DoAutoScaleX

Scales the x-axis of the plot automatically. The function works for local x-scales only. If the x-scale is not local a warning is shown in the output window and 1 is returned by the function.

```
int VisDraw.DoAutoScaleX()
```

RETURNS

0 Automatic scaling was executed.

1 An Error occurred.

DoAutoScaleY

Scales the y-axis of the plot automatically. The function works for local y-scales only. If the x-scale is not local a warning is shown in the output window and 1 is returned by the function.

```
int VisDraw.DoAutoScaleY()
```

RETURNS

0	Automatic scaling was executed.
1	An Error occurred.

4.6.44 VisHrm

Overview

Clear
DoAutoScaleX
DoAutoScaleY
GetScaleObjX
GetScaleObjY
SetAutoScaleX
SetAutoScaleY
SetCrvDesc
SetDefScaleX
SetDefScaleY

Clear

Removes all curves by clearing table named 'Curves'.

```
None VisHrm.Clear()
```

DoAutoScaleX

Scales x-axis automatically.

```
int VisHrm.DoAutoScaleX()
```

RETURNS

0	Ok, call to DoAutoScaleX() was successfull
1	Failed, because the x-scale is not local

DoAutoScaleY

Scales y-axis automatically.

```
int VisHrm.DoAutoScaleY()
```

RETURNS

0	Ok, call to DoAutoScaleY() was successfull
1	Failed, because the y-scale is not local

GetScaleObjX

Gets the object used for scaling the x-axis.

```
DataObject VisHrm.GetScaleObjX()
```

RETURNS

this object In case that 'Use local Axis' is set to 'Local'.
the virtual instrument panel In case that 'Use local axis' is set to 'Current Page'.
the graphics board In case that 'Use local axis' is set to 'Graphics Board'.

GetScaleObjY

Gets the object used for scaling the y-axis.

```
DataObject VisHrm.GetScaleObjY()
```

RETURNS

this object In case that 'Use local Axis' is enabled.
the plot type In case that 'Use local axis' is disabled.

SetAutoScaleX

Sets Auto Scale setting of the x-scale. The scale is automatic set to local, in case that the waveform plot is using the scale of the graphics board or the virtual instrument panel.

```
None VisHrm.SetAutoScaleX(int mode)
```

ARGUMENTS

mode Possible values:

0	never
1	after simulation

SetAutoScaleY

Sets Auto Scale setting of the y-scale. The scale is automatic set to local, in case that the waveform plot is using the scale of the plot type.

```
None VisHrm.SetAutoScaleY(int mode)
```

ARGUMENTS

mode Possible values:

0	never
1	after simulation

SetCrvDesc

Sets the user defined description of a curve.

```
None VisHrm.SetCrvDesc(int curveIndex, str curveDescription)
```

ARGUMENTS

curveIndex
Curve index; first curve in table is index 1.

curveDescription
Description to set

SetDefScaleX

Sets the x-scale to be used to the graphics board.

```
None VisHrm.SetDefScaleX()
```

SetDefScaleY

Sets the y-scale to be used to the plot type.

```
None VisHrm.SetDefScaleY()
```

4.6.45 VisMagndiffplt

Overview

AddRelay
AddRelays
Clear
DoAutoScaleX
DoAutoScaleY
Refresh

AddRelay

Adds a relay to the plot and optionally sets the drawing style at the same time.

```
None VisMagndiffplt.AddRelay(DataObject relay,
                             [float colour,]
                             [float style,]
                             [float width]
                             )
```

ARGUMENTS

relay Relay to be added.

colour (optional)
The colour to be used.

style (optional)
The line style to be used.

width (optional)
The line width to be used.

AddRelays

Adds relays to the plot.

```
None VisMagndiffplt.AddRelays(list relays)
```

ARGUMENTS

relays	Relays to be added.

Clear

Removes all protection devices from the plot.

```
None VisMagndiffplt.Clear()
```

DoAutoScaleX

Scales the x-axis of the plot automatically. The function works for local x-scales only. If the x-scale is not local a warning is shown in the output window and 1 is returned by the function.

```
int VisMagndiffplt.DoAutoScaleX()
```

RETURNS

0	Automatic scaling was executed.
1	An Error occurred.

DoAutoScaleY

Scales the y-axis of the plot automatically. The function works for local y-scales only. If the x-scale is not local a warning is shown in the output window and 1 is returned by the function.

```
int VisMagndiffplt.DoAutoScaleY()
```

RETURNS

0	Automatic scaling was executed.
1	An Error occurred.

Refresh

Refreshes the plot by attempting to automatically scale both axes.

```
None VisMagndiffplt.Refresh()
```

4.6.46 VisOcplot

Overview

AddRelay
AddRelays
Clear
DoAutoScaleX
DoAutoScaleY
Refresh

AddRelay

Adds one or more relays to the plot. The version for one relay can also set the drawing style at the same time.

```
None VisOcplot.AddRelay(DataObject relay,
                        [float colour,]
                        [float style,]
                        [float width]
                        )
```

ARGUMENTS

relay Protection device (ElmRelay or RelFuse) to be added.

colour (optional)
The colour to be used.

style (optional)
The line style to be used.

width (optional)
The line width to be used.

AddRelays

Adds relays to the plot.

```
None VisOcplot.AddRelays(list relay)
```

ARGUMENTS

relays Protection devices (ElmRelay or RelFuse) to be added.

Clear

Removes all protection devices from the plot.

```
None VisOcplot.Clear()
```

DoAutoScaleX

Scales the x-axis of the plot automatically. The function works for local x-scales only. If the x-scale is not local a warning is shown in the output window and 1 is returned by the function.

```
int VisOcplot.DoAutoScaleX()
```

RETURNS

0	Automatic scaling was executed.
1	An Error occurred.

DoAutoScaleY

Scales the y-axis of the plot automatically. The function works for local y-scales only. If the x-scale is not local a warning is shown in the output window and 1 is returned by the function.

```
int VisOcplot.DoAutoScaleY()
```

RETURNS

0	Automatic scaling was executed.
1	An Error occurred.

Refresh

Refreshes the plot by attempting to automatically scale both axes.

```
None VisOcplot.Refresh()
```

4.6.47 VisPath

Overview

Clear
DoAutoScaleX
DoAutoScaleY
SetAdaptX
SetAdaptY
SetScaleX
SetScaleY

Clear

Removes all curves by clearing table named 'Variables' on page 'Curves'.

```
None VisPath.Clear()
```

DoAutoScaleX

Scales x-axis automatically.

```
int VisPath.DoAutoScaleX()
```

RETURNS

Always 0

DoAutoScaleY

Scales y-axis automatically.

```
int VisPath.DoAutoScaleY()
```

RETURNS

Always 0

SetAdaptX

Sets the Adapt Scale option of the x-scale.

```
None VisPath.SetAdaptX(int mode)
```

ARGUMENTS

mode Possible values:

0	off
1	on

SetAdaptY

Sets the Adapt Scale option of the x-scale.

```
None VisPath.SetAdaptY(int mode)
```

ARGUMENTS

mode Possible values:

0	off
1	on

SetScaleX

Sets x-axis scale.

```
None VisPath.SetScaleX(float min,
                       float max,
                       )
```

ARGUMENTS

min Minimum of x-scale.

max Maximum of x-scale.

SetScaleY

Sets y-axis scale limits.

```
None VisPath.SetScaleY(float min,
                       float max,
                       [int log]
                       )
```

ARGUMENTS

min Minimum of y-scale.

max Maximum of y-scale.

log (optional)
Possible values:

0	linear
1	logarithmic

4.6.48 VisPcompdiffplt

Overview

AddRelay
AddRelays
CentreOrigin
Clear
DoAutoScaleX
DoAutoScaleY

AddRelay

Adds a relay to the plot and sets optionally the drawing style.

```
None VisPcompdiffplt.AddRelay(DataObject relay,
                              [float colour,]
                              [float style,]
                              [float width])
```

ARGUMENTS

relay The protection device (ElmRelay or RelFuse) to be added.

colour (optional)
The colour to be used.

style (optional)
The line style to be used.

width (optional)
The line width to be used.

AddRelays

Adds relays to the plot.

```
None VisPcompdiffplt.AddRelays(list relays)
```

ARGUMENTS

relays The protection devices (ElmRelay or RelFuse) to be added.

CentreOrigin

Centre the origin of the plot

```
None VisPcompdiffplt.CentreOrigin()
```

Clear

Removes all protection devices from the plot.

```
None VisPcompdiffplt.Clear()
```

DoAutoScaleX

Scales the x-axis of the plot automatically. The function works for local x-scales only. If the x-scale is not local a warning is shown in the output window and 1 is returned by the function.

```
int VisPcompdiffplt.DoAutoScaleX()
```

RETURNS

0	Automatic scaling was executed.
1	An Error occurred.

DoAutoScaleY

Scales the y-axis of the plot automatically. The function works for local y-scales only. If the x-scale is not local a warning is shown in the output window and 1 is returned by the function.

```
int VisPcompdiffplt.DoAutoScaleY()
```

RETURNS

0	Automatic scaling was executed.
1	An Error occurred.

4.6.49 VisPlot

Overview

AddResVars
AddVars
Clear
DoAutoScaleX
DoAutoScaleY
GetScaleObjX
GetScaleObjY
SetAdaptX
SetAdaptY
SetAutoScaleX
SetAutoScaleY
SetCrvDesc
SetDefScaleX
SetDefScaleY

SetScaleX
SetScaleY
SetXVar

AddResVars

Appends variables to the plot. Variables which are already in the plot are not added.

```
None VisPlot.AddResVars(DataObject elmRes
                        DataObject element,
                        str varname
                        )
```

ARGUMENTS

elmRes	Result object, classanme ElmRes
elememt	Element to add
varname	Variable name

AddVars

Appends variables to the plot. Variables which are already in the plot are not added.

```
None VisPlot.AddVars(DataObject element,
                     str varname
                     )
```

ARGUMENTS

elememt	Element to add
varname	Variable name

Clear

Removes all curves from plot.

```
None VisPlot.Clear()
```

DoAutoScaleX

Scales x-axis automatically.

```
int VisPlot.DoAutoScaleX()
```

RETURNS

0	Ok, call to DoAutoScaleX() was successfull
1	Failed, because the x-scale is not local

DoAutoScaleY

Scales y-axis automatically.

```
int VisPlot.DoAutoScaleY()
```

Returns

0	Ok, call to DoAutoScaleY() was successfull
1	Failed, because the y-scale is not local

GetScaleObjX

Gets the object used for scaling the x-axis.

```
DataObject VisPlot.GetScaleObjX()
```

Returns

this object In case that 'Use local Axis' is set to 'Local'.

the virtual instrument panel In case that 'Use local axis' is set to 'Current Page'.

the graphics board In case that 'Use local axis' is set to 'Graphics Board'.

GetScaleObjY

Gets the object used for scaling the y-axis.

```
DataObject VisPlot.GetScaleObjY()
```

Returns

this object In case that 'Use local Axis' is enabled.

the plot type In case that 'Use local axis' is disabled.

SetAdaptX

Sets the Adapt Scale option of the local x-scale.

```
None VisPlot.SetAdaptX(int mode,
                       [float trigger]
                       )
```

Arguments

mode Possible values:

0	off
1	on

trigger (optional)
Trigger value, unused if mode is off or empty

SetAdaptY

Sets the Adapt Scale option of the local y-scale.

```
None VisPlot.SetAdaptY(int mode,
                       [float offset]
                       )
```

ARGUMENTS

mode Possible values:

0	off
1	on

offset (optional)
Offset value, unused if mode is off or empty

SetAutoScaleX

Sets Auto Scale setting of the x-scale. The scale is automatic set to local, in case that the plot is using the scale of the graphics board or the virtual instrument panel.

```
None VisPlot.SetAutoScaleX(int mode)
```

ARGUMENTS

mode Possible values:

0	never
1	after simulation
2	during simulation

SetAutoScaleY

Sets Auto Scale setting of the y-scale. The scale is automatic set to local, in case that the plot is using the scale of the plot type.

```
None VisPlot.SetAutoScaleY(int mode)
```

ARGUMENTS

mode Possible values:

0	never
1	after simulation
2	during simulation

SetCrvDesc

Sets the user defined description of a curve.

```
None VisPlot.SetCrvDesc(int curveIndex, str curveDescription)
```

ARGUMENTS

curveIndex
Curve index; first curve in table is index 1.

curveDescription
Description to set

SetDefScaleX

Sets the x-scale to be used to the graphics board.

```
None VisPlot.SetDefScaleX()
```

SetDefScaleY

Sets the y-scale to be used to the plot type.

```
None VisPlot.SetDefScaleY()
```

SetScaleX

Sets the local x-axis scale. A function call without arguments sets the Auto Scale setting to On without changing the scale itself.

```
None VisPlot.SetScaleX()
None VisPlot.SetScaleX(float min,
                       float max,
                       [int log]
                       )
```

ARGUMENTS

min (optional)
: Minimum of x-scale.

max (optional)
: Maximum of x-scale.

log (optional)
: Possible values:

0	linear
1	logarithmic

SetScaleY

Sets the local y-axis scale. A function call without arguments sets the Auto Scale setting to On without changing the scale itself.

```
None VisPlot.SetScaleY()
None VisPlot.SetScaleY(float min,
                       float max,
                       [int log]
                       )
```

ARGUMENTS

min (optional)
: Minimum of y-scale.

max (optional)
: Maximum of y-scale.

log (optional)
: Possible values:

0	linear
1	logarithmic

SetXVar

Sets the local x-axis variable. If The default x-axis variable (time) is set if no argument is passed.

```
None VisPlot.SetXVar()
None VisPlot.SetXVar(DataObject obj,]
                     str varname
                     )
```

ARGUMENTS

obj (optional)
x-axis object

varname (optional)
variable of obj

4.6.50 VisPlot2

Overview

AddResVars
AddVars
Clear
DoAutoScaleX
DoAutoScaleY
DoAutoScaleY2
GetScaleObjX
GetScaleObjY
SetAdaptX
SetAdaptY
SetAutoScaleX
SetAutoScaleY
SetCrvDesc
SetDefScaleX
SetDefScaleY
SetScaleX
SetScaleY
SetXVar
ShowY2

AddResVars

Appends variables to the plot. Variables which are already in the plot are not added.

```
None VisPlot2.AddResVars(DataObject elmRes
                         DataObject element,
                         str varname,
                         [int y2]
                         )
```

ARGUMENTS

elmRes	Result object, classanme ElmRes
elememt	Element to add
varname	Variable name

y2 (optional)
Possible values:

1	y1-axis, default value
2	y2 axis

AddVars

Appends variables to the plot. Variables which are already in the plot are not added.

```
None VisPlot2.AddVars(DataObject element,
                      str varname,
                      [int y2]
                      )
```

ARGUMENTS

elememt	Element to add
varname	Variable name

y2 (optional)
Possible values:

1	y1-axis, default value
2	y2 axis

Clear

Removes variables from plot

```
None VisPlot2.Clear([int y2])
```

ARGUMENTS

y2 (optional)
Possible values:

1	y1-axis, default value
2	y2 axis

DoAutoScaleX

Scales x-axis automatically.

```
int VisPlot2.DoAutoScaleX()
```

RETURNS

0	Ok, call to DoAutoScaleX() was successfull
1	Failed, because the x-scale is not local

DoAutoScaleY

Scales y1-axis automatically.

```
int VisPlot2.DoAutoScaleY()
```

RETURNS

0	Ok, call to DoAutoScaleY() was successfull
1	Failed, because the y-scale is not local

DoAutoScaleY2

Scales y2-axis automatically.

```
int VisPlot2.DoAutoScaleY2()
```

RETURNS

0	Ok, call to DoAutoScaleY() was successfull
1	Failed, because the y-scale is not local

GetScaleObjX

Gets the object used for scaling the x-axis.

```
DataObject VisPlot2.GetScaleObjX()
```

RETURNS

this object In case that 'Use local Axis' is set to 'Local'.

the virtual instrument panel In case that 'Use local axis' is set to 'Current Page'.

the graphics board In case that 'Use local axis' is set to 'Graphics Board'.

GetScaleObjY

Returns used object defining y-scale. The returned object is either the plot itself or the plot type (IntPlot).

```
DataObject VisPlot2.GetScaleObjY ([int y2])
```

RETURNS

this object In case that 'Use local Axis' is enabled.

the plot type In case that 'Use local axis' is disabled.

SetAdaptX

Sets the Adapt Scale option of the local x-scale.

```
None VisPlot2.SetAdaptX(int mode,
                        [float trigger]
                        )
```

ARGUMENTS

mode Possible values:

0	off
1	on

trigger (optional)
Trigger value, unused if mode is off or empty

SetAdaptY

Sets the Adapt Scale option of the local y-scale.

```
None VisPlot2.SetAdaptY(int mode,
                        [float offset,]
                        [int y2]
                        )
```

ARGUMENTS

mode Possible values:

0	off
1	on

offset (optional)
Offset value, unused if mode is off or empty

y2 (optional)
Possible values:

1	y1-axis, default value
2	y2 axis

SetAutoScaleX

Sets Auto Scale setting of the x-scale. The scale is automatic set to local, in case that the plot is using the scale of the graphics board or the virtual instrument panel.

```
None VisPlot2.SetAutoScaleX(int mode)
```

ARGUMENTS

mode Possible values:

0	never
1	after simulation
2	during simulation

SetAutoScaleY

Sets automatic scaling mode of the y-scale. The axis given in the second argument is automatically set to local.

```
None VisPlot2.SetAutoScaleY (int mode,
                             [int y2]
                             )
```

ARGUMENTS

mode Possible values:

0	never
1	after simulation
2	during simulation

y2 (optional)
Possible values:

1	y1-axis, default value
2	y2 axis

SetCrvDesc

Sets the user defined description of a curve.

```
None VisPlot2.SetCrvDesc(int curveIndex, str curveDescription)
```

ARGUMENTS

curveIndex
Curve index; first curve in table is index 1.

curveDescription
Description to set

SetDefScaleX

Sets the x-scale to be used to the graphics board.

```
None VisPlot2.SetDefScaleX()
```

SetDefScaleY

Sets the y-scale to be used to the plot type.

```
None VisPlot2.SetDefScaleY([int y2])
```

ARGUMENTS

y2 (optional)
Possible values:

1	y1-axis, default value
2	y2 axis

SetScaleX

Sets the local x-axis scale. A function call without arguments sets the Auto Scale setting to On without changing the scale itself.

```
None VisPlot.SetScaleX()
None VisPlot.SetScaleX(float min,
                      float max,
                      [int log]
                      )
```

ARGUMENTS

min (optional)
Minimum of x-scale.

max (optional)
Maximum of x-scale.

log (optional)
Possible values:

0	linear
1	logarithmic

SetScaleY

Sets scale of y-axis. Calling the function without any argument sets the Auto Scale option for the y axis (both share the same setting) to On.

```
None VisPlot2.SetScaleY()
None VisPlot2.SetScaleY(float min,
                        float max,
                        [int log,]
                        [int Y2]
                        )
```

ARGUMENTS

min (optional)
Minimum of y-scale.

max (optional)
Maximum of y-scale.

log (optional)
Possible values:

0	linear
1	logarithmic

y2 (optional)
Possible values:

1	y1-axis, default value
2	y2 axis

SetXVar

Sets the local x-axis variable. If The default x-axis variable (time) is set if no argument is passed.

```
None VisPlot.SetXVar()
None VisPlot.SetXVar(DataObject obj,]
                     str varname
                     )
```

ARGUMENTS

obj (optional)
x-axis object

varname (optional)
variable of obj

ShowY2

Enables or disables the y2 axis.

```
None VisPlot2.ShowY2([int show])
```

ARGUMENTS

show (optional)
Possible values:

0	hide y2 axis
1	show y2 axis (default)

4.6.51 VisPlottz

Overview

AddRelay
AddRelays
Clear
DoAutoScaleX
DoAutoScaleY

AddRelay

Adds a relay to the plot and sets optionally the drawing style.

```
None VisPlottz.AddRelay(DataObject relay,
                        [float colour,]
                        [float style,]
                        [float width])
```

ARGUMENTS

relay The protection device (ElmRelay or RelFuse) to be added.

colour (optional)
The colour to be used.

style (optional)
The line style to be used.

width (optional)
The line width to be used.

AddRelays

Adds relays to the plot.

```
None VisPlottz.AddRelays(list relays)
```

ARGUMENTS

relays	The protection devices (ElmRelay or RelFuse) to be added.

Clear

Removes all protection devices from the plot.

```
None VisPlottz.Clear()
```

DoAutoScaleX

Scales the x-axis of the plot automatically. The function works for local x-scales only. If the x-scale is not local a warning is shown in the output window and 1 is returned by the function.

```
int VisPlottz.DoAutoScaleX()
```

RETURNS

0	Automatic scaling was executed.
1	An Error occurred.

DoAutoScaleY

Scales the y-axis of the plot automatically. The function works for local y-scales only. If the x-scale is not local a warning is shown in the output window and 1 is returned by the function.

```
int VisPlottz.DoAutoScaleY()
```

RETURNS

0	Automatic scaling was executed.
1	An Error occurred.

4.6.52 VisVec

Overview

CentreOrigin

CentreOrigin

Centre the origin of the plot

```
None VisVec.CentreOrigin()
```

4.6.53 VisXyplot

Overview

Clear
DoAutoScaleX
DoAutoScaleY
SetCrvDescX
SetCrvDescY

Clear

Removes all curves from plot.

```
None VisXyplot.Clear()
```

DoAutoScaleX

Scales all used x-axes automatically.

```
int VisXyplot.DoAutoScaleX()
```

RETURNS

0	Ok, call to DoAutoScaleX() was successfull
1	Failed, because the x-scales are not local

DoAutoScaleY

Scales all used y-axes automatically.

```
int VisXyplot.DoAutoScaleY()
```

RETURNS

0	Ok, call to DoAutoScaleX() was successfull
1	Failed, because the x-scales are not local

SetCrvDescX

Sets the user defined description of a curve for the x-variable.

```
None VisXyplot.SetCrvDescX(int curveIndex, str curveDescription)
```

ARGUMENTS

curveIndex
Curve index; first curve in table is index 1.

curveDescription
Description to set

SetCrvDescY

Sets the user defined description of a curve for the y-variable.

```
None VisXyplot.SetCrvDescY(int curveIndex, str curveDescription)
```

ARGUMENTS

curveIndex
Curve index; first curve in table is index 1.

curveDescription
Description to set

Index