



POWERFACTORY

PowerFactory 2021

DPL Function Reference

PF2021

POWER SYSTEM SOLUTIONS

MADE IN GERMANY

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:
<https://www.digsilent.de>

Copyright © 2021 DIgSILENT GmbH

All rights reserved. No part of this publication may be reproduced or distributed in any form without written permission of DIgSILENT GmbH.

February 3, 2021
Revision 4

Contents

1 General Description	1
2 Global Functions	2
2.1 File System	35
2.2 Date/Time	44
2.3 Dialogue Boxes	50
2.4 Environment	57
2.5 Mathematics	64
2.6 MS Office	82
2.6.1 MS Access	82
2.6.2 MS Excel	91
2.7 Multibyte Encoded String	114
2.8 Output Window	117
2.9 String	122
3 Object Methods	132
3.1 General Methods	132
3.2 Network Elements	169
3.2.1 ElmArea	169
3.2.2 ElmAsm	172
3.2.3 ElmAsmsc	175
3.2.4 ElmBbone	177
3.2.5 ElmBmu	182
3.2.6 ElmBoundary	183
3.2.7 ElmBranch	185
3.2.8 ElmCabsys	186
3.2.9 ElmComp	187
3.2.10 ElmCoup	188
3.2.11 ElmDsl	190
3.2.12 ElmFeeder	191
3.2.13 ElmFile	195
3.2.14 ElmFilter	196

3.2.15 ElmGenstat	197
3.2.16 ElmGndswt	201
3.2.17 ElmLne	203
3.2.18 ElmLnsec	212
3.2.19 ElmNec	213
3.2.20 ElmNet	213
3.2.21 ElmPvsys	215
3.2.22 ElmRelay	219
3.2.23 ElmRes	224
3.2.24 ElmShnt	246
3.2.25 ElmStactrl	246
3.2.26 ElmSubstat	249
3.2.27 ElmSvs	255
3.2.28 ElmSym	256
3.2.29 ElmTerm	261
3.2.30 ElmTr2	268
3.2.31 ElmTr3	272
3.2.32 ElmTr4	276
3.2.33 ElmTrfstat	281
3.2.34 ElmVac	286
3.2.35 ElmVoltreg	286
3.2.36 ElmXnet	288
3.2.37 ElmZone	290
3.3 Station Elements	293
3.3.1 StaCt	293
3.3.2 StaCubic	294
3.3.3 StaExtbrkmea	301
3.3.4 StaExtcmdmea	306
3.3.5 StaExtdatmea	311
3.3.6 StaExtfmea	316
3.3.7 StaExtfuelmea	321
3.3.8 StaExtimea	326
3.3.9 StaExtpfmea	331
3.3.10 StaExtpmea	336
3.3.11 StaExtqmea	341
3.3.12 StaExtsmea	346
3.3.13 StaExttapmea	350
3.3.14 StaExtv3mea	356
3.3.15 StaExtvmea	361
3.3.16 StaSwitch	366

3.4 Commands	368
3.4.1 ComAddlabel	368
3.4.2 ComAddon	369
3.4.3 ComAmpacity	380
3.4.4 ComAuditlog	381
3.4.5 ComBoundary	381
3.4.6 ComCapo	381
3.4.7 ComCheck	385
3.4.8 ComCimdbexp	385
3.4.9 ComCimdbimp	385
3.4.10 ComCimvalidate	386
3.4.11 ComConreq	390
3.4.12 ComContingency	390
3.4.13 ComCoordreport	396
3.4.14 ComDiff	406
3.4.15 ComDllmanager	406
3.4.16 ComDpl	407
3.4.17 ComFlickermeter	413
3.4.18 ComGenrelinc	414
3.4.19 ComGridtocim	415
3.4.20 ComHostcap	416
3.4.21 ComImport	418
3.4.22 ComInc	419
3.4.23 ComLdf	419
3.4.24 ComLink	422
3.4.25 ComMerge	425
3.4.26 ComMot	430
3.4.27 ComNmink	430
3.4.28 ComOmr	432
3.4.29 ComOpc	435
3.4.30 ComOutage	436
3.4.31 ComPfdimport	439
3.4.32 ComPrjconnector	439
3.4.33 ComProtgraphic	440
3.4.34 ComPvcurves	440
3.4.35 ComPython	440
3.4.36 ComRed	444
3.4.37 ComRel3	445
3.4.38 ComRelpost	447
3.4.39 ComRelreport	449

3.4.40 ComRes	449
3.4.41 ComShc	450
3.4.42 ComShctrace	453
3.4.43 ComSim	456
3.4.44 ComSimoutage	458
3.4.45 ComSvgexport	463
3.4.46 ComSvgimport	464
3.4.47 ComTablereport	465
3.4.48 ComTasks	506
3.4.49 ComTececo	518
3.4.50 ComTransfer	518
3.4.51 ComUcte	520
3.4.52 ComUcteexp	520
3.4.53 ComWktimp	527
3.5 Settings	528
3.5.1 SetCluster	528
3.5.2 SetColscheme	530
3.5.3 SetDataext	536
3.5.4 SetDeskpage	538
3.5.5 SetDesktop	539
3.5.6 SetDistrstate	547
3.5.7 SetFilt	548
3.5.8 SetLevelvis	548
3.5.9 SetParalman	550
3.5.10 SetPath	551
3.5.11 SetSelect	554
3.5.12 SetTboxconfig	560
3.5.13 SetTime	561
3.5.14 SetUser	563
3.5.15 SetVipage	564
3.6 Others	572
3.6.1 BlkDef	572
3.6.2 BlkSig	574
3.6.3 ChaVecfile	575
3.6.4 CimArchive	575
3.6.5 CimModel	575
3.6.6 CimObject	580
3.6.7 GrpPage	584
3.6.8 IntAddonvars	590
3.6.9 IntCase	594

3.6.10 IntComtrade	596
3.6.11 IntComtradeset	605
3.6.12 IntDataset	614
3.6.13 IntDocument	615
3.6.14 IntDplmap	616
3.6.15 IntDplvec	623
3.6.16 IntEvt	626
3.6.17 IntExtaccess	627
3.6.18 IntGate	628
3.6.19 IntGrf	628
3.6.20 IntGrfgroup	628
3.6.21 IntGrflayer	630
3.6.22 IntGrfnet	632
3.6.23 IntlIcon	633
3.6.24 IntLibrary	634
3.6.25 IntMat	635
3.6.26 IntMon	643
3.6.27 IntOutage	646
3.6.28 IntPlannedout	649
3.6.29 IntPlot	649
3.6.30 IntPrj	652
3.6.31 IntPrjfolder	663
3.6.32 IntQlim	665
3.6.33 IntRas	666
3.6.34 IntRunarrange	667
3.6.35 IntScenario	668
3.6.36 IntScsnsched	671
3.6.37 IntScheme	673
3.6.38 IntSscheduler	674
3.6.39 IntSstage	675
3.6.40 IntSubset	678
3.6.41 IntThrating	680
3.6.42 IntUrl	681
3.6.43 IntUser	682
3.6.44 IntUserman	682
3.6.45 IntVec	684
3.6.46 IntVecobj	687
3.6.47 IntVersion	689
3.6.48 IntViewbookmark	691
3.6.49 PltDataseries	691

3.6.50 PltLinebarplot	693
3.6.51 RelZpol	698
3.6.52 ScnFreq	698
3.6.53 ScnFrt	701
3.6.54 ScnSpeed	703
3.6.55 ScnSync	705
3.6.56 ScnVar	707
3.6.57 ScnVolt	709
3.6.58 StoMaint	711
3.6.59 TypAsmo	711
3.6.60 TypCtcore	712
3.6.61 TypLne	713
3.6.62 TypQdsI	713
3.6.63 TypTr2	715
3.6.64 VisBdia	716
3.6.65 VisDraw	719
3.6.66 VisHrm	722
3.6.67 VisMagndiffplt	728
3.6.68 VisOcplot	729
3.6.69 VisPath	731
3.6.70 VisPcompdifffplt	735
3.6.71 VisPlot	736
3.6.72 VisPlot2	748
3.6.73 VisPlottz	762
3.6.74 VisVec	764
3.6.75 VisVecres	764
3.6.76 VisXyplot	764
4 Set Routines	768
Index	776

1 General Description

The DLgSILENT Programming Language (DPL) provides a large set of build-in functions. For overview and understanding they can be grouped into the following categories

1. global functions,
2. general methods,
3. specialised methods,
4. set routines.

The “global functions” are a kind of static functions that can be called everywhere in the script.

The term ‘methods’ is used for functions that can be called on an object. Generally it is distinguished between methods that are available for objects of different classes and those available for distinct classes only. The first group is called “general methods”, the second “specialised methods” with reference to its class.

The last group of “set routines” is formed by functions that are available for working with DPL own data type ‘set’.

This document comes as a reference manual. The following chapters describe the syntax and meaning of all available functions.

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

Functions which are marked with an asterisk (*) are available in

1. QDSL (see [here](#)),
2. Configuration scripts for initialisation of RMS/EMT simulation (see [here](#)).

2 Global Functions

Overview

ActivateProject
ClearCommands
ClearRecycleBin
CommitTransaction
CreateFaultCase
CreateProject
Delete
DeleteUntouchedObjects
ExecuteCmd
exit*
GetActiveCalculationStr*
GetActiveNetworkVariations*
GetActiveProject*
GetActiveScenario*
GetActiveScenarioScheduler*
GetActiveStages*
GetActiveStudyCase*
GetAllUsers*
GetBorderCubicles*
GetBrowserSelection
GetCalcRelevantObjects*
GetClassDescription*
GetCurrentDiagram
GetCurrentSelection
GetCurrentUser*
GetCurrentZoomScaleLevel
GetDataFolder*
GetDescription*
GetDiagramSelection
GetFlowOrientation*
GetFromStudyCase*
GetGlobalLibrary*
GetGraphicsBoard*
GetLanguage*
GetLastCmd
GetLocalLibrary*
GetMem*
GetPageLen*
GetPFileVersion*
GetProjectFolder*
GetRecordingStage*
GetResData

GetResDesc
GetResObj
GetResUnit
GetResVar
GetSettings*
GetSummaryGrid*
GetUserManager*
HttpGet
ImportDz
ImportSnapshot
IsLdfValid*
IsRmsValid*
IsScenarioAttribute*
IsShcValid*
IsSimValid*
LoadProfile
LoadResData
MarkInGraphics
OutputFlexibleData*
PostCommand
PrepForUntouchedDelete
Rebuild*
ReleaseResData
ReloadProfile
ResetCalculation
ResFirstValidObject
ResFirstValidObjectVar
ResFirstValidVar
ResGetMax
ResGetMin
ResIndex
ResNextValidObject
ResNextValidObjectVar
ResNextValidVar
ResNval
ResNvars
ResSortToVar
SaveAsScenario
SearchObjectByForeignKey*
SelectToolbox
SetShowAllUsers
Sleep*
SplitLine
StatFileGetXrange
StatFileResetXrange
StatFileSetXrange

ActivateProject

Activates a project with its name.

```
int ActivateProject(string 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.

ClearCommands

Clears the command pipe of the input window. It does not interrupt currently running command.

```
void ClearCommands()
```

ClearRecycleBin

Clears the recycle bin of currently logged in user.

```
void ClearRecycleBin()
```

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.

```
void CommitTransaction()
```

DEPRECATED NAMES

CommitTx_

CreateFaultCase

Create fault cases from the given elements.

```
int 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 elements 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.

```
object CreateProject(string projectName,  
                     string gridName,  
                     [object 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.

Delete

Deletes an object or a set of objects from the database. The objects are not destroyed but are moved to the recycle bin.

```
void Delete(object object)  
void Delete(set objects)
```

ARGUMENTS*object*

Object to delete.

objects

Set of objects to delete.

SEE ALSO

[object.CreateObject\(\)](#), [object.AddCopy\(\)](#), [object.PasteCopy\(\)](#), [object.Move\(\)](#)

EXAMPLE

The following example removes all "Dummy" fuses from the network. The 'DummyType' variable is a local variable in the DPL script. A set of objects to delete is created first and then that set is deleted. This has the advantage that one single entry in the recycle bin is created which contains all deleted fuses. Manually restoring ('undelete') the deleted fuses, in case of a mistake, can then be done using a single restore command.

```
object obj;  
set fuses, toDelete;  
fuses = GetCalcRelevantObjects();  
obj = fuses.FirstFilt('*.RelFuse');  
while (obj) {
```

```
if (obj:type_id=DummyType) {  
    toDelete.Add(obj);  
}  
obj = fuses.NextFilt();  
}  
Delete(toDelete);
```

DeleteUntouchedObjects

Delete all objects stored in the grid (or stored in set) which are not modified. Requires call of PrepForUntouchedDelete() first, see [PrepForUntouchedDelete\(\)](#). Hint: function should only be used when a variation is active. For details please contact support.

```
int DeleteUntouchedObjects(object grid)  
int DeleteUntouchedObjects(set grids)
```

EXAMPLE

```
PrepForUntouchedDelete(grid);  
  
! do here modifications e.g. ... run UCTE import  
  
DeleteUntouchedObjects(grid);
```

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.

```
void ExecuteCmd(string command)
```

ARGUMENTS

command

The command string

DEPRECATED NAMES

Exe

EXAMPLE

The following script executed a load flow calculation with default options

```
ExecuteCmd('ldf');
```

exit*

Terminates a DPL script immediately. If called within a subscript, only the subscript itself will be terminated and code is returned by 'subscript.Execute()' in the main script.

```
void exit([int code])
```

ARGUMENTS

code The integer returned by 'subscript.Execute()' in the main script, when exit() is called within a subscript. The default value is 1.

SEE ALSO

[ComDpl.Execute\(\)](#)

EXAMPLE

```
int number;
int sum;

! sums up all entered numbers
while(1) {
    input(number, 'Enter a number please (0 to stop):');
    if (number == 0) {
        printf('Sum: %d', sum);
        exit(); ! terminate script here
    }
    sum += number;
}
```

GetActiveCalculationStr*

Gets “calculation string” of currently valid calculation.

```
string 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 Contingency 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 arclash
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.

```
set GetActiveNetworkVariations()
```

RETURNS

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

EXAMPLE

The following example returns the currently active variations:

```
object variation;
set variations;
variations = GetActiveNetworkVariations();
for(variation = variations.First();variation;variation = variations.Next()) {
    variation.ShowFullName();
}
```

GetActiveProject*

This function returns the currently active project.

```
object GetActiveProject()
```

DEPRECATED NAMES

ActiveProject

EXAMPLE

The following example prints the currently active project to output window:

```
object project;
project = GetActiveProject();
printf('Currently active project: %o', project);
```

RETURNS

Returns currently active *IntPrj* object or NULL in case of no project being currently active.

GetActiveScenario*

Returns the currently active scenario. NULL is returned if there is no active scenario.

```
object GetActiveScenario()
```

EXAMPLE

The following example prints the currently active scenario to output window:

```
object scenario;
scenario = GetActiveScenario();
printf('Active scenario: %o', scenario);
```

RETURNS

Returns currently active *IntScenario* object or NULL in case of no scenario being currently active.

GetActiveScenarioScheduler*

Returns currently active scenario scheduler.

```
object GetActiveScenarioScheduler()
```

RETURNS

Returns currently active *IntScensched* object or NULL in case of no scheduler being currently active.

GetActiveStages*

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

```
set GetActiveStages([object 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.

EXAMPLE

The following example returns the currently active recording stages:

```
object stage;
set stages;
stages = GetActiveStages();
for(stage = stages.First(); stage; stage = stages.Next()) {
    stage.ShowFullName();
}
```

GetActiveStudyCase*

Returns the active Study Case. NULL is returned if there is no active study case.

```
object GetActiveStudyCase()
```

RETURNS

The active study case (*IntCase* object) or NULL.

DEPRECATED NAMES

ActiveCase

EXAMPLE

The following example writes the active study case to the output window:

```
object case;
case = GetActiveStudyCase();
printf('Active study case: %o', case);
```

RETURNS

Returns currently active *IntCase* object or NULL in case of no study case being currently active.

GetAllUsers*

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

```
set GetAllUsers([forceReload = 0])
```

```
list GetAllUsers([forceReload = 0])
```

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.

```
set GetBorderCubicles(object 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).

```
set GetBrowserSelection()
```

RETURNS

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

SEE ALSO

[GetCurrentSelection\(\)](#), [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).

In contrast to [object.IsCalcRelevant\(\)](#) it does not return objects of the active study case e.g. simulation events (Evt*).

```
set GetCalcRelevantObjects([string 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.

DEPRECATED NAMES**AllRelevant****SEE ALSO**[object.IsCalcRelevant\(\)](#)**EXAMPLE**

The following example prints the names of all calculation relevant loads

```
set loads;
object load;

loads = GetCalcRelevantObjects('*.ElmLod');
for(load = loads.First(); load; load = loads.Next()) {
    load.ShowFullName();
}
```

GetClassDescription*

Returns a description for a PowerFactory class.

```
string GetClassDescription(string name)
```

ARGUMENTS

name Name of a **PowerFactory** class

RETURNS

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

GetCurrentDiagram

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

```
object GetCurrentDiagram()
```

GetCurrentSelection

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

```
set GetCurrentSelection()
```

RETURNS

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

SEE ALSO[GetBrowserSelection\(\)](#), [GetDiagramSelection\(\)](#)**GetCurrentUser***

Returns the PowerFactory user of current session.

```
object GetCurrentUser()
```

EXAMPLE

The following example prints the user of current session to output window:

```
object user;
user = GetCurrentUser();
printf('Current user: %o', user);
```

RETURNS

Returns an *IntUser* object, never NULL.

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 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.

EXAMPLE

```
int level;
object diagram;

diagram = GetCurrentDiagram();
if (.not. diagram) {
  Error('No diagram active!');
  exit();
}

level = GetCurrentZoomScaleLevel();
if (diagram:iGPS) {
  printf('Scale level of %o: 1:%d', diagram, level);
}
else {
  printf('Zoom level of %o: %d%s', diagram, level, '%');
```

GetDataFolder*

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

```
object GetDataFolder(string classname,
                     [int iCreate])
```

ARGUMENTS

classname

Classname of the elements:

ElmBmu
ElmAra
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, NULL 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.

EXAMPLE

The following example returns the network data folder for 'ElmBoundary' elements:

```
object folder;
folder = GetDataFolder ('ElmBoundary');
folder.ShowFullName();
```

SEE ALSO

[object.IsNetworkDataFolder\(\)](#)**GetDescription***

Returns the display text or unit of measurement for an attribute.

```
string GetDescription(string class,
                      string attribute,
                      [int type = 0])
```

ARGUMENTS

class Name of the class.*attribute* Name of the attribute.*type (optional)*

Description type.

- 0 Default, long form.

- 1 Short form.

- 2 Unit.

RETURNS

Descriptive text or unit, can be empty (e.g. in case where given attribute does not exist).

EXAMPLE

```
string long, short, unit;

long = GetDescription('ElmLod', 'plini', 0);
short = GetDescription('ElmLod', 'plini', 1);
unit = GetDescription('ElmLod', 'plini', 2);

printf('ElmLod:plini means "%s (%s)" and is given in unit %s.', long, short, unit);
```

SEE ALSO

[object.lnm\(\)](#), [object.snm\(\)](#), [object.unm\(\)](#)

GetDiagramSelection

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

```
set GetDiagramSelection()
```

RETURNS

Objects marked in the “on top” diagram.

SEE ALSO

[GetCurrentSelection\(\)](#), [GetBrowserSelection\(\)](#)

GetFlowOrientation*

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

```
int 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 menu load-flow, short-circuit, transient simulation, etc.,

```
object GetFromStudyCase(string className)
```

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.

DEPRECATED NAMES

[GetCaseObject](#), [GetCaseCommand](#)

EXAMPLE

The following example uses the default SetTime object to change the calculation time, and then executes the load flow command with the name 'Unbalanced'.

```
object time, ldf;
time = GetFromStudyCase('SetTime');
time:hourofyear = 1234;
ldf = GetFromStudyCase('Unbalanced.ComLdf');
ldf.Execute();
```

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.

```
object GetGlobalLib([string ClassName])
```

ARGUMENTS*ClassName (optional)*

The classname of the objects for which the library folder is sought

RETURNS

The libary folder

SEE ALSO

[GetLocalLibrary\(\)](#)

DEPRECATED NAMES

[GetGlobalLib](#)

EXAMPLE

The following example shows the contents of the global library for line types.

```
object lib, obj;
set contents;
lib = GetGlobalLibrary('TypLine');
contents = lib.GetContents();
obj = contents.First();
while (obj) {
    obj.ShowFullName();
    obj = contents.Next();
}
```

GetGraphicsBoard*

Returns the currently active Graphics Board.

```
object GetGraphicsBoard()
```

RETURNS

The graphics board object

DEPRECATED NAMES

GetGraphBoard

EXAMPLE

The following example looks for an opened Graphics Board and sets its default results to the results object named 'Results'.

```
object graphBoard;
! Look for opened graphics board.
graphBoard=GetGraphicsBoard();
if (graphBoard) {
    ! Set default results object
    graphBoard.SetResults(Results);
}
```

GetLanguage*

Returns a string for the current program language setting.

```
string GetLanguage()
```

RETURNS

en	English
de	German
es	Spanish
fr	French
ru	Russian
cn	Simplified Chinese
tr	Turkish

GetLastCmd

Returns the last executed command of the given class. If no class is given, the last executed command is returned.

```
GetLastCmd( [string class] )
```

ARGUMENTS

class (optional)
command class

RETURNS

The last executed command.

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.

```
object GetLocalLibrary([string ClassName])
```

ARGUMENTS

ClassName (optional)

The classname of the objects for which the library folder is sought

RETURNS

The libary folder

SEE ALSO

[GetGlobalLibrary\(\)](#)

DEPRECATED NAMES

GetLocalLib

EXAMPLE

The following example shows the contents of the local library for line types.

```
object lib, obj;
set contents;
lib = GetLocalLib('TypLne');
contents = lib.GetContents();
obj = contents.First();
while (obj) {
    obj.ShowFullName();
    obj = contents.Next();
}
```

GetMem*

Allows to trace memory consumption (current working set size).

```
int GetMem([int calculateDelta=0],
           [int inMegaByte=0]
          )
```

ARGUMENTS

calculateDelta (optional)

Measure absolute memory consumption if 0, measure the delta since the last time it was called if 1 (default: 0).

inMegaByte (optional)

Returns consumption in byte if 0, in megabyte if 1 (default: 0).

RETURNS

The current working set size or the delta since the last call.

GetPageLen*

Returns the number of lines per page according to the currently selected printer and paper size.

```
int GetPageLen([int orientation])
```

ARGUMENTS

orientation (*optional*)

- | | |
|----------|---------------------|
| 0 | (default) Landscape |
| 1 | Portrait |

RETURNS

The maximum number of lines that can be printed on a single sheet of paper in given orientation.

GetPFVersion*

Returns the internal version number of currently running PowerFactory application.

```
string GetPFVersion()
```

RETURNS

Version string of format ##.##.##, e.g. 20.0.3.1

EXAMPLE

```
string strVersion;
strVersion=GetPFVersion();
printf('%s', strVersion);
```

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.

```
object GetProjectFolder(string type, [int create])
```

ARGUMENTS

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

create Optional, default=0. Determines whether folder shall be created if it does not exist (1=create, 0=don't create).

RETURNS

An *IntPrjFolder* object. If no project is currently active or project folder of this type does not exist and 'create' is not given as 0, NULL is returned.

EXAMPLE

The following example returns the study case project folder:

```
object folder;
folder = GetProjectFolder('study');
folder.ShowFullName();
```

GetRecordingStage*

Returns the currently active recording scheme stage.

```
object GetRecordingStage()
```

RETURNS

An *IntSstage* object; NULL if there is no recording stage.

EXAMPLE

The following example returns the currently active recording stage:

```
object stage;
stage = GetRecordingStage();
stage.ShowFullName();
```

GetResData

This function is deprecated. Please use [ElmRes.GetValue\(\)](#) or [IntComtrade.GetValue\(\)](#) instead.

```
int GetResData(double& d,
               object resultObject,
               int ix,
               [int col])
```

GetResDesc

This function is deprecated. Please use [ElmRes.GetDescription\(\)](#) or [IntComtrade.GetDescription\(\)](#) instead.

```
string GetResDesc(object resultObject,
                  int col,
                  [int ishort])
```

GetResObj

This function is deprecated. Please use [ElmRes.GetObject\(\)](#) instead.

```
object GetResObj(object res,
                 int col)
```

GetResUnit

This function is deprecated. Please use [ElmRes.GetUnit\(\)](#) or [IntComtrade.GetUnit\(\)](#) instead.

```
string GetResUnit(object resultObject,  
                  int col)
```

GetResVar

This function is deprecated. Please use [ElmRes.GetVariable\(\)](#) or [IntComtrade.GetVariable\(\)](#) instead.

```
string GetResVar(object resultObject,  
                  int col)
```

GetSettings*

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

```
string GetSettings(string key)
```

ARGUMENTS

key	Return type Description
username	string Name of logged-in user
installationdir	string Fully qualified path of installation directory of PowerFactory
workingdir	string Fully qualified path of working directory of PowerFactory
tempdir	string Fully qualified path of temporary directory used by PowerFactory
sessionid	integer ID of current session
db_driver	string Name of used database driver
logfile	string Path of current log file

RETURNS

Value of settings as string

EXAMPLE

The following example demonstrated how to access those settings:

```
string s;  
int i;  
  
s = GetSettings('username');  
printf('Username: %s', s);  
  
s = GetSettings('installationdir');  
printf('InstallationDir: %s', s);  
  
s = GetSettings('workingdir');  
printf('WorkingDir: %s', s);  
  
s = GetSettings('tempdir');  
printf('TempDir: %s', s);
```

```
s = GetSettings('db_driver');
printf('DBDriver: %s', s);

i = GetSettings('sessionid');
printf('SessionID: %d', i);

s = GetSettings('logfile');
printf('Used log file: %s', s);
```

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.

```
object GetSummaryGrid()
```

RETURNS

A *ElmNet* object, or a 'NULL' object when no grids are active

DEPRECATED NAMES

SummaryGrid

EXAMPLE

The following example performs a load-flow and returns the total grid active power losses.

```
object sumGrid;
sumGrid = GetSummaryGrid();
if (sumGrid) {
    Ldf.Execute();
    printf('Active Power Losses=%f', sumGrid:c:LossP);
}
```

GetUserManager*

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

```
object GetUserManager()
```

RETURNS

The user manager object

HttpGet

This function performs a HTTP/HTTPS request and returns the response. Access to the passed URL must be allowed by the global security settings (*IntExtaccess*).

```
int HttpGet(string url,
            string& | object& content,
            [string& | object& headers,]
            [string& error])
```

ARGUMENTS

url URL to fetch. Only HTTP and HTTPS protocols are supported. Empty or malformed URLs are not allowed. Examples: `https://www.digilent.de`

content (out)

Returned content (body) of the http response. The passed argument must be a string or an instance of class `IntDplVec`.

headers (optional, out)

Headers of the http response. The passed argument must be a string or an instance of class `IntDplVec`.

errors (optional, out)

Descriptive error text, if any and available.

RETURNS

Standard http code, e.g. 200=OK, 404=Not found etc.

Single exception is a value 1 that is returned in case of a general error

EXAMPLE

```
int i, ret, size;
string errors, s;
content.Clear(); !content and headers are intended to be IntDplvec
headers.Clear(); !objects stored inside the script

ret = HttpGet('https://www.digilent.de', content, headers, errors);

printf('Return Code: %d', ret);
printf('Errors: %s', errors);
printf('Headers:');

size = headers.Size();
for(i=0; i<size; i+=1) {
    s = headers.Get(i);
    printf('%s', s);
}

printf('Content:');
size = content.Size();
for(i=0; i<size; i+=1) {
    s = content.Get(i);
    printf('%s', s);
}
```

SEE ALSO

[IntUrl.View\(\)](#)

ImportDz

Imports a DZ file. Overwrites data if it already exists.

```
int ImportDz(object target,
             string dzFilePath,
             set& importedObjects)
```

ARGUMENTS

target Target object for imported data.
dzFilePath Path to the DZ file that should be imported.
importedObjects (out) Collection of top-level objects imported.

RETURNS

0 Success
-1 Wrong file extension
nonzero Import failed

ImportSnapshot

Imports a Snapshot DZS file.

```
int ImportSnapshot (string dzsFilePath,  
                    set<Object> &importedObjects)
```

ARGUMENTS

dzsFilePath Path to the DZ file that should be imported.
importedObjects (out) Collection of top-level objects imported.

RETURNS

0 Success
-1 Wrong file extension
nonzero Import failed

IsLdfValid*

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

```
int IsLdfValid()
```

RETURNS

0 if no load-flow results are available

DEPRECATED NAMES

validLDF

EXAMPLE

The following example checks if a load-flow is available, and performs one when not.

```
int valid;  
valid = IsLdfValid();  
if (.not.valid) {  
    Ldf.Execute();  
}
```

IsRmsValid*

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

```
int IsRmsValid()
```

RETURNS

0 if no RMS simulation results are available

DEPRECATED NAMES

validRMS

EXAMPLE

The following example checks if a RMS simulation is available, and performs one when not.

```
int valid;
valid = IsRmsValid();
if (.not.valid) {
    Rms.Execute();
}
```

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 IsScenarioAttribute(string classname, string attributename)
```

ARGUMENTS

classname

Name of a **PowerFactory** class

attributename

Name of an attribute of given class

EXAMPLE

```
int t;

t = IsScenarioAttribute('ElmTerm', 'loc_name');
if (t=1) {
    printf('Attribute loc_name of class ElmTerm can be recorded by a scenario');
}
else {
    printf('Attribute loc_name of class ElmTerm cannot be recorded by a scenario');

}

t = IsScenarioAttribute('ElmTerm', 'iEarth');
if (t=1) {
    printf('Attribute iEarth of class ElmTerm can be recorded by a scenario');
}
else {
    printf('Attribute iEarth of class ElmTerm cannot be recorded by a scenario');
```

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 IsShcValid()
```

RETURNS

0 if no short-circuit results are available

DEPRECATED NAMES

validSHC

EXAMPLE

The following example checks if a short-circuit result is available, and performs one when not.

```
int valid;
valid = IsShcValid();
if (.not.valid) {
    Shc.Execute();
}
```

IsSimValid*

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

```
int IsSimValid()
```

RETURNS

0 if no simulation results are available

DEPRECATED NAMES

validSIM

EXAMPLE

The following example checks if a simulation result is available.

```
int valid;
valid = validSIM();
if (.not.valid) {
    printf('No simulation result available');
}
```

LoadProfile

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

```
int LoadProfile(string profileName)
```

ARGUMENTS

profileName

Name of profile to be loaded.

RETURNS

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

EXAMPLE

The following example demonstrates how to active the 'Base Package' profile

```
int ret;

ret = LoadProfile('Base Package');
if (ret = 1) {
    printf('New profile has successfully been activated');
}
else {
    printf('Activation of new profile has failed');
}
```

SEE ALSO

[ReloadProfile\(\)](#)

LoadResData

This function is deprecated. Please use [ElmRes.Load\(\)](#) or [IntComtrade.Load\(\)](#) instead.

```
void LoadResData(object resultObject)
```

MarkInGraphics

This function is not supported in GUI-less mode.

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

```
void MarkInGraphics(set 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 (default).
- 1** Only search in currently opened diagrams and open the first diagram in which the elements were found.

EXAMPLE

The following example will mark a set of lines in the currently visible diagram or if not found in one of the other diagrams which are currently opened .

```
set lines;
lines = GetCalcRelevantObjects('*.ElmLne');
MarkInGraphics(lines, 1);
```

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.

```
void OutputFlexibleData(set objects,
                        [string 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.

EXAMPLE

The following example collects all lines and terminals which are relevant to the calculation and output their defined Flexible Data to the output window:

```
set elements;
elements = GetCalcRelevantObjects('*.ElmLne,*.ElmTerm');
OutputFlexibleData(elements);
```

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.

```
void PostCommand(string command)
```

ARGUMENTS

command

The command string.

EXAMPLE

The following command causes **PowerFactory** to end after the DPL script has finished.

```
PostCommand('exit');
```

PrepForUntouchedDelete

Mark (as not modified) all objects stored in the grid (or stored in set). Required for function [DeleteUntouchedObjects\(\)](#).

```
void PrepForUntouchedDelete(object grid)
void PrepForUntouchedDelete(set grids)
```

Rebuild*

Rebuilds the currently visible single line diagram or plot.

```
void Rebuild([int iMode])
```

ARGUMENTS

iMode (optional)

- 0** Draws graphic objects only
- 1** (default) Reads graphic objects (IntGrf) from database and draws
- 2** For single line diagrams: Reads graphic objects (IntGrf) from database, re-calculates intersections and draws.
For plot pages: Adjust view to page format and re-create plots.

EXAMPLE

```
Rebuild(1); !typical usage to perform a single line diagram update
```

ReleaseResData

This function is deprecated. Please use [ElmRes.Release\(\)](#) or [IntComtrade.Release\(\)](#) instead.

```
void ReleaseResData(object resultObject)
```

ReloadProfile

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

```
void ReloadProfile()
```

SEE ALSO

[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.

```
void ResetCalculation()
```

SEE ALSO

[IsAutomaticCalculationResetEnabled\(\)](#), [SetAutomaticCalculationResetEnabled\(\)](#)

ResFirstValidObject

This function is deprecated. Please use [ElmRes.GetFirstValidObject\(\)](#) instead.

```
int ResFirstValidObject(object resultFile,
                      int row,
                      [string classNames]
                      [string variableName,]
                      [double limit,]
                      [int limitOperator,]
                      [double limit2,]
                      [int limitOperator2])
int ResFirstValidObject([object resultFile,
                      [int row,]
                      [set objects])
```

ResFirstValidObjectVar

This function is deprecated. Please use [ElmRes.GetFirstValidObjectVariable\(\)](#) instead.

```
int ResFirstValidObjectVar(object resultFile,
                          [string variableNames])
```

ResFirstValidVar

This function is deprecated. Please use [ElmRes.GetFirstValidVariable\(\)](#) instead.

```
int ResFirstValidVar(object resultFile,
                     int row,
                     [string variableNames])
```

ResGetMax

This function is deprecated. Please use [ElmRes.FindMaxInColumn\(\)](#) or [IntComtrade.FindMaxInColumn\(\)](#) instead.

```
int ResGetMax(object resultFile,
              int col,
              [double& value])
```

ResGetMin

This function is deprecated. Please use [ElmRes.FindMinInColumn\(\)](#) or [IntComtrade.FindMinInColumn\(\)](#) instead.

```
int ResGetMin(object resultFile,
              int col,
              [double& value])
```

ResIndex

This function is deprecated. Please use [ElmRes.FindColumn\(\)](#) or [IntComtrade.FindColumn\(\)](#) instead.

```
int ResIndex(object resultFile,
             object obj,
             [string varName])
int ResIndex(object resultFile,
             object obj,
             [int colIndex])
int ResIndex(object resultFile,
             [string varName,]
             [int colIndex])
```

ResNextValidObject

This function is deprecated. Please use [ElmRes.GetNextValidObject\(\)](#) instead.

```
int ResNextValidObject(object resultFile,
                      [string classNames]
                      [string variableName,]
                      [double limit,]
                      [int limitOperator,]
                      [double limit2,]
                      [int limitOperator2])
int ResNextValidObject(object resultFile,
                      set objects)
```

ResNextValidObjectVar

This function is deprecated. Please use [ElmRes.GetNextValidObjectVariable\(\)](#) instead.

```
int ResNextValidObjectVar(object resultFile,
                          [string variableNames])
```

ResNextValidVar

This function is deprecated. Please use [ElmRes.GetNextValidVariable\(\)](#) instead.

```
int ResNextValidVar(object resultFile,
                     [string variableNames])
```

ResNval

This function is deprecated. Please use [ElmRes.GetNumberOfRows\(\)](#) or [IntComtrade.GetNumberOfRows\(\)](#) instead.

```
int ResNval(object resultObject,
            [int col])
```

ResNvars

This function is deprecated. Please use [ElmRes.GetNumberOfColumns\(\)](#) or [IntComtrade.GetNumberOfColumns\(\)](#) instead.

```
int ResNvars(object resultObject)
```

ResSortToVar

This function is deprecated. Please use [ElmRes.SortAccordingToColumn\(\)](#) or [IntComtrade.SortAccordingToColumn\(\)](#) instead.

```
int ResSortToVar(object resultObject,  
                 int col)
```

SaveAsScenario

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

```
object SaveAsScenario(string 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. NULL is returned in case of creation of a new scenario was not allowed (e.g. no active project).

DEPRECATED NAMES

SaveScenarioAs

EXAMPLE

The following example demonstrates how to save a scenario:

```
object scenario;  
scenario = SaveAsScenario('NewScenario', 0); ! do not activate the new scenario  
printf('The following scenario was created: %o', scenario);
```

SearchObjectByForeignKey*

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

```
object SearchObjectByForeignKey(string foreignKey)
```

ARGUMENTS

foreignKey

Foreign key

RETURNS

Object if found, otherwise NULL.

EXAMPLE

The following example shows how to search for an object by foreign key:

```
object obj;
str foreignKey;
foreignKey = '...';
obj = SearchObjectByForeignKey(foreignKey);
printf('Object found: %o', obj);
```

SelectToolbox

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

```
int SelectToolbox(int toolbar,
                  string groupName,
                  string toolboxName)
```

ARGUMENTS

- | | | |
|----------------|----------|------------------------|
| <i>toolbar</i> | 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. |

SetShowAllUsers

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

```
int SetShowAllUsers(int enabled)
```

ARGUMENTS

- | | |
|----------------|---|
| <i>enabled</i> | |
| 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. |

Sleep*

Suspends the execution of the script for given duration.

```
void Sleep(double duration)
```

ARGUMENTS

duration Duration in milliseconds

SplitLine

Splits the passed line or ElmBranch-object at a given position

```
object SplitLine(object Line,
                 [double percent = 50,]
                 [int createSwitchSide0 = 0,]
                 [int createSwitchSide1 = 0])
```

ARGUMENTS

double rPercent (optional)

Position in percent from the first connection side where line shall be split.

int iCreateSwitchSide0 (optional)

0 (default) Do not create switch on first connection side.

1 Create switch on first connection side.

int iCreateSwitchSide1 (optional)

0 (default) Do not create switch on second connection side.

1 Create switch on second connection side.

RETURNS

Inserted terminal

0 = error

StatFileGetXrange

Gets the x-range for the statistic result file.

```
int StatFileGetXrange(double& min,
                      double& max)
```

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.

```
void StatFileResetXrange()
```

StatFileSetXrange

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

```
void StatFileSetXrange(double min,
                      double max)
```

ARGUMENTS

min First point in time to be considered in statistics.

max Last point in time to be considered in statistics.

2.1 File System

Overview

```
CheckFileExists*
CopyFile*
CreateDir*
fclose*
fflush*
fopen*
fprintf*
fscanf*
fscanfsep*
GetDirectories*
GetFileDate*
GetFiles*
GetInstallationDirectory*
GetTemporaryDirectory*
GetWorkspaceDirectory*
```

CheckFileExists*

Checks if a file exists.

```
int CheckFileExists(string filename)
```

ARGUMENTS

filename Name and path of a file.

The file name can contain characters such as '*' and '?'. In case of wildcard characters are used, the function searches for existence of any file that matches the criteria.

Example e.g. 'c:*.exe' to search for any executable on drive C.

RETURNS

Returns 1 if exists, 0 otherwise.

EXAMPLE

Checks for existence of Windows Help Viewer:

```
int exists;
string file;

file = 'C:\Windows\winhlp32.exe';

exists = CheckFileExists(file);
if (exists = 1) {
    printf('File "%s" exists', file);
}
else {
    printf('File "%s" does not exist', file);
}
```

RETURNS

- 1** If file exists
- 0** If file does not exist

SEE ALSO

[GetDirectories\(\)](#), [GetFiles\(\)](#)

CopyFile*

Creates a copy of a file.

```
int CopyFile(string sourceFile, string targetFile)
```

ARGUMENTS

- sourceFile* Path and name of the file to copy
targetFile Path and name of the new file

RETURNS

1 if successfull, 0 if not.

EXAMPLE

The following example creates a copy of a file and prints the result code of CopyFile.

```
string sourceFile;
string targetFile;
int success;

sourceFile = 'E:\tmp\source.txt';
targetFile = 'E:\tmp\target.txt';
success = CopyFile(sourceFile, targetFile);
printf('CopyFile returned %d', success);
```

CreateDir*

Creates a complete directory path with all necessary subdirectories.

```
void CreateDir(string path)
```

ARGUMENTS

path The path that should be created

fclose*

Closes a file previously opened by [fopen\(\)](#).

Please note, calling the fclose for a currently not opened or invalid file handle will result in a scripting error.

```
void fclose(int handleNumber)
```

ARGUMENTS

handleNumber

File handle number of an open file (see [fopen\(\)](#)).

SEE ALSO

[fopen\(\)](#)

fflush*

Writes all content of the associated buffer to the file.

```
void fflush(int handleNumber)
```

ARGUMENTS

handleNumber

File handle number of an open file (see [fopen\(\)](#)).

fopen*

Opens a file for output or input. The function [fclose\(\)](#) is to be called for closing the file and releasing the handle.

```
int fopen(string filename,
          string mode,
          int handleNumber,
          [int continueScriptOnError])
```

ARGUMENTS

filename Path of file to open. Path must exist. File could be created depending on the 'mode'

mode Access mode for opening the file (r,w,a,r+,w+,a+,rb,wb,ab,r+b,w+b,a+b).
Same as used for C++ fopen().

handleNumber

≥ 0 , file handle number to be used for accessing the file.

continueScriptOnError (optional)

- 0** Default. In case of an error the script will be stopped and an error messagebox will appear.
- 1** Function will return a proper execution code informing about the success of the open action.

RETURNS

- 0** On success.
- 1** On error.

SEE ALSO

[fclose\(\)](#), [fprintf\(\)](#), [fscanf\(\)](#)

EXAMPLE

The following example demonstrates a basic writing and reading process:

```
int i, ret;
string line;

!output some text to a file
ret = fopen('d:\test.txt', 'w', 1, 1);
if (ret > 0) {
    Error('File could not be opened writing');
    exit();
}
for(i=0; i<10; i+=1) {
    fprintf(1, 'This is line %d', i);
}
fclose(1);

!read in the file again and print lines to output window
ret = fopen('d:\test.txt', 'r', 1, 1);
if (ret > 0) {
    Error('File could not be opened reading');
    exit();
}
SetLineFeed(0);!disable newline for printf
while(1) {
    !read next line
    ret = fscanf(1, '\l%s', line); !read complete line
    if (ret < 1) {
        break; !no next line
    }
    printf('%s', line);
}
fclose(1);
```

fprintf*

Prints a formatted string to a file and automatically inserts a line-break.

The DPL script stops with an appropriate error when the passed arguments don't match the format string. The line-break insertion can be disabled with [SetLineFeed\(\)](#).

```
string fprintf(int handleNumber,
               string format,
               [int|double|string|object argument0,]
               [...])
```

ARGUMENTS

handleNumber

File handle number of an open file (see [fopen\(\)](#)).

format

C++-like printf() format string. See the [Format String Syntax](#) for more information.

argument... (optional)

Arguments matching the format string.

RETURNS

The printed string.

SEE ALSO

[fopen\(\)](#), [fclose\(\)](#), [fscanf\(\)](#), [fscanfsep\(\)](#)

EXAMPLE

```
double x;
int i;
string s;
fopen('e:\tmp\test.txt', 'w', 0);
x = 123456789.987654321;
i = 2468;
s = 'hello dpl';
fprintf(0, 'string:%s int=%d double=%f', s, i, x);
fclose(0);
```

fscanf*

Parses a line of values which are separated by comma or tabulator from the file according the given format. Each parsed value is stored into the passed arguments. Empty tokens are skipped.

```
int fscanf(int handleNumber,
           string format,
           [int|double|string argument0,]
           [...])
```

ARGUMENTS

handleNumber

File handle number of an open file (see [fopen\(\)](#)).

format

C++-like printf() format string. See the [Format String Syntax](#) for more information.
Only int, double and string types are supported.

argument... (optional)

Arguments matching the format string.

RETURNS

≥ 0 Number of values successfully converted and assigned.

-1 On error.

SEE ALSO

[fscanfsep\(\)](#), [fprintf\(\)](#)

EXAMPLE

The file 'test.txt'

```
249.87 Hz
250.11 Hz
```

can be read by the following script

```
int result;
double freq;
string unit;

fopen('e:\tmp\test.txt', 'r', 0);

while (result>=0) {
    result = fscanf(0, '%f %s', freq, unit);
    printf('%f %s (result = %d)', freq, unit, result);
}

fclose(0);

! Output:
! 249,870000 Hz (result = 2)
! 250,110000 Hz (result = 2)
! 250,110000 Hz (result = -1)
```

fscanfsep*

Parses a line of values which are separated by a separation character from the file according the given format. Each parsed value is stored into the passed arguments. Empty tokens are skipped.

```
int fscanfsep(int handleNumber,
              string format,
              [int|double|string] argument0,
              [...],
              string separator,
              [int stopAfterLine = 0]
            )
```

ARGUMENTS

handleNumber

File handle number of an open file (see [fopen\(\)](#)).

format C++-like printf() format string. See the [Format String Syntax](#) for more information. Only int, double and string types are supported.

argument... (*optional*)

Arguments matching the format string.

separator Character that separates values in given input string.

stopAfterLine (*optional*)

- 1** Interpretation of the line will be stopped after the current line.
- 0** Continued interpretation (default).

RETURNS

- ≥ 0 Number of values successfully parsed and stored.
- 1 On error.

SEE ALSO

[fscanf\(\)](#), [fprintf\(\)](#)

EXAMPLE

The file 'test.txt'

```
249.87;Hz;
250.11;Hz;
```

can be read by the following script

```
int result;
double freq;
string unit;

fopen('e:\tmp\test.txt', 'r', 0);

while (result > -1) {
    result = fscanfsep(0, '%f %s', freq, unit, ';', 0);
    if (result == -1) {
        break;
    }
    else {
        printf('%f %s    (result = %d)', freq, unit, result);
    }
}
fclose(0);
```

GetDirectories*

Searches a given directory with optional filter criteria for subdirectories and stores the result in an *IntDplvec*. The directory must either end with a backslash or a filter, e.g. 'example*'.

```
void GetDirectories(string directory,
                   object resultVec)
```

ARGUMENTS

directory Path with optional filter of the search directory

resultVec *IntDplvec* with the found directory names

EXAMPLE

Searches a directory for all subdirectories beginning with 'example' and prints them to the Output Window. The script requires 'vec' to be an *IntDplvec* stored below the script.

```
int vecSize;
int i;
```

```

string fileName;

vec.Clear();
GetDirectories('E:\\tmp\\example*', vec);
vecSize = vec.Size();

for (i = 0; i < vecSize; i+=1)
{
    fileName = vec.Get(i);
    printf('%s', fileName);
}

```

GetFileDialog*

Returns the last modification date of given file.

```
string GetFileDialog(string file)
```

ARGUMENTS

file File name, either absolute or relative

RETURNS

Last modification date of given file for local timezone in format "YYYY-MM-DD HH:MM:SS", e.g. "2009-09-08 15:29:40". An empty string is returned in case that the file does not exist or access is denied.

SEE ALSO

[CheckFileExists\(\)](#)

GetFiles*

Searches a given directory with optional filter criteria for files and stores the result in an *IntDplvec*. The directory must either end with a backslash or a file filter, e.g. '*.TXT'.

```
void GetFiles(string directory,
              object resultVec)
```

ARGUMENTS

directory Path with optional filter of the search directory

resultVec *IntDplvec* with the found file names

EXAMPLE

Searches a directory for all files ending with 'dgs' and prints them to the Output Window. The script requires 'vec' to be an *IntDplvec* stored below the script.

```

int vecSize;
int i;
string fileName;

vec.Clear();
GetFiles('E:\\tmp\\*.dgs', vec);
vecSize = vec.Size();

```

```
for (i = 0; i < vecSize; i+=1)
{
    fileName = vec.Get(i);
    printf('%s', fileName);
}
```

GetInstallationDirectory*

Returns the installation directory of **PowerFactory**.

```
string GetInstallationDirectory()
```

RETURNS

Full path to installation directory of current **PowerFactory**.

DEPRECATED NAMES

GetInstallDir

SEE ALSO

[GetTemporaryDirectory\(\)](#), [GetWorkspaceDirectory\(\)](#)

GetTemporaryDirectory*

Returns the temporary directory of used by **PowerFactory**.

```
string 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

[GetWorkspaceDirectory\(\)](#), [GetInstallationDirectory\(\)](#)

GetWorkspaceDirectory*

Returns the workspace directory of **PowerFactory**.

```
string GetWorkspaceDirectory()
```

RETURNS

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

DEPRECATED NAMES

GetWorkingDir

SEE ALSO

[GetTemporaryDirectory\(\)](#), [GetInstallationDirectory\(\)](#)

2.2 Date/Time

Overview

[FormatDateLT*](#)
[FormatDateUTC*](#)
[GetStudyTimeObject*](#)
[GetSystemTime*](#)
[GetSystemTimeUTC*](#)
[GetTime*](#)
[ParseDateLT*](#)
[ParseDateUTC*](#)
[strftime*](#)

FormatDateLT*

Creates a formatted date string. The time must be given in seconds elapsed since 01.01.1970 00:00 in UTC and is converted to a display string in local time.

```
string FormatDateLT(string format,
                     int timeInSecondsUTC)
```

ARGUMENTS

format

%d	Day of month as decimal number (01..31)
%H	Hour in 24-hour format (00..23)
%m	Month as decimal number (01..12)
%M	Minute as decimal number (00..59)
%S	Second as decimal number (00..59)
%Y	Year with century, as decimal number

timeInSecondsUTC

Time in seconds since 01.01.1970 00:00 in UTC

RETURNS

Formatted date string in local time

EXAMPLE

The following example gets the formatted date string in local time:

```
string str;
int t;

t = 1180703210;
str = FormatDateLT('%Y-%m-%d %H:%M:%S', t);

printf('%d seconds in utc -> local time string %s', t, str);

!Output: 1180703210 seconds in utc -> local time string 2007-06-01 15:06:50
```

FormatDateUTC*

Creates a formatted date string. The time must be given in seconds elapsed since 01.01.1970 00:00 and is considered to be in UTC time. No time zone corrections are done.

```
string FormatDateUTC(string format,
                      int timeInSecondsUTC)
```

ARGUMENTS

format

%a	Abbreviated weekday name
%A	Full weekday name
%b	Abbreviated month name
%B	Full month name
%c	Date and time representation appropriate for locale
%d	Day of month as decimal number (01..31)
%H	Hour in 24-hour format (00..23)
%I	Hour in 12-hour format (01..12)
%j	Day of year as decimal number (001..366)
%m	Month as decimal number (01..12)
%M	Minute as decimal number (00..59)
%p	Current locale's A.M./P.M. indicator for 12-hour clock
%S	Second as decimal number (00..59)
%U	Week of year as decimal number, Sunday as first day of week (00..53)
%w	Weekday as decimal number (0..6; Sunday is 0)
%W	Week of year as decimal number, Monday as first day of week (00..53)
%x	Date representation for current locale
%X	Time representation for current locale
%y	Year without century, as decimal number (00..99)
%Y	Year with century, as decimal number
%tz	Time-zone as offset (+/- hh:mm), always +00:00
%%	Percent sign

timeInSecondsUTC

Time in seconds since 01.01.1970 00:00 in UTC time

RETURNS

Formatted date string

EXAMPLE

The following example gets the formatted date string:

```
string date;
int seconds;

date = FormatDateUTC('%Y-%m-%dT%H:%M:%S%tz', 1153840426);
printf('%s', date);

!Output: 2006-07-25T15:13:46+00:00

seconds = GetSystemTimeUTC();
```

```

date = FormatDate('%Y-%m-%dT%H:%M:%S%tz', seconds);
printf('%s', date);      !current time

!Output: 'current time', e.g. 2010-06-08T06:12:15+02:00

date = FormatDate ('%Y-%m-%dT%H:%M:%S%tz', 0);
printf('%s', date);

!Output: 1970-01-01T00:00:00+00:00  seconds;

```

GetStudyTimeObject*

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

```
object GetStudyTimeObject()
```

RETURNS

SetTime or NULL.

EXAMPLE

The following example uses the default SetTime object to change the calculation time, and then executes the load flow command with the name 'Unbalanced'.

```

object time, ldf;
time = GetStudyTimeObject();
time:hourofyear = 1234;
ldf = GetFromStudyCase('Unbalanced.ComLdf');
ldf.Execute();

```

GetSystemTime*

Returns the current system time in seconds since 00:00 01.01.1970GMT. This time is always in local time. For getting the time in UTC, a function [GetSystemTimeUTC\(\)](#) is available.

```
int GetSystemTime()
```

RETURNS

Current system time in seconds since 00:00 01.01.1970GMT

EXAMPLE

The following example gets the current system time in seconds:

```

int seconds;
seconds = GetSystemTime();
printf('Now in local time %d', seconds);

```

GetSystemTimeUTC*

Returns the current system time in seconds since 00:00 01.01.1970GMT. This time is always in UTC. For getting the local time, a function [GetSystemTime\(\)](#) is available.

```
int GetSystemTimeUTC()
```

RETURNS

Current system time in seconds since 00:00 01.01.1970GMT for UTC zone.

EXAMPLE

The following example gets the current system time in UTC seconds:

```
int seconds;
seconds = GetSystemTimeUTC();
printf('Now in UTC seconds %d', seconds);
```

GetTime*

Returns current processor time.

```
double GetTime(int precision)
```

ARGUMENTS

precision Precision after decimal point

RETURNS

Current processor time in seconds

ParseDateLT*

Parses a given date string that represents a date in local time and returns the corresponding UTC time in seconds elapsed since 01.01.1970 00:00 UTC.

```
int ParseDateLT(string format,
                string date)
```

ARGUMENTS

format

%d	Day of month as decimal number (01..31)
%H	Hour in 24-hour format (00..23)
%m	Month as decimal number (01..12)
%M	Minute as decimal number (00..59)
%S	Second as decimal number (00..59)
%Y	Year with century, as decimal number

date Formatted date string (local time)

RETURNS

Date in seconds since 00:00 01.01.1970 UTC.

EXAMPLE

The following example returns the date in seconds since 00:00 01.01.1970 UTC:

```
string date;
int t;

date = '2007-06-01 15:06:50';
t = ParseDateLT('%Y-%m-%d %H:%M:%S', date);
printf('%d', t);

!Output: 1180703210
```

ParseDateUTC*

Parses a given date string and returns the date in seconds elapsed since 01.01.1970 00:00 UTC. If a time zone is given and specified in format string this information is used to convert the seconds to UTC.

```
int ParseDateUTC(string format,
                 string date)
```

ARGUMENTS*format*

%d	Day of month as decimal number (01..31)
%H	Hour in 24-hour format (00..23)
%m	Month as decimal number (01..12)
%M	Minute as decimal number (00..59)
%S	Second as decimal number (00..59)
%Y	Year with century, as decimal number

date Formatted date string

RETURNS

Date in seconds since 00:00 01.01.1970 UTC that is represented by given date string.

EXAMPLE

The following example returns the date in seconds since 00:00 01.01.1970 UTC:

```
int t;

t = ParseDateUTC('%Y-%m-%d %H:%M', '1970-01-01 00:00');
printf('%i', t);

!Output: 0

t = ParseDateUTC('%Y-%m-%d %H:%M%tz', '1970-01-01 00:00-01:00');
printf('%i', t);

!Output: 3600

t = ParseDateUTC('%Y-%m-%d %H:%M', '2006-07-25 11:59');
printf('%i', t);

!Output: 1153828740
```

strftime*

Creates a formatted time string.

```
string strftime(string format, int time)
```

ARGUMENTS

Format The format string The following formatting codes are recognized in the format string.

%a	Abbreviated weekday name
%A	Full weekday name
%b	Abbreviated month name
%B	Full month name
%c	Date and time representation appropriate for locale
%d	Day of month as decimal number (01..31)
%H	Hour in 24-hour format (00..23)
%I	Hour in 12-hour format (01..12)
%j	Day of year as decimal number (001..366)
%m	Month as decimal number (01..12)
%M	Minute as decimal number (00..59)
%p	Current locale's A.M./P.M. indicator for 12-hour clock
%S	Second as decimal number (00..59)
%U	Week of year as decimal number, Sunday as first day of week (00..53)
%w	Weekday as decimal number (0..6; Sunday is 0)
%W	Week of year as decimal number, Monday as first day of week (00..53)
%x	Date representation for current locale
%X	Time representation for current locale
%y	Year without century, as decimal number (00..99)
%Y	Year with century, as decimal number
%z, %Z	Time-zone name or abbreviation; no characters if zone is unknown
%%	Percent sign

RETURNS

The formatted time string

DEPRECATED NAMES

FormatDate

EXAMPLE

The following example shows the date.

```
str = strftime('Today is %A, day %d of %B in the year %Y.');
printf('%s', str);
```

Output: Today is Wednesday, day 30 of April in the year 2003.

2.3 Dialogue Boxes

Overview

[CloseTableReports](#)
[GetTableReports](#)
[input](#)
[MessageBox](#)
[ShowModalBrowser](#)
[ShowModalOpenFileDialog](#)
[ShowModalSaveFileDialog](#)
[ShowModalSelectBrowser](#)
[ShowModalSelectFolderDialog](#)
[ShowModelessBrowser](#)
[UpdateTableReports](#)

CloseTableReports

This function is not supported in GUI-less mode.

Closes all open table reports.

Please note: Table reports currently running one of their scripts are not closed.

```
void CloseTableReports()
```

GetTableReports

This function is not supported in GUI-less mode.

Returns all open table reports.

```
set GetTableReports()
```

input

This function is not supported in GUI-less mode.

Provides the possibility to get user input during the execution of a DPL script. When executed, an input box is displayed. The execution of the script pauses until the user presses the OK button. On cancel, the running DPL script is aborted.

```
void input(string inputStr,
          string msg,
          [int& length])
void input(double inputDbl,
          string msg,
          [int& length])
```

ARGUMENTS

inputStr — inputDbl

Output variable that will hold the user's input; depending on the type, the input is returned as string or as double.

msg Message displayed in the input box.

length (optional, out)

If given, the input is limited to 'length' characters. In addition, this determines the dialog's size (default: 18).

RETURNS

Please note, that the execution of the script is aborted if the user cancels the input request.

EXAMPLE

The following example displays first an input box to enter a number and then to enter a text:

```
double number;
string text;

input(number, 'Please enter a number');
printf('Entered number: %f', number);

input(text, 'Please enter some text');
printf('Entered text: %s', text);
```

MessageBox

This function is not supported in GUI-less mode.

MessageBox shows a message box while pausing the execution of a DPL script, and then returns the pressed button.

```
int MessageBox(string title,
              string message,
              [int buttons = 1,]
              [int defaultButton = 1]
              [int icon = 1])
```

ARGUMENTS

title Title of the message box

message Message displayed in the message box

buttons (optional)

Flag for the buttons of the message box (default 1)

- 1** Ok button (default)
- 2** Cancel button
- 4** Yes button
- 8** 'Yes to all' button
- 16** No button

defaultButton (optional)

Default button of the message box (default 1)

icon (optional)

Icon of the message box (default 0)

- 0** Information icon
- 1** Question icon
- 2** Warning icon
- 3** Error icon

RETURNS

Returns an integer for the pressed button.

EXAMPLE

```
int res;

! simple message box
MessageBox('Title', 'Message');

! message box with 'ok' and 'cancel' button, 'ok' is default
! and information icon
res = MessageBox('Title', 'Message', 1+2, 2, 0);
if (res=1) {
    printf('Ok pressed.');
}
else {
    printf('Cancel pressed.');
}

! message box with 'yes' and 'no' button, 'no' is default
! and question icon
res = MessageBox('Title', 'Message', 4+16, 16, 1);
if (res=4) {
    printf('Yes pressed.');
}
else if (res=16) {
    printf('No pressed.');
}

! message box with 'yes', 'yes to all' and 'no' button,
! 'no' is default and warning icon
res = MessageBox('Title', 'Message', 4+8+16, 16, 2);
if (res=4) {
    printf('Yes pressed.');
}
else if (res=8) {
    printf('Yes to all pressed.');
}
else if (res=16) {
    printf('No pressed.');
}

! simple error message box
MessageBox('Title', 'Error occurred:\nOr not?', 1, 1, 3);
```

ShowModalBrowser

This function is not supported in GUI-less mode.

Opens a modal browser window and lists all given objects.

```
void ShowModalBrowser(set objects,
                      [int detailMode = 0,]
                      [string title = '',]
                      [string 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.

ShowModalOpenFileDialog

This function is not supported in GUI-less mode.

Opens a modal *Open File* dialogue which prompts the user to select an existing file.

```
string ShowModalOpenFileDialog([string title = '',  
                                [string filters = '',  
                                 [string initialPath = ''])
```

ARGUMENTS

title (optional)

The title of the dialogue. If this argument is not provided or an empty string, a suitable default title is used instead.

filters (optional)

Available file extension filters. A filter is specified by giving

- a non-empty filter description which may consist of multiple words, but no parentheses (e.g. 'Image files') and
- a semicolon-separated list of extensions (each starting with the prefix '*.') in parentheses (e.g ('*.jpg; *.jpeg; *.png; *.bmp')).

Please note that the filter selection field in the file dialogue will show the filter description *and* the corresponding extensions. Multiple filter definitions are separated by semicolons (e.g. 'Filter one (*.txt; *.out); Filter two (*.py; *.pyc); All Files (*.*)'). If multiple filter definitions are provided, the first filter in the list is selected when the dialogue is shown. If this argument is not provided or an empty string, an *All files* filter is used which allows the selection of any file.

initialPath (optional)

The folder to be displayed when the dialogue is shown. If this argument is not provided or an empty string, a suitable folder is used as initial location.

RETURNS

The path to the file selected by the user. If the user cancels the dialogue, an empty string is returned instead.

EXAMPLE

```

string title, filters, initialPath, selectedFile;
int pathLength;

title = 'Please select a file to open';
filters = 'Output files (*.out;*.txt); All files (*.*)';
initialPath = 'D:\temp'; ! change, if path does not exist

selectedFile = ShowModalOpenFileDialog(title, filters, initialPath);

pathLength = strlen(selectedFile);
if (pathLength > 0) {
    ! a file path has been provided by the user
    printf('User selected the file %s', selectedFile);
}
else {
    ! the user cancelled the file dialogue
    printf('User cancelled the dialogue');
}

```

SEE ALSO

[ShowModalSaveFileDialog\(\)](#), [ShowModalSelectFolderDialog\(\)](#)

ShowModalSaveFileDialog

This function is not supported in GUI-less mode.

Opens a modal *Save File* dialogue which prompts the user to specify a file. If the user chooses an existing file, they are asked if they want to replace the existing file.

```

string ShowModalSaveFileDialog([string title = '',
                                [string filters = '',
                                [string initialPath = '']

```

ARGUMENTS*title (optional)*

The title of the dialogue. If this argument is not provided or an empty string, a suitable default title is used instead.

filters (optional)

Available file extension filters. A filter is specified by giving

- a non-empty filter description which may consist of multiple words, but no parentheses (e.g. 'Image files') and
- a semicolon-separated list of extensions (each starting with the prefix '.') in parentheses (e.g. '(*.jpg; *.jpeg; *.png; *.bmp)').

Please note that the filter selection field in the file dialogue will show the filter description *and* the corresponding extensions. Multiple filter definitions are separated by semicolons (e.g. 'Filter one (*.txt; *.out); Filter two (*.py; *.pyc); All Files (*.*)'). If multiple filter definitions are provided, the first filter in the list is selected when the dialogue is shown. If this argument is not provided or an empty string, an *All files* filter is used which allows the selection of any file. Please note that if the user enters a file name and does not provide an extension from the active filter, the first extension of the active filter is appended automatically.

initialPath (optional)

The folder to be displayed when the dialogue is shown. If this argument is not provided or an empty string, a suitable folder is used as initial location.

RETURNS

The path to the file specified by the user. If the user cancels the dialogue, an empty string is returned instead.

EXAMPLE

```
string title, filters, initialPath, selectedFile;
int pathLength;

title = 'Export results to';
filters = 'Output files (*.out;*.txt); All files (*.*)';
initialPath = 'D:\temp'; ! change, if path does not exist

selectedFile = ShowModalSaveFileDialog(title, filters, initialPath);

pathLength = strlen(selectedFile);
if (pathLength > 0) {
    ! a file path has been provided by the user
    printf('User selected the file %s', selectedFile);
}
else {
    ! the user cancelled the file dialogue
    printf('User cancelled the dialogue');
}
```

SEE ALSO

[ShowModalOpenFileDialog\(\)](#), [ShowModalSelectFolderDialog\(\)](#)

ShowModalSelectBrowser

This function is not supported in GUI-less mode.

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

```
set ShowModalSelectBrowser(set objects,
                           [string title,]
                           [string classFilter,]
                           [string 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.

ShowModalSelectFolderDialog

This function is not supported in GUI-less mode.

Opens a modal *Select Folder* dialogue which prompts the user to select a folder.

```
string ShowModalSelectFolderDialog([string description = '', ]
                                    [string initialPath = ''])
```

ARGUMENTS

description (optional)

The text shown above the folder selection field of the dialogue. If this argument is not provided or an empty string, a suitable default description is used instead.

initialPath (optional)

The folder to select when the dialogue is shown. If this argument is not provided or an empty string, a suitable folder is used as initial selection.

RETURNS

The path to the folder specified by the user. If the user cancels the dialogue, an empty string is returned instead.

EXAMPLE

```
string description, initialPath, selectedFolder;
int pathLength;

description = 'Please select a folder';
initialPath = 'D:\temp'; ! change, if path does not exist

selectedFolder = ShowModalSelectFolderDialog(description, initialPath);
pathLength = strlen(selectedFolder);

if (pathLength > 0) {
    ! a folder has been selected by the user
    printf('User selected the folder %s', selectedFolder);
}
else {
    ! the user cancelled the folder selection dialogue
    printf('User cancelled the dialogue');
}
```

SEE ALSO

[ShowModalOpenFileDialog\(\)](#), [ShowModalSaveFileDialog\(\)](#)

ShowModelessBrowser

This function is not supported in GUI-less mode.

Opens a modeless browser window and lists all given objects.

```
void ShowModelessBrowser(set 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

This function is not supported in GUI-less mode.

Updates all open table reports.

```
void UpdateTableReports()
```

2.4 Environment

Overview

[EchoOff](#)
[EchoOn](#)
[GetDiffMode](#)
[IsAutomaticCalculationResetEnabled*](#)
[IsFinalEchoOnEnabled](#)
[NoFinalUpdate](#)
[SetAutomaticCalculationResetEnabled](#)
[SetConsistencyCheck](#)
[SetDiffMode](#)
[SetFinalEchoOnEnabled](#)
[SetGraphicUpdate](#)
[SetGuiUpdateEnabled](#)
[SetProgressBarUpdatesEnabled](#)
[SetUserBreakEnabled](#)

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 SetFinalEchoOnEnabled().

```
void EchoOff()
```

SEE ALSO

[EchoOn\(\)](#), [IsFinalEchoOnEnabled\(\)](#), [SetFinalEchoOnEnabled\(\)](#)

EXAMPLE

The following example de-activates the user-interface to speed up the calculations, after which the user-interface is re-activated again.

```
EchoOff();
...
do some calculation ...
EchoOn();
```

EchoOn

Re-activates the user interface. For more informations see [EchoOff\(\)](#).

```
void EchoOn()
```

SEE ALSO

[EchoOff\(\)](#), [IsFinalEchoOnEnabled\(\)](#), [SetFinalEchoOnEnabled\(\)](#)

GetDiffMode

Returns the currently set result access mode. See [SetDiffMode\(\)](#) for more information.

```
int GetDiffMode()
```

RETURNS

- 0** Base case results access.
- 1** Compare case results access.

SEE ALSO

[SetDiffMode\(\)](#), [ComDiff.Start\(\)](#), [ComDiff.Stop\(\)](#)

IsAutomaticCalculationResetEnabled*

Returns whether the automatic calculation reset while setting attributes via the ":" notation e.g. "load:scale0 = 0.98" is enabled. [object.SetVal\(\)](#) is not affected. See [SetAutomaticCalculationResetEnabled\(\)](#) for more informations.

```
int IsAutomaticCalculationResetEnabled()
```

SEE ALSO

[SetAutomaticCalculationResetEnabled\(\)](#), [ResetCalculation\(\)](#)

IsFinalEchoOnEnabled

Returns whether the automatic [EchoOn\(\)](#) at the end of each *ComDpl* or *ComPython* is enabled.

```
int IsFinalEchoOnEnabled();
```

RETURNS

- 1** Final [EchoOn\(\)](#) is enabled.
- 0** Final [EchoOn\(\)](#) is disabled.

SEE ALSO

[SetFinalEchoOnEnabled\(\)](#), [EchoOn\(\)](#), [EchoOff\(\)](#)

NoFinalUpdate

This function is deprecated. Use [SetFinalEchoOnEnabled\(\)](#) instead.

```
void NoFinalUpdate();
```

SetAutomaticCalculationResetEnabled

Enables or disables the automatic calculation reset while setting attributes via the ":" notation e.g. "load:scale0 = 0.98". [object.SetVal\(\)](#) is not affected.

In DPL/QDSL the automatic calculation reset is by default disabled. Thus only changing the "outserv" attribute of an 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 a disabled automatic calculation reset can be helpful e.g. when calculating a load-flow without a flat start after changing the scaling factor of a load. 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 [ResetCalculation\(\)](#).

If the automatic calculation reset is enabled, changing an arbitrary 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.

```
void SetAutomaticCalculationResetEnabled(int enabled);
```

SEE ALSO

[IsAutomaticCalculationResetEnabled\(\)](#), [ResetCalculation\(\)](#)

SetConsistencyCheck

This function enables or disables the value consistency check executed whenever an attribute is set. The consistency check is enabled by default.

Note: Disabling of consistency check might be required when dependent attributes are set where the object is temporarily left in an invalid state until all attributes are set. In Python please use [Application.SetWriteCacheEnabled\(\)](#) for the same purpose.

```
int SetConsistencyCheck(int iEnable);
```

ARGUMENTS

iEnable

- | | |
|----------|-------------------------------------|
| 0 | Disable attribute consistency check |
| 1 | Enable attribute consistency check |

RETURNS

- 1** If consistency check was enabled before.
- 0** If consistency check was disabled before.

SetDiffMode

This function allows switching between base and compare case result access mode when using the comparing results functionality of **PowerFactory** (see User Manual Chapter: Comparisons Between Calculations). Depending on this mode, the access to object parameters returns base case values or is redirected to result case values. There is no need to adapt the parameter access statements.

```
void SetDiffMode (int mode)
```

ARGUMENTS

mode

- 0** Base case results access.
- 1** Compare case results access.

SEE ALSO

[GetDiffMode\(\)](#), [ComDiff.Start\(\)](#), [ComDiff.Stop\(\)](#)

EXAMPLE

The following example demonstrates how to access the comparing results functionality from DPL.

```
object term, comldf, comdiff;
set terms;
double u1, u2;

!get load flow calculation command
comldf = GetFromStudyCase('ComLdf');
comdiff = GetFromStudyCase('ComDiff');

comdiff:imode = 1; !set compare mode to desired mode

!initially set length of a special line to 1km
lne:dline = 1.0;

!calculate load flow
comldf.Execute();

!start comparing results
comdiff.Start();

!change length of line to 1000km
lne:dline = 1000.0;

!calculate load flow again
comldf.Execute();

!report differences:
!for all relevant terminals
terms = GetCalcRelevantObjects('* ElmTerm');
for (term = terms.First(); term; term = terms.Next()) {
    SetDiffMode(0); !base case results
```

```

u1 = term:m:u;
SetDiffMode(1); !compare case results
u2 = term:m:u;
printf('%o: u1=%f p.u.    u2=%f', term, u1, u2);
}
!stop comparing of results
comdiff.Stop();

!restore original settings
lne:dline = 1.0; ! 1kmLoadedElms(10);

```

SetFinalEchoOnEnabled

Enables or disables the automatic [EchoOn\(\)](#) at the end of each *ComDpl* or *ComPython*.

```
void SetFinalEchoOnEnabled(int enabled);
```

ARGUMENTS

enabled

- | | |
|---|---|
| 1 | Enables the final EchoOn() . |
| 0 | Disables the final EchoOn() . |

SEE ALSO

[IsFinalEchoOnEnabled\(\)](#), [EchoOn\(\)](#), [EchoOff\(\)](#)

EXAMPLE

```

EchoOff();
SetFinalEchoOnEnabled(0);
... do some calculation which calls other scripts ...
SetFinalEchoOnEnabled(1);

```

SetGraphicUpdate

Enables or disables the updates of the single line graphics.

```
void SetGraphicUpdate(int enabled)
```

ARGUMENTS

enabled

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

EXAMPLE

The following example disables and enables the graphics update:

```

!disable graphic updates
SetGraphicUpdate(0);
!do some calculations
!...
!enable graphic updates again
SetGraphicUpdate(1);

```

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.

Please note that the progress bar, which is located in the status bar of the application window, is not affected by this. Updates of the progress bar can be enabled or disabled separately by invoking [SetProgressBarUpdatesEnabled\(\)](#).

The updates will automatically be re-enabled after termination of the script. In case of subscripts, the restore is done at termination of main script.

```
int 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

[SetProgressBarUpdatesEnabled\(\)](#), [SetGraphicUpdate\(\)](#)

EXAMPLE

The following example disables and enables the GUI updates

```
int oldState;
! disable gui updates
oldState = SetGuiUpdateEnabled(0);
!do some calculations
!...
!restore gui updates old state
SetGuiUpdateEnabled(oldState);
```

SetProgressBarUpdatesEnabled

Enables or disables updates of the progress bar (located in the status bar of the application window) while the script is running. Other components of the status bar are not affected.

If a script executes a high number of small, fast commands that report progress (noticeable by the progress bar repeatedly and quickly filling up), disabling progress bar updates can provide an immense performance boost.

The updates will automatically be re-enabled after termination of the script. In case of subscripts, the restore is done at termination of main script.

```
int SetProgressBarUpdatesEnabled(int enabled)
```

ARGUMENTS

enabled

- 0** Disables progress bar updates.
- 1** Enables progress bar updates.

RETURNS

Previous state before the function was called

- 0** Progress bar updates were disabled before.
- 1** Progress bar updates were enabled before.

SEE ALSO

[SetGuiUpdateEnabled\(\)](#), [SetGraphicUpdate\(\)](#)

EXAMPLE

The following example disables and enables the GUI updates

```
int oldProgressBarUpdateState;
! disable progress bar updates
oldProgressBarUpdateState = SetProgressBarUpdatesEnabled(0);
! execute a high number of small, fast commands that report progress
!...
! restore progress bar updates
SetProgressBarUpdatesEnabled(oldProgressBarUpdateState);
```

SetUserBreakEnabled

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

```
void SetUserBreakEnabled(int enabled)
```

ARGUMENTS

enabled

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

DEPRECATED NAMES

`EnableUserBreak`, `SetEnableUserBreak`

2.5 Mathematics

Overview

abs*
acos*
asin*
atan*
atan2*
BinaryAnd*
BinaryOr*
CalcWeibullPar*
ceil*
cos*
cosh*
exp*
floor*
frac*
GetRandomNumber*
GetRandomNumberEx*
InvertMatrix*
ln*
log*
max*
min*
modulo*
pi*
pow*
RndExp*
RndGetMethod*
RndGetSeed*
RndNormal*
RndSetup*
RndUnifInt*
RndUnifReal*
RndWeibull*
round*
SetRandomSeed*
sin*
sinh*
sqr*
sqrt*
tan*
tanh*
time*
trunc*
twopi*

abs*

Calculates the absolute value.

```
double abs(double x)
```

ARGUMENTS

x A value.

RETURNS

The absolute value.

acos*

Calculates the arc cosine.

```
double acos(double x)
```

ARGUMENTS

x For $x \in [-1, 1]$.

RETURNS

Arc cosine of x, in radians.

asin*

Calculates the arc sine.

```
double asin(double x)
```

ARGUMENTS

x For $x \in [-1, 1]$.

RETURNS

Arc sine of x, in radians.

atan*

Calculates the arc tangent.

```
double atan(double x)
```

ARGUMENTS

x For a value x.

RETURNS

Arc tangent of x, in radians.

atan2*

Calculates $\text{atan}(y/x)$.

```
double atan2(double y, double x)
```

ARGUMENTS

- y** For a value y.
- x** For a value x.

RETURNS

Returns $\text{atan}(y/x)$, in radians.

BinaryAnd*

Performs a binary 'AND' operation on the two input arguments.

```
int BinaryAnd(int a, int b);
```

ARGUMENTS

- a** First value, either 0 or 1
- b** Second value, either 0 or 1

RETURNS

- 0** If one of the arguments is 0
- 1** If all of the arguments are 1

EXAMPLE

```
int r;
r = BinaryAnd(0, 0);
printf('0 and 0 = %d', r);
r = BinaryAnd(1, 0);
printf('1 and 0 = %d', r);
r = BinaryAnd(0, 1);
printf('0 and 1 = %d', r);
r = BinaryAnd(1, 1);
printf('1 and 1 = %d', r);
```

BinaryOr*

Performs a binary 'OR' operation on the two input arguments.

```
int BinaryOr(int a, int b);
```

ARGUMENTS

- a** First value, either 0 or 1
- b** Second value, either 0 or 1

RETURNS

- 0** If all of the arguments are 0
- 1** If one of the arguments is 1

EXAMPLE

```

int r;
r = BinaryOr(0, 0);
printf('0 or 0 = %d', r);
r = BinaryOr(1, 0);
printf('1 or 0 = %d', r);
r = BinaryOr(0, 1);
printf('0 or 1 = %d', r);
r = BinaryOr(1, 1);
printf('1 or 1 = %d', r);

```

CalcWeibullPar*

Performs a conversion between the weibull parameters listed below:

mean Mean value 'm'

lambda Scale Factor 'l'

variance Variance 'v'

beta Shape Factor 'b'

The output parameter is calculated from the two input parameters (e.g.: lambda = f(mean, beta))

```

int CalcWeibullPar(double& parameter,
                    double in1,
                    double in2,
                    string keyout,
                    string keyin1,
                    string keyin2)

```

ARGUMENTS

parameter (out)
Calculated weibull parameter

in1 The first input parameter

in2 The second input parameter

keyout Key specifying the variable of the output parameter

keyin1 Key specifying the variable of the first input parameter

keyin2 Key specifying the variable of the second input parameter

RETURNS

0 Ok

1 Calculation failed

2 Invalid or unknown key(s)

EXAMPLE

```
double mean,lambda,variance,beta;
int failed;
mean = 22;
beta = 3;
! Calculate the scale factor from the mean value and the shape
failed = CalcWeibullPar(lambda,mean,beta,'l','m','b');
if (failed) {
    printf('CalcWeibullPar failed');
}
else {
    printf('lambda = f(mean, beta):\n%f = f(%f,%f)',lambda,mean,beta);
}
```

ceil*

Calculates the smallest larger integer.

```
double ceil(double x)
```

ARGUMENTS

x A value.

RETURNS

The smallest larger integer.

cos*

Calculates the cosine.

```
double cos(double x)
```

ARGUMENTS

x A value in radians.

RETURNS

Cosine of *x*.

cosh*

Calculates the hyperbolic cosine.

```
double cosh(double x)
```

ARGUMENTS

x A value.

RETURNS

Hyperbolic cosine of *x*.

exp*

Calculates e^x , with the mathematical constant e (Euler's number).

```
double exp([double x = 1])
```

ARGUMENTS

x (optional)

A value (default = 1).

RETURNS

The value e^x .

floor*

Calculates the largest smaller integer.

```
double floor(double x)
```

ARGUMENTS

x A value.

RETURNS

The largest smaller integer.

frac*

Calculates the fractional part of *x*.

```
double frac(double x)
```

ARGUMENTS

x A value.

RETURNS

The integral part of *x*.

GetRandomNumber*

This function is marked as deprecated since PowerFactory 2017. Please use [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 interval [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*].

```
double GetRandomNumber([double x1,]
                      [double x2])
```

ARGUMENTS

x1 (optional)

x2 not given: maximum; *x1* and *x2* given: minimum

x2 (optional)

maximum

RETURNS

A uniformly distributed random number

DEPRECATED NAMES

Random

EXAMPLE

The following example sets a load to a random active power prior to calculating a load-flow.

```
double P;
Load:plini = GetRandomNumber(1.2, 2.3);
Ldf.Execute();
```

GetRandomNumberEx*

This function is marked as deprecated since PowerFactory 2017. Please use [RndUnifReal\(\)](#), [RndNormal\(\)](#) or [RndWeibull\(\)](#) instead.

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

```
double GetRandomNumberEx(int distribution,
                         [double p1,]
                         [double 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.

DEPRECATED NAMES

fRand

EXAMPLE

The following example prints random numbers for the following distributions:

```

!uni0 : an uniform distribution in [0..1]
!uni1 : an uniform distribution in [0..50]
!uni2 : an uniform distribution in [-8, 21];
!norm : a normal distribution with mean=30 and standard deviation=5
!weib : a Weibull distribution with scale=5 and shape=30
int n;
double uni0,uni1,uni2,norm,weib;
SetRandomSeed(2);
for (n=0; n<10; n+=1) {
    uni0 = GetRandomNumberEx(0);
    uni1 = GetRandomNumberEx(0, 50);
    uni2 = GetRandomNumberEx(0, -8, 21);
    norm = GetRandomNumberEx(1, 30, 5);
    weib = GetRandomNumberEx(2, 5, 30);
    printf('%f %f %f %f %f', uni0, uni1, uni2, norm, weib);
}

```

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

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

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 InvertMatrix(object realPart,
                 [object 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 successful. Resulting inverted matrix returned in input matrix/matrices. |

DEPRECATED NAMES

MatrixInvert

EXAMPLE

The following example shows the two possible applications of this method.

```

int errCode;
double val1,
       val2,
       val3,
       val4;

printf('--EXAMPLE FOR REAL MATRIX INVERSION--');
printf('The matrix A is an IntMat-object in the
      contents of this DPL script');

A.Resize(2, 2);
A.Set(1, 1, 0.5);
A.Set(1, 2, 0.0);
A.Set(2, 1, 0.0);
A.Set(2, 2, 0.5);

printf('The input matrix is:');
val1 = A.Get(1, 1);
val2 = A.Get(1, 2);
printf('A = [ %.1f %.1f ]', val1, val2);
val1 = A.Get(2, 1);
val2 = A.Get(2, 2);
printf('      [ %.1f %.1f ]', val1, val2);

errCode = InvertMatrix(A);
if (errCode) {
    printf('Matrix A could not be inverted.');
}
else {
    printf('The inverse of the input matrix is:');
    val1 = A.Get(1, 1);
    val2 = A.Get(1, 2);
    printf('Ainv = [ %.1f %.1f ]', val1, val2);
    val1 = A.Get(2, 1);
    val2 = A.Get(2, 2);
    printf('      [ %.1f %.1f ]', val1, val2);
}

printf('\n--EXAMPLE FOR COMPLEX MATRIX INVERSION--');
printf('The matrices realPart and imaginaryPart
      are IntMat-objects in the contents of this DPL script');

realPart.Resize(2, 2);
realPart.Set(1, 1, 0.5);
realPart.Set(1, 2, 0.0);
realPart.Set(2, 1, 0.0);
realPart.Set(2, 2, 0.5);

imaginaryPart.Resize(2, 2);
imaginaryPart.Set(1, 1, 0.5);
imaginaryPart.Set(1, 2, 0.0);
imaginaryPart.Set(2, 1, 0.0);
imaginaryPart.Set(2, 2, 0.5);

printf('The complex input matrix is:');
val1 = realPart.Get(1, 1);
val2 = realPart.Get(1, 2);

```

```

val3 = imaginaryPart.Get(1, 1);
val4 = imaginaryPart.Get(1, 2);
printf('C = [ %.1f + j*(%.1f)    %.1f + j*(%.1f) ]',
      val1, val3, val2, val4);
val1 = realPart.Get(2, 1);
val2 = realPart.Get(2, 2);
val3 = imaginaryPart.Get(2, 1);
val4 = imaginaryPart.Get(2, 2);
printf('      [ %.1f + j*(%.1f)    %.1f + j*(%.1f) ]',
      val1, val3, val2, val4);

errCode = InvertMatrix(realPart, imaginaryPart);
if (errCode) {
    printf('Matrix C could not be inverted.');
}
else {
    printf('The inverse of the input matrix C is:');
    val1 = realPart.Get(1, 1);
    val2 = realPart.Get(1, 2);
    val3 = imaginaryPart.Get(1, 1);
    val4 = imaginaryPart.Get(1, 2);
    printf('Cinv = [ %.1f + j*(%.1f)    %.1f + j*(%.1f) ]',
           val1, val3, val2, val4);
    val1 = realPart.Get(2, 1);
    val2 = realPart.Get(2, 2);
    val3 = imaginaryPart.Get(2, 1);
    val4 = imaginaryPart.Get(2, 2);
    printf('      [ %.1f + j*(%.1f)    %.1f + j*(%.1f) ]',
           val1, val3, val2, val4);
}

```

ln*

Calculates the natural logarithm of x (base e).

```
double ln(double x)
```

ARGUMENTS

x A value.

RETURNS

The natural logarithm of x (base e).

log*

Calculates the logarithm of x with base 10.

```
double log(double x)
```

ARGUMENTS

x A value.

RETURNS

The logarithm of x with base 10.

max*

Calculates the larger value.

```
double max(double x, double y)
```

ARGUMENTS

x A value.

y A value.

RETURNS

The larger value.

min*

Calculates the smaller value.

```
double min(double x, double y)
```

ARGUMENTS

x A value.

y A value.

RETURNS

The smaller value.

modulo*

Calculates the remainder after a division.

```
double modulo(double x, double y)
```

ARGUMENTS

x A value.

y A value.

RETURNS

The remainder after a division.

pi*

Return the mathematical constant π .

```
double pi()
```

RETURNS

The mathematical constant π .

pow*

Calculates x raised to the power y .

```
double pow(double x, double y)
```

ARGUMENTS

x A value.

y A value.

RETURNS

The value x^y .

RndExp*

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

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

ARGUMENTS

rate Rate of exponential distribution.

rngNum (optional)

Number of the random number generator.

0 (default) 'Global random number generator'.

1, 2, ... Other random number generators accessible 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 [RndSetup\(\)](#).

```
string 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 accessible 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 [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 accessible 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 [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 accessible 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 ComGenrelinc. Numbers 1,2,... will access different random number generators, which can be setup individually.

```
void 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 accessible via this number.

EXAMPLE

```
int i;
int numOfDraws;
int usedSeed;

int rngType;           ! Type of the random number generator.
int seedAutomatic;   ! automatic seeding true or false
int seed;              ! seed for the random number generator
int rngNum;            ! the number of the used random number generator
double draw;           ! drawn random numbers

rngType = 0;           ! Mersenne Twister
seedAutomatic = 0;     ! User defined seed
seed = 999;             ! Specific seed
rngNum = 1;             ! The random number generator used
                        ! (not the 'global random number generator',
                        ! global rng corresponds to number 0)
numOfDraws = 50;

RndSetup(seedAutomatic, seed, rngType, rngNum);

printf('Uniformly distributed random numbers (real):');
for(i = 0; i<numOfDraws; i=i+1){
    ! draw random number distributed according to
    ! uniform distribution in the intervall $[0,1]$
```

```

! using the random number generator rngNum
draw = RndUnifReal(0,1,rngNum);
printf('%.15e',draw);
}

printf('Uniformly distributed random numbers: (integer)');
for(i = 0; i<numOfDraws; i=i+1){
    ! draw random number distributed according to
    ! uniform distribution on the set of numbers $\{0,1,\ldots, 10\}$
    ! using the random number generator rngNum
    draw = RndUnifInt(0,10,rngNum);
    printf('%d',draw);
}

printf('Exponentially distributed random numbers:');
for(i = 0; i<numOfDraws; i=i+1){
    ! draw random number distributed according to
    ! exponential distribution of rate $0.5$
    ! using the random number generator rngNum
    draw = RndExp(0.5,rngNum);
    printf('%d',draw);
}

printf('Normally distributed random numbers:');
for(i = 0; i<numOfDraws; i=i+1){
    ! draw random number distributed according to
    ! normal distribution with mean $2$ and standard deviation $4$
    ! using the random number generator rngNum
    draw = RndNormal(2,4,rngNum);
    printf('%d',draw);
}

printf('Weibull distributed random numbers:');
for(i = 0; i<numOfDraws; i=i+1){
    ! draw random number distributed according to
    ! Weibull distribution with shape $1$ and scale $2$
    ! using the random number generator rngNum
    draw = RndWeibull(1,2,rngNum);
    printf('%d',draw);
}

! Get the seed, used to initialise the random number generator rngNum.
usedSeed = RndGetSeed(rngNum);
printf('Used seed: %d', usedSeed);

! Get the method of the random number generator rngNum.
printf('Used method to generate random numbers: %s', RndGetMethod(rngNum));

```

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 [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 accessible 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 [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 accessible 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 [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 accessible via this number.

RETURNS

double Random number

round*

Calculates the closest integer.

```
double round(double x)
```

ARGUMENTS

x A value.

RETURNS

The closest integer.

SetRandomSeed*

This function is marked as deprecated since PowerFactory 2017. Please use [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.

```
void SetRandomSeed(int seed)
```

ARGUMENTS

seed seed 0..9

DEPRECATED NAMES

SetRandSeed

sin*

Calculates the sine.

```
double sin(double x)
```

ARGUMENTS

x A value in radians.

RETURNS

Sine of *x*.

sinh*

Calculates the hyperbolic sine.

```
double sinh(double x)
```

ARGUMENTS

x A value.

RETURNS

Hyperbolic sine of x.

sqr*

Calculates the square.

```
double sqr(double x)
```

ARGUMENTS

x A value.

RETURNS

The square x^2 .

sqrt*

Calculates the square root.

```
double sqrt(double x)
```

ARGUMENTS

x A value.

RETURNS

The square root.

tan*

Calculates the tangent.

```
double tan(double x)
```

ARGUMENTS

x A value in radians.

RETURNS

Tangent of x.

tanh*

Calculates the hyperbolic tangent.

```
double tanh(double x)
```

ARGUMENTS

x A value.

RETURNS

Hyperbolic tangent of x.

time*

Return the current simulation time.

```
double time()
```

RETURNS

The current simulation time.

trunc*

Calculates the integral part of x.

```
double trunc(double x)
```

ARGUMENTS

x A value.

RETURNS

The integral part of x.

twopi*

Return the mathematical constant 2π .

```
double twopi()
```

RETURNS

The mathematical constant 2π .

2.6 MS Office

2.6.1 MS Access

Overview

[mdbClose](#)
[mdbExecuteSqlQuery](#)
[mdbExecuteSqlStatement](#)
[mdbFetchResult](#)
[mdbGetResultColumnCount](#)
[mdbGetResultColumnName](#)
[mdbGetResultColumnType](#)
[mdbGetResultColumnValue](#)
[mdbOpen](#)
[mdbSetDebug](#)

mdbClose

Closes currently opened MS Access file.

```
void mdbClose()
```

SEE ALSO

[mdbOpen\(\)](#)

mdbExecuteSqlQuery

Executes a SQL query. The result of the query can be obtained by calling `FetchResult()`.

Please note: Executing a new query will invalidate a previous one. It is not possible to have multiple queries open in parallel.

```
int mdbExecuteSqlQuery(string statement)
```

ARGUMENTS

statement

Sql statement as text, e.g. `SELECT FROM...` .

RETURNS

- 0** On success.
- 1** On error.

EXAMPLE

```
!// example to get contents of table 'Info'
ret = mdbExecuteSqlStatement('SELECT * from Info');
if (ret = 1) {
    Error('Execution of SQL query failed');
}
```

mdbExecuteSqlStatement

Executes a SQL statement that does not return any values.

Executing a statement invalidates a previous query if there exists one. Special queries:

- 'SQLTables' to enumerate over all table definitions
- 'SQLColumns \c tablename' to enumerate over all columns of a table

```
int mdbExecuteSqlStatement(string statement)
```

ARGUMENTS

statement

Sql statement as text, e.g. `CREATE TABLE...` .

RETURNS

- 0** On success.
- 1** On error.

EXAMPLE

```
// example to create a new table called 'Info'
ret = mdbExecuteSqlStatement(
    'CREATE TABLE [Info] ([Name] VARCHAR(20), [Version] INTEGER)'
);
if (ret == 1) {
    Error('Execution of SQL statement failed');
}
```

[mdbFetchResult](#)

Fetches next data set returned by previous SQL query.

To get all result sets, this function must be called until 0 is returned.

```
int mdbFetchResult()
```

RETURNS

- 0** On success.
- 1** On error.

EXAMPLE

```
ret = mdbExecuteSqlStatement('SELECT * from Info');
if (ret == 1) {
    Error('Execution of SQL query failed');
    exit();
}
ret = mdbFetchResult();
while(ret == 1) {
    ...
    ret = mdbFetchResult();
}
```

[mdbGetResultColumnCount](#)

Returns the number of data columns a result set has.

All sets of a query have identical number of column counts. Therefore, it is sufficient to get this value only once while iterating over the results. Please note, this function requires that results values have already been fetched via [mdbFetchResult\(\)](#).

```
int mdbGetResultColumnCount()
```

RETURNS

Number of columns in result set (always ≥ 0).

EXAMPLE

```
ret = mdbExecuteSqlStatement('SELECT * from Info');
...
// get first data set
ret = mdbFetchResult();
...
ret = mdbGetResultColumnCount();
printf('Number of columns in result: %d', ret);
```

mdbGetResultColumnName

Returns the field name of a result column.

All sets of a query have identical number of column counts. Therefore, it is sufficient to get this value only once while iterating over the results. Please note, this function requires that results values have already been fetched via [mdbFetchResult\(\)](#).

```
string mdbGetResultColumnName(int column)
```

ARGUMENTS

column Column index, $1 \leq \text{index} \leq \text{mdbGetResultColumnCount()}$.

RETURNS

Name of the column. String is empty if index is out of valid range.

EXAMPLE

```
ret = mdbExecuteSqlStatement('SELECT * from Info');
...
// get first data set
ret = mdbFetchResult();
printf('Number of columns in result: %d', ret);
```

mdbGetResultColumnType

Returns the data type of the colum in result set. Please note, this function requires that results values have already been fetched via [mdbFetchResult\(\)](#).

```
int mdbGetResultColumnType(int column)
```

ARGUMENTS

column Column index, $1 \leq \text{index} \leq \text{mdbGetResultColumnCount()}$.

RETURNS

Data type of values in given column:

- 0** string
- 1** integer
- 2** double

EXAMPLE

```
ret = mdbExecuteSqlStatement('SELECT * from Info');
...
ret = mdbFetchResult();
...
ret = mdbGetResultColumnType(1);
printf('Data type of column 1 is: %d', ret);
```

mdbGetResultColumnValue

Returns the value of a column in current data set.

```
int mdbGetResultColumnValue(int column
                            int& | double& | string& value)
```

ARGUMENTS

column Column index, $1 \leq \text{index} \leq \text{mdbGetResultColumnCount()}$.

value (out)

Output variable. The variable type must match the data type of the column. The only exception is as string: It is allowed to retrieve all values as strings.

RETURNS

0 On success.

1 On error.

EXAMPLE

```
ret = mdbExecuteSqlStatement('SELECT * from Info');
...
//get first data set
ret = mdbFetchResult();
...
ret = mdbGetResultColumnValue(1, str);
printf('Value is %s', str);
```

mdbOpen

Opens an MS Access file.

```
int mdbOpen(string file,
            [int createIfNotExists = 0,]
            [int accessMode = 0])
```

ARGUMENTS

file Full file name of mdb.

createIfNotExists (optional)

- 0** Do not create a new file if it does not exist (default).
- 1** Create a new file if it does not exist.

accessMode (optional)

- 0** Read/write (default).
- 1** Read only.

RETURNS

0 On success.

1 On error.

SEE ALSO

[mdbClose\(\)](#)

EXAMPLE

```
mdbOpen('c:\database.mdb', 1);
```

EXAMPLE

This example demonstrates the creation of a new table and insertion of some values.

```
int error, i, ival;
string sql, s;
double dval;

// open a new database (create if it does not exist)
error = mdbOpen('c:\example.mdb',1);
if (error) {
    Error('Unable to open/create access file');
    mdbClose();
    exit();
}

// create a new table 'MyTable'
sql = 'CREATE TABLE [MyTable] ([Name] VARCHAR(20), [Index] INTEGER,
                           [Value] DOUBLE)';
error = mdbExecuteSqlStatement(sql);
if (error) {
    Error('Table creation failed');
    mdbClose();
    exit();
}

// fill in some data
for(i = 1; i<5; i+= 1) {
    dval = i/2;
    s = sprintf('%f', dval);
    strchng(s, ',', '.'); // replace ',' by '.'
    sql = sprintf('INSERT INTO MyTable ([Name], [Index], [Value])
                  VALUES ("Entry%d", %d, %s)', i, i, s);
    error = mdbExecuteSqlStatement(sql);
    if (error) {
        Error('Insertion of data failed (%s)', sql);
        mdbClose();
        exit();
    }
}

// close database
mdbClose();
```

EXAMPLE

This example demonstrates how to access information about available tables.

```
int error, colcount, i, t;
string sql, s;

// open an existing database
error = mdbOpen('c:\example.mdb',0);
if (error) {
    Error('Unable to open database');
    exit();
}
```

```

// list available tables
sql = 'SQLTables';
error = mdbExecuteSqlQuery(sql);
if (.not. error) {
    // get first result set
    error = mdbFetchResult();
}
if (error) {
    Error('Unable to get table information');
    mdbClose();
    exit();
}

// get number of columns in result set
colcount = mdbGetResultColumnCount();

// output table information
printf('The database contains the following tables:');
SetLineFeed(0); // disable automatic line feed
// header information (column name and type)
for(i =1; i<= colcount; i+=1) {
    if (i>1) {
        printf('; ');
    }
    s = mdbGetResultColumnName(i);
    t = mdbGetResultColumnType(i);
    printf('%s(%d)', s, t);
}
printf('\n');

// output result data sets
while(error = 0) {
    for(i =1; i<= colcount; i+=1) {
        if (i>1) {
            printf('; ');
        }
        mdbGetResultColumnValue(i, s);
        printf('%s', s);
    }
    printf('\n');
    error = mdbFetchResult();
}
SetLineFeed(1); // enable automatic line feed again

// close database
mdbClose();

```

EXAMPLE

This example demonstrates how to get information about the columns of a specific table.

```

int error, colcount, i, t;
string sql, s;

// open an existing database
error = mdbOpen('c:\example.mdb', 0);
if (error) {
    Error('Unable to open database');
    exit();
}

```

```

}

!// list fields of table 'MyTable'
sql = 'SQLColumns MyTable';
error = mdbExecuteSqlQuery(sql);
if (.not. error) {
    !// get first result set
    error = mdbFetchResult();
}
if (error) {
    Error('Unable to get field information for table "MyTable"');
    mdbClose();
    exit();
}

!// get number of columns in result set
colcount = mdbGetResultColumnCount();

!// output table information
printf('The table "MyTable" contains the columns:');
SetLineFeed(0);
!// header information (column name and type)
for(i =1; i<= colcount; i+=1) {
    if (i>1) {
        printf('; ');
    }
    s = mdbGetResultColumnName(i);
    t = mdbGetResultColumnType(i);
    printf('%s(%d)', s, t);
}
printf('\n');

!// output result data sets
while(error = 0) {
    for(i =1; i<= colcount; i+=1) {
        if (i>1) {
            printf('; ');
        }
        mdbGetResultColumnValue(i, s);
        printf('%s', s);
    }
    printf('\n');
    error = mdbFetchResult();
}

SetLineFeed(1);

!// close database
mdbClose();

```

EXAMPLE

This example demonstrates how to read data from an existing table.

```

int error, colcount, i, t;
string sql, s;

!// open an existing database
error = mdbOpen('c:\example.mdb', 0);
if (error) {
    Error('Unable to open database');
}

```

```

    exit();
}

// list fields of table 'MyTable'
sql = 'SELECT * FROM [MyTable]';
error = mdbExecuteSqlQuery(sql);
if (.not. error) {
    // get first result set
    error = mdbFetchResult();
}
if (error) {
    Error('Unable to get data from table "MyTable"');
    mdbClose();
    exit();
}

// get number of columns in result set
colcount = mdbGetResultColumnCount();

// output table information
printf('The table "MyTable" contains the following data:');
SetLineFeed(0);
// header information (column name and type)
for(i =1; i<= colcount; i+=1) {
    if (i>1) {
        printf('; ');
    }
    s = mdbGetResultColumnName(i);
    t = mdbGetResultColumnType(i);
    printf('%s(%d)', s, t);
}
printf('\n');

// output result data sets
while(error = 0) {
    for(i =1; i<= colcount; i+=1) {
        if (i>1) {
            printf('; ');
        }
        mdbGetResultColumnValue(i, s);
        printf('%s', s);
    }
    printf('\n');
    error = mdbFetchResult();
}

SetLineFeed(1);

// close database
mdbClose();

```

mdbSetDebug

Enables output of additional information of errors during communication with MS Access.

The information is printed as plain text to the **PowerFactory** output window.

```
void mdbSetDebug(int debug)
```

ARGUMENTS

debug

- 0** Disable debug mode.
- 1** Enable debug mode.

EXAMPLE

```
mdbSetDebug(1); // enable debug info for MS Access communication
```

2.6.2 MS Excel

Overview

xlActivateWorksheet
 xlAddWorksheet
 xlCloseWorkbook
 xlDeleteWorksheet
 xlGetActiveWorksheetIndex
 xlGetDateSeparator
 xlGetDecimalSeparator
 xlGetThousandsSeparator
 xlGetValue
 xlGetWorksheetCount
 xlGetWorksheetName
 xlNewWorkbook
 xlOpenWorkbook
 xlResetTextStyle
 xlRunMacro
 xlSaveWorkbook
 xlSaveWorkbookAs
 xlSetBorder
 xlSetColumnWidth
 xlSetDebug
 xlSetFillColor
 xlSetFontName
 xlFontSize
 xlSetHorizontalAlignment
 xlSetNumberFormat
 xlSetPrintTitleRows
 xlSetRowHeight
 xlSetTextColor
 xlSetTextStyle
 xlSetValue
 xlSetValues
 xlSetVerticalAlignment
 xlSetVisible
 xlSetWorksheetName
 xlSetWrapText
 xlStart
 xlTerminate

xlActivateWorksheet

Activates sheet with given index in active workbook.

```
int xlActivateWorksheet (int sheetIndex)
```

ARGUMENTS

sheetIndex

Index of the sheet that should become active. This index is 1-based, this means $1 \leq \text{sheetIndex} \leq \text{xlGetWorksheetCount}()$.

RETURNS

- 0** On success.
- 1** On error.

SEE ALSO

[xlGetWorksheetCount\(\)](#)

EXAMPLE

```
xlActivateSheet(1); // activates first sheet
```

xlAddWorksheet

Adds a new worksheet to current workbook. The new worksheet will automatically be set to be the active one.

```
int xlAddWorksheet ([string name])
```

ARGUMENTS

name (optional)

Name for new worksheet.

RETURNS

- 0** On success.
- 1** On error.

EXAMPLE

```
xlAddWorksheet();
```

xlCloseWorkbook

Closes currently opened workbook. Any unsaved modifications will be lost.

```
int xlCloseWorkbook ()
```

RETURNS

- 0** On success.
- 1** On error.

SEE ALSO

[xlSaveWorkbookAs\(\)](#), [xlSaveWorkbook\(\)](#)

EXAMPLE

```
xlCloseWorkbook();
```

xlDeleteWorksheet

Deletes a worksheet from current workbook.

```
int xlDeleteWorksheet(int sheetIndex)
```

ARGUMENTS

sheetIndex

Index of sheet to delete. $1 \leq \text{sheetIndex} \leq \text{GetWorksheetCount}()$.

RETURNS

- 0** On success.
- 1** On error.

EXAMPLE

```
int count;
// delete all worksheets (except last one = error)
count = xlGetWorksheetCount();
while(count > 0) {
    error = xlDeleteWorksheet(count);
    if (error == 0) {
        printf('Successfully deleted sheet %d', count);
    }
    else {
        printf('Sheet %d could not be deleted', count);
        break;
    }
    count -= 1;
}
```

xlGetActiveWorksheetIndex

Returns the index of currently active sheet.

```
int xlGetActiveWorksheetIndex()
```

RETURNS

Index of active worksheet, $1 \leq \text{sheetIndex} \leq \text{xlGetWorksheetCount}()$.

SEE ALSO

[xlGetWorksheetCount\(\)](#)

xlGetDateSeparator

Returns currently used date separator.

```
string xlGetDateSeparator()
```

RETURNS

Data separator, e.g. "/".

EXAMPLE

```
sep = xlGetDateSeparator();
```

xlGetDecimalSeparator

Returns currently used decimal separator.

```
string xlGetDecimalSeparator()
```

RETURNS

Decimal separator, e.g. ",".

EXAMPLE

```
sep = xlGetDecimalSeparator();
```

xlGetThousandsSeparator

Returns currently used thousands separator.

```
string xlGetThousandsSeparator()
```

RETURNS

Thousands separator, e.g. "..".

EXAMPLE

```
sep = xlGetThousandsSeparator();
```

xlGetValue

Returns the value of a cell.

```
int xlGetValue(int column,
              int row,
              int|double|string value)
```

ARGUMENTS

column Column index of cell (≥ 1).

row Row index of cell (≥ 1).

value (out)

Variable in which the output will be stored. It is always possible to get a cell value as strings. For other data types, the type of the variable must match that of the cell value.

RETURNS

- 0** On success.
- 1** On error.

SEE ALSO

[xlSetValue\(\)](#)

EXAMPLE

```
string value;
xlGetValue(1, 1, value);
printf('The value of Cell A1 is %s', value);
```

xlGetWorksheetCount

Returns the number of worksheets in current workbook.

```
int xlGetWorksheetCount()
```

RETURNS

Number of sheets (always ≥ 0).

EXAMPLE

```
count = xlGetWorksheetCount();
printf('Current Workbook contains %d sheets', count);
```

xlGetWorksheetName

Gets the name of a worksheet (in active workbook).

```
string xlGetWorksheetName(int sheetIndex)
```

ARGUMENTS**sheetIndex**

Index of sheet for which the name shall be returned. This index is 1-based, this means $1 \leq \text{sheetIndex} \leq \text{xlGetWorksheetCount}()$.

RETURNS

Name of sheet or empty in case sheet does not exist.

SEE ALSO

[xlGetWorksheetCount\(\)](#), [xlSetWorksheetName\(\)](#)

EXAMPLE

```
count = xlGetWorksheetCount();
printf('Number of sheets in current workbook: %d', count);
while(count > 0) {
    name = xlGetWorksheetName(count);
    printf('Worksheet[%d]: Name=%s', count, name);
    count -= 1;
}
```

xlNewWorkbook

Creates a new Workbook.

```
int xlNewWorkbook()
```

RETURNS

- 0** On success.
- 1** On error.

EXAMPLE

```
xlNewWorkbook();
```

xlOpenWorkbook

Opens an existing workbook.

```
int xlOpenWorkbook(string file)
```

ARGUMENTS

- name** Name of existing MS Excel file to open.

RETURNS

- 0** On success.
- 1** On error.

EXAMPLE

```
ret = xlOpenWorkbook('c:\test.xlsx'); // opens c:\test.xlsx
```

xlResetTextStyle

Resets given text style for a cell or for a range of cells.

Note: If column2 and row2 are given, the text style is changed for the whole range from column1, row1 to column2, row2.

```
int xlResetTextStyle(int column1,
                     int row1,
                     int style)
```

```
int xlResetTextStyle(int column1,
                     int row1,
                     int column2,
                     int row2,
                     int style)
```

ARGUMENTS

column1 Column index (≥ 1).

row1 Row index (≥ 1).

column2 (optional)

2nd column index for specifying a range (≥ 1).

row2 (optional)

2nd row index for specifying a range (≥ 1).

style Text style to be reset:

- 1** bold
- 2** italic
- 4** underline
- 8** strikethrough
- 16** superscript
- 32** subscript

Note, multiple styles can be combined by summing up the corresponding style values, e.g. bold and italic is 3 (=1+2)

RETURNS

0 On success.

1 On error.

SEE ALSO

[xlSetTextStyle\(\)](#)

EXAMPLE

```
xlResetTextStyle(1,1,3,2,1); // no bold for range A1:B2
xlResetTextStyle(5,2,6); // no italic and no underline for cell D2
```

xlRunMacro

Executes a macro.

```
int xlRunMacro(string macro)
```

ARGUMENTS

macro Macro name; if a macro of that name does not exist, a value of 1 is returned.

createIfNotExists (optional)

- 0** Do not create a new file if it does not exist (default).
- 1** Create a new file if it does not exist.

accessMode (optional)

- 0** Read/write (default).
- 1** Read only.

RETURNS

- 0** On success.
- 1** On error.

This is not the return value of the macro itself but just an indicator whether macro has been found an executed.

EXAMPLE

```
xlRunMacro('MyMacro');
```

xlSaveWorkbook

Saves a modified workbook. The existing file will be overwritten with current version of the workbook. Please note, for new workbooks the SaveAs() function has to be used.

```
int xlSaveWorkbook();
```

RETURNS

- 0** On success.
- 1** On error.

SEE ALSO

[xlSaveWorkbookAs\(\)](#)

EXAMPLE

```
xlSaveWorkbook(); // overwrites existing file with current version
                  // of workbook
```

xlSaveWorkbookAs

Saves current workbook as a new file.

```
int xlSaveWorkbookAs(string file)
```

ARGUMENTS

file Full file name of new MS Excel file.

RETURNS

- 0** On success.
- 1** On error.

EXAMPLE

```
ret = xlSaveWorkbookAs('c:\test_new.xlsx'); // save current workbook as
                                         // c:\test_new.xlsx
```

xlSetBorder

Sets/resets the border of a cell or a range of cells.

To reset a border, use `lineStyle=none`. In this case, the given `weight` and `color` is ignored.

```
int xlSetBorder(int column1,
                int row1,
                int borders,
                int lineStyle,
                int weight,
                int colorR,
                int colorG,
                int colorB)
int xlSetBorder(int column1,
                int row1,
                int column2,
                int row2,
                int borders,
                int lineStyle,
                int weight,
                int colorR,
                int colorG,
                int colorB)
```

ARGUMENTS

column1 Column index (≥ 1).

row1 Row index (≥ 1).

column2 (optional)

2nd column index for specifying a range (≥ 1).

row2 (optional)

2nd row index for specifying a range (≥ 1).

borders Identifier of border, possible values are

- 1** edge bottom
- 2** edge right
- 4** edge top
- 8** edge left
- 16** inside horizontal
- 32** inside vertical
- 64** diagonal down
- 128** diagonal up

lineStyle Style of the line, possible values are

- 0** none (resets the border)
- 1** continuous
- 2** dash
- 3** dash dot
- 4** dash dot dot
- 5** dot
- 6** double
- 7** slant dash dot

weight Weight of the border, possible values are

- 1** hairline
- 2** medium
- 3** thick
- 4** thin

colorR Red part of RGB color, $0 \leq \text{colorR} \leq 255$.

colorG Green part of RGB color, $0 \leq \text{colorG} \leq 255$.

colorB Blue part of RGB color, $0 \leq \text{colorB} \leq 255$.

RETURNS

- 0** On success.
- 1** On error.

EXAMPLE

```
xlSetBorder(2,2,15,1,2,0,0,0); // sets a border around B2,
                                // bottom+right+top+left
xlSetBorder(2,2,15,0,0,0,0,0); // resets border for B2
```

xlSetColumnWidth

Sets the width of a given column in active worksheet.

```
int xlSetColumnWidth(int column,
                     double width)
```

ARGUMENTS

column Column index (≥ 1).

width New width. If a value < 0 is passed, the optimal width will be automatically detected ('autofit').

RETURNS

- 0** On success.
- 1** On error.

EXAMPLE

```
xlSetColumnWidth(1, 20.25); // set width of column 1 to 20.25
```

xlSetDebug

Enables output of additional information of errors during communication with MS Excel.

The information is printed as plain text to the **PowerFactory** output window.

Note: This method must not be called before xlStart if the alternative Excel implementation is used.

```
void xlSetDebug(int debug)
```

ARGUMENTS

debug

- 0** Disable debug mode.
- 1** Enable debug mode.

EXAMPLE

```
xlSetDebug(1); // enable debug info for MS Excel communication
```

xlSetFillColor

Sets the background color for a cell or a range of cells. The color must be given in RGB parts.

Note: If column2 and row2 are given, the text style is changed for the whole range from column1, row1 to column2, row2.

```
int xlSetFillColor(int column1,
                   int row1,
                   int colorR,
                   int colorG,
                   int colorB)
int xlSetFillColor(int column1,
                   int row1,
                   int column2,
                   int row2,
                   int colorR,
                   int colorG,
                   int colorB)
```

ARGUMENTS

column1 Column index (≥ 1).*row1* Row index (≥ 1).*column2 (optional)*2nd column index for specifying a range (≥ 1).*row2 (optional)*2nd row index for specifying a range (≥ 1).*colorR* Red part of RGB color, $0 \leq \text{colorR} \leq 255$.*colorG* Green part of RGB color, $0 \leq \text{colorG} \leq 255$.*colorB* Blue part of RGB color, $0 \leq \text{colorB} \leq 255$.

RETURNS

0 On success.**1** On error.

EXAMPLE

```
xlSetFillColor(1,1,3,2,255, 255, 150); // range A1:B2 to yellow
```

xlSetFontName

Sets a new text font for a cell or a range of cells.

```
int xlSetFontName(int column1,
                  int row1,
                  string fontname)
int xlSetFontName(int column1,
                  int row1,
                  int column2,
                  int row2,
                  string fontname)
```

ARGUMENTS

column1 Column index (≥ 1).

row1 Row index (≥ 1).

column2 (optional)

2nd column index for specifying a range (≥ 1).

row2 (optional)

2nd row index for specifying a range (≥ 1).

fontname Windows font name, e.g. "Arial".

RETURNS

0 On success.

1 On error.

EXAMPLE

```
xlSetFontName(1, 1, 'Courier'); // text will now be visualized
                                // in font 'Courier'
```

xlSetFontSize

Sets a new size for text font of a cell or a range of cells.

```
int xlSetFontSize(int column1,
                  int row1,
                  double fontsize)
int xlSetFontSize(int column1,
                  int row1,
                  int column2,
                  int row2,
                  double fontsize)
```

ARGUMENTS

column1 Column index (≥ 1).

row1 Row index (≥ 1).

column2 (optional)

2nd column index for specifying a range (≥ 1).

row2 (optional)

2nd row index for specifying a range (≥ 1).

fontsize Size, e.g. 12.0.

RETURNS

- 0** On success.
- 1** On error.

EXAMPLE

```
xlSetFontSize(1,1,9.75);
```

xlSetHorizontalAlignment

Sets the horizontal content alignment for a cell or a range of cells.

Note: If column2 and row2 are given, the alignment is changed for the whole range from column1, row1 to column2, row2.

```
int xlSetHorizontalAlignment(int column1,
                           int row1,
                           int alignment)
int xlSetHorizontalAlignment(int column1,
                           int row1,
                           int column2,
                           int row2,
                           int alignment)
```

ARGUMENTS

column1 Column index (≥ 1).

row1 Row index (≥ 1).

column2 (optional)

2nd column index for specifying a range (≥ 1).

row2 (optional)

2nd row index for specifying a range (≥ 1).

alignment

New horizontal alignment. Possible values are:

- 0** left
- 1** center
- 2** right

RETURNS

- 0** On success.
- 1** On error.

EXAMPLE

```
xlSetHorizontalAlignment(1,1,2); // set horizontal text alignment
                               // for A1 to center
```

xlSetNumberFormat

Sets the number format for a cell or a range of cells. Please note that decimal, date separators are localized and must be used according to current settings.

```
int xlSetNumberFormat(int column1,
                      int row1,
                      string format)
int xlSetNumberFormat(int column1,
                      int row1,
                      int column2,
                      int row2,
                      string format)
```

ARGUMENTS

column1 Column index (≥ 1).

row1 Row index (≥ 1).

column2 (optional)

2nd column index for specifying a range (≥ 1).

row2 (optional)

2nd row index for specifying a range (≥ 1).

format New number format, e.g. "0.##".

RETURNS

0 On success.

1 On error.

EXAMPLE

```
xlSetValue(3.1415, 1, 1);
xlSetNumberFormat(1, 1, '0,0'); // value will now be displayed as '3.1'
```

xlSetPrintTitleRows

Allows to set fixed header rows for printing. The corresponding setting in Excel is called "rows to repeat on top" and can be found in 'Page Setup' on tab 'Sheet'.

Calling this function with $\text{row1}=\text{row2}=-1$ will reset the setting.

```
int xlSetPrintTitleRows(int row1,
                       int row2)
```

ARGUMENTS

row1 First row index, -1 or ≥ 1 .

row2 Second row index, -1 or ≥ 1 ; $\text{row2} \geq \text{row1}$.

RETURNS

0 On success.

1 On error.

EXAMPLE

```
xlSetPrintTitleRows(1,1); // row1 will now be printed on
                         // each page (on printout)
```

xlSetRowHeight

Sets the height of a given row in active worksheet.

```
int xlSetRowHeight(int row,
                   double height)
```

ARGUMENTS

row Row index (≥ 1).

height New height. If a value < 0 is passed, the optimal height will be automatically detected ('autofit').

RETURNS

0 On success.

1 On error.

EXAMPLE

```
xlSetRowHeight(1, 10); //set height of row 1 10
```

xlSetTextColor

Sets the text color for a cell or a range of cells. The color must be given in RGB parts.

Note: If column2 and row2 are given, the text style is changed for the whole range from column1, row1 to column2, row2.

```
int xlSetTextColor(int column1,
                   int row1,
                   int colorR,
                   int colorG,
                   int colorB)
int xlSetTextColor(int column1,
                   int row1,
                   int column2,
                   int row2,
                   int colorR,
                   int colorG,
                   int colorB)
```

ARGUMENTS

column1 Column index (≥ 1).

row1 Row index (≥ 1).

column2 (optional)

2nd column index for specifying a range (≥ 1).

row2 (optional)

2nd row index for specifying a range (≥ 1).

colorR Red part of RGB color, $0 \leq \text{colorR} \leq 255$.

colorG Green part of RGB color, $0 \leq \text{colorG} \leq 255$.

colorB Blue part of RGB color, $0 \leq \text{colorB} \leq 255$.

RETURNS

0 On success.

1 On error.

EXAMPLE

```
xlSetTextColor(1,1,3,2,255,0,0); // range A1:B2 to red
```

xlSetTextStyle

Sets given text style for a cell or for a range of cells.

Note: If col2 and row2 are given, the text style is changed for the whole range from col1, row1 to col2, row2.

The formatting can be undone using function [xlResetTextStyle\(\)](#).

```
int xlSetTextStyle(int column1,
                  int row1,
                  int style)
int xlSetTextStyle(int column1,
                  int row1,
                  int column2,
                  int row2,
                  int style)
```

ARGUMENTS

column1 Column index (≥ 1).

row1 Row index (≥ 1).

column2 (optional)

2nd column index for specifying a range (≥ 1).

row2 (optional)

2nd row index for specifying a range (≥ 1).

style Text style to be set:

1 bold

2 italic

4 underline

8 strikethrough

16 superscript

32 subscript

Note, multiple styles can be combined by summing up the corresponding style values, e.g. bold and italic is 3 (=1+2)

RETURNS

- 0** On success.
- 1** On error.

SEE ALSO

[xlResetTextStyle\(\)](#)

EXAMPLE

```
xlSetTextStyle(1,1,3,2,1); // range A1:B2 to bold
xlSetTextStyle(5,2,6); // cell D2 to italic+underline
```

xlSetValue

Sets a cell's value.

```
int xlSetValue(int column,
              int row,
              int|double|string value)
```

ARGUMENTS

- column* Column index of cell (≥ 1).
- row* Row index of cell (≥ 1).
- value* New value to be set.

RETURNS

- 0** On success.
- 1** On error.

SEE ALSO

[xlGetValue\(\)](#), [xlSetNumberFormat\(\)](#)

EXAMPLE

```
xlSetValue(1, 1, 'My text'); // sets text for cell A1
```

xlSetValues

Sets values for a row of cells.

```
int xlSetValues(int column,
                int row,
                string values,
                string sep)
```

ARGUMENTS

- column* Column index of cell (≥ 1).
- row* Row index of cell (≥ 1).
- values* New values separated by 'sep'.
- sep* Used separator.

RETURNS

- 0** On success.
- 1** On error.

SEE ALSO

[xlSetValue\(\)](#)

EXAMPLE

```
xlSetValues(1, 1, 'Hello|World', '|'); // sets text for cell
                                         // A1=Hello, A2=World
```

xlSetVerticalAlignment

Sets the vertical content alignment for a cell or a range of cells.

Note: If column2 and row2 are given, the alignment is changed for the whole range from column1, row1 to column2, row2.

```
int xlSetVerticalAlignment(int column1,
                           int row1,
                           int alignment)
int xlSetVerticalAlignment(int column1,
                           int row1,
                           int column2,
                           int row2,
                           int alignment)
```

ARGUMENTS

column1 Column index (≥ 1).

row1 Row index (≥ 1).

column2 (optional)

2nd column index for specifying a range (≥ 1).

row2 (optional)

2nd row index for specifying a range (≥ 1).

alignment

New vertical alignment. Possible values are:

- 0** top
- 1** center
- 2** bottom

RETURNS

- 0** On success.
- 1** On error.

EXAMPLE

```
xlSetVerticalAlignment(1,1,2); // set vertical text alignment
                             // for A1 to center
```

xlSetVisible

Sets visibility of MS Excel application window. By default, the window is hidden.

```
int xlSetVisible(int visible)
```

ARGUMENTS

visible)

- 0** Makes MS Excel invisible.
- 1** Makes MS Excel visible.

RETURNS

- 0** On success.
- 1** On error.

EXAMPLE

```
xlSetVisbile(1); // makes application window visible
```

xlSetWorksheetName

Sets the name of a worksheet (in active workbook).

```
int xlSetWorksheetName(int sheetIndex,
                      string name)
```

ARGUMENTS

sheetIndex

Index of sheet for which the name shall be set. This index is 1-based, this means $1 \leq \text{sheetIndex} \leq \text{xlGetWorksheetCount}()$.

name New name to be set.

RETURNS

- 0** On success.
- 1** On error.

SEE ALSO

[xlGetWorksheetCount\(\)](#), [xlGetWorksheetName\(\)](#)

EXAMPLE

```
count = xlGetWorksheetCount();
printf('Number of sheets in current workbook: %d', count);
while(count > 0) {
    name = sprintf('My Sheet %d', count);
    xlSetWorksheetName(count, name);
    count -= 1;
}
```

xlSetWrapText

Enables or disables text wrapping for a cell or range of cells.

```
int xlSetWrapText(int column1,
                  int row1,
                  int enabled)
int xlSetWrapText(int column1,
                  int row1,
                  int column2,
                  int row2,
                  int enabled)
```

ARGUMENTS

column1 Column index (≥ 1).

row1 Row index (≥ 1).

column2 (optional)

2nd column index for specifying a range (≥ 1).

row2 (optional)

2nd row index for specifying a range (≥ 1).

enabled

0 Wrapping disabled.

1 Wrapping enabled.

RETURNS

0 On success.

1 On error.

EXAMPLE

```
xlSetWrapText(1,1,1); // enabled text wrapping for A1
xlSetValue(1,1,'Hello\nExcel'); // set a text that contains
                                // an explicit line break
```

xlStart

Creates a new MS Excel instance. This function must be called once at the beginning of any communication with MS Excel.

The interface will per default use the COM interface Microsoft Excel provides. This requires Excel to be installed on the machine. Passing 1 for the type parameter uses an alternative implementation which does not require Excel to be installed. However, the alternative implementation cannot handle macro-enabled Excel files (.xlsm).

```
int xlStart([int interface = 0])
```

ARGUMENTS

interface (optional)

0= Use COM, 1= Use alternative implementation

RETURNS

0 On success.

1 On error.

SEE ALSO

[xlTerminate\(\)](#)

EXAMPLE

```
xlStart(); // starts MS Excel
...
xlTerminate();
```

EXAMPLE

This example demonstrates how to export data of PowerFactory elements into an Excel sheet.

```
string class, attributes;
string s, desc, type, sval, sep, numberFormat;
set objs;
object obj, oval;
double dval;
int error, pos, i, t, row, col, maxRow, maxCol;

// export definition
class = 'ElmLine';
attributes = 'loc_name,typ_id,bus1,bus2,dline';

error = xlStart(); // start MS Excel
if (error) {
    Error('Unable to start MS Excel application');
    exit();
}

// get decimal separator and build number format used here
sep = xlGetDecimalSeparator();
numberFormat = sprintf('0%s000', sep);

// create a new workbook
xlNewWorkbook();
xlSetWorksheetName(1, class);

// iterate over attributes and write header row
row = 1;
col = 1;
s = strtok(attributes, ',', pos, col);
while(pos > -1) {
    xlSetValue(col, row, s);
    col+=1;
    s = strtok(attributes, ',', pos, col);
}
maxCol = col-1;

// change format of header row
xlSetTextStyle(1,1,maxCol,1, 1); // bold
xlSetFillColor(1,1,maxCol,1, 255, 255, 150); // yellow
xlSetBorder(1,1,maxCol,1,1,1,2,0,0,0); // border at bottom

// export data
row = 2;
objs = AllRelevant(class);
for(obj = objs.First(); obj; obj = objs.Next()) {
    col = 1;
```

```

s = strtok(attributes,',',pos,col);
while(pos > -1) {
    obj.GetVarType(s, type);
    t = strcmp(type, 'string');
    if (t==0) {
        obj.GetVal(sval, s);
        xlSetValue(col, row, sval);
    }
    t = strcmp(type, 'object');
    if (t==0) {
        obj.GetVal(oval, s);
        sval = oval:loc_name;
        xlSetValue(col, row, sval);
    }
    t = strcmp(type, 'double');
    if (t==0) {
        obj.GetVal(dval, s);
        xlSetValue(col, row, dval);
        xlSetNumberFormat(col, row, numberFormat);
    }

    col+=1;
    s = strtok(attributes,',',pos,col);
}
row += 1;
}

// save and exit
error = xlSaveWorkbookAs('c:\export.xls');
if (error) {
    Error('Workbook could not be saved');
}
xlTerminate(); // terminate MS Excel

```

EXAMPLE

This example demonstrates how to open an Excel file and read values from active sheet.

```

int error, row, col, count, active, i, t, stop;
string str;

error = xlStart(); // start MS Excel
if (error) {
    Error('Unable to start MS Excel application');
    exit();
}

error = xlOpenWorkbook('c:\test.xls'); // opens c:\test.xls
if (error) {
    Error('Unable to open Excel file.');
    xlTerminate();
    exit();
}

// get number of sheets
count = xlGetWorksheetCount();
active = xlGetActiveWorksheetIndex();
printf('The workbook contains the following sheets:');
for(i = 1; i<=count; i+= 1) {
    str = xlGetWorksheetName(i);
    if (i == active) {

```

```

        printf(' %d: %s (active)', i, str);
    }
else {
    printf(' %d: %s', i, str);
}
}

// get cell values starting at A1 up to first empty cell
printf('Listing contents of current sheet:');
row = 1;
stop = 0;
while(stop == 0) {
    col = 1;
    while(1) {
        xlGetValue(col, row, str);
        t = strlen(str);
        if (t == 0) { // stop at empty cell, continue with next row
            if (col == 1) {
                stop = 1; // completely stop if cell in first column is empty
            }
            break;
        }
        printf('row: %d, col: %d, value: %s', row, col, str);
        col += 1;
    }
    row += 1;
}

xlTerminate(); // terminate MS Excel

```

xlTerminate

Closes currently active MS Excel instance. This function should be called at the end of a script if all communication with MS Excel is finished.

```
int xlTerminate()
```

RETURNS

- 0** On success.
- 1** On error.

SEE ALSO

[xlStart\(\)](#)

EXAMPLE

```

xlStart();
...
xlTerminate(); // closes MS Excel

```

2.7 Multibyte Encoded String

Overview

`mbschg*`
`mbscmp*`
`mbscpy*`
`mbslen*`
`mbsprintf*`
`mbsscanf*`
`mbsstr*`
`mbstok*`
`tombchar*`
`tombcharcodepoint*`
`tombslower*`
`tombsupper*`

mbschg*

Searches a substring and replaces it. See [strchg\(\)](#) for more details.

```
int mbschg(string& str,
            string find,
            string new)
```

SEE ALSO

[strchg\(\)](#)

mbscmp*

Compares two strings case sensitive. See [strcmp\(\)](#) for more details.

```
int mbscmp(string str1,
            string str2,
            [int maxCount])
```

SEE ALSO

[strcmp\(\)](#)

mbscpy*

Returns a copy of a substring. See [strcpy\(\)](#) for more details.

```
string mbscpy(string str,
                int startIndex,
                [int maxCount])
```

SEE ALSO

[strcpy\(\)](#)

mbslen*

Returns the multibyte length of a string.

```
int mbslen(string str)
```

ARGUMENTS

str The string.

RETURNS

Returns the number of multibyte characters of a string. For strings containing invalid multibyte characters 0 is returned.

SEE ALSO

[strlen\(\)](#)

mbsprintf*

Returns a formatted multibyte character string. See [sprintf\(\)](#) for more details.

```
string mbsprintf(string format,
                  [int|double|string|object argument0,
                   [...])
```

SEE ALSO

[mbprintf\(\)](#), [mbsscanf\(\)](#)

mbsscanf*

Parses a multibyte character string of values. See [sscanf\(\)](#) for more details.

```
int mbsscanf(string source,
             string format,
             [int|double|string argument0,
              [...])
```

SEE ALSO

[sscanf\(\)](#), [mbsprintf\(\)](#)

mbsstr*

Searches for a substring in a string and returns its multibyte position. See [strstr\(\)](#) for more details.

```
int mbsstr(string str,
            string substr)
```

SEE ALSO

[strstr\(\)](#)

mbstok*

Splits a string into tokens. See [strtok\(\)](#) for more details.

```
string mbstok(string source,
              string delimiters,
              int& index,
              [int tokenNumber = 1],
              [int skipEmptyToken = 1])
```

SEE ALSO

[strtok\(\)](#)

tombchar*

Converts the given codepoint (number) to the corresponding multibyte character.

```
string tombchar(int codePoint)
```

ARGUMENTS

codePoint

Number between 0 and 65535 which represents a multibyte character in the current Windows ANSI code page.

RETURNS

A string of multibyte length 1 containing the multibyte character representation of the given codepoint in the current Windows ANSI code page or an empty string if no multibyte character representation exists.

SEE ALSO

[tombcharcodepoint\(\)](#), [tochar\(\)](#)

tombcharcodepoint*

Converts a multibyte character to its code point.

```
int tombcharcodepoint(string str)
```

ARGUMENTS

str A string of multibyte length 1 containing the multibyte character.

RETURNS

Number between 0 and 65535 which represents the given multibyte character in the current Windows ANSI code page or 0 if an error occurred.

SEE ALSO

[tombcharcodepoint\(\)](#), [tochar\(\)](#)

tombslower*

Creates a lowercased copy of a string. See [tolower\(\)](#) for more details.

```
tombslower(string str)
```

SEE ALSO

[tombsupper\(\)](#), [tolower\(\)](#)

tombsupper*

Creates a uppercased copy of a string. See [toupper\(\)](#) for more details.

```
tombsupper(string str)
```

SEE ALSO

[tombslower\(\)](#), [toupper\(\)](#)

2.8 Output Window

Overview

[ClearOutputWindow*](#)
[Error*](#)
[Info*](#)
[mbprintf*](#)
[printf*](#)
[SetLineFeed*](#)
[SetOutputWindowState](#)
[Warn*](#)
[Write*](#)

ClearOutputWindow*

Clears the output window.

```
void ClearOutputWindow()
```

DEPRECATED NAMES

ClearOutput

Error*

Prints a formatted string as an error to the Output Window and automatically inserts a line-break.

The DPL script stops with an appropriate error when the passed arguments don't match the format string. The line-break can't be disabled.

To stop a script after an error use [exit\(\)](#).

```
string Error(string format,
            [int|double|string|object argument0,]
            [...] )
```

ARGUMENTS

format C++-like printf() format string. See the [Format String Syntax](#) for more information.

argument... (optional)

Arguments matching the format string.

RETURNS

The printed string.

SEE ALSO

[printf\(\)](#), [Info\(\)](#), [Warn\(\)](#)

EXAMPLE

The following example writes an error to the output window.

```
Error('Index could not be calculated.');
```

Info*

Prints a formatted string as information to the Output Window and automatically inserts a line-break.

The DPL script stops with an appropriate error when the passed arguments don't match the format string. The line-break can't be disabled.

```
string Info(string format,
           [int|double|string|object argument0,]
           [...])
```

ARGUMENTS

format C++-like printf() format string. See the [Format String Syntax](#) for more information.

argument... (optional)
Arguments matching the format string.

RETURNS

The printed string.

SEE ALSO

[printf\(\)](#), [Warn\(\)](#), [Error\(\)](#)

EXAMPLE

The following example writes an info message to the output window.

```
Info('Trying to calculate first index...');
```

mbprintf*

Prints a formatted multibyte string as plain text to the Output Window. See [printf\(\)](#) for more details.

To print a formatted multibyte string as information, warning or error use [mbsprintf\(\)](#) together with [Info\(\)](#), [Warn\(\)](#) or [Error\(\)](#).

```
string mbprintf(string format,
                [int|double|string|object argument0,]
                [...]);
```

SEE ALSO

[Info\(\)](#), [Warn\(\)](#), [Error\(\)](#), [mbsprintf\(\)](#)

printf*

Prints a formatted string as plain text to the Output Window and automatically inserts a line-break.

The DPL script stops with an appropriate error when the passed arguments don't match the format string. The line-break insertion can be disabled with [SetLineFeed\(\)](#).

```
string printf(string format,
              [int|double|string|object argument0,]
              [...])
```

ARGUMENTS

format C++-like printf() format string. See the [Format String Syntax](#) for more information.

argument... (optional)

Arguments matching the format string.

RETURNS

The printed string.

DEPRECATED NAMES

fWrite

SEE ALSO

[Info\(\)](#), [Warn\(\)](#), [Error\(\)](#), [mbprintf\(\)](#), [sprintf\(\)](#)

EXAMPLE

```
object load;

!load = ...; ! a load

! Print object link (clickable loc_name) without class icon
printf('%s', load);
! Print class icon and object link (clickable loc_name)
printf('%o', load);
! Limit or extend length of class icon + loc_name to 5 characters
printf('%5o', load);

! Print bitmap for object
printf('%b', load);
! Print bitmap for class name
printf('%b', 'ElmLoad');

! A table
printf('\n\n0123456789 0123456789\n');
printf('%10F %10F', -3.143458903850, 1.71);
printf('%10F %10F', 1234567890, 1234567890123);
printf('%10F %10F', 1.0, 1234.0005);
```

SetLineFeed*

Sets or resets the automatic line feed for `printf()`, `mbprintf()` and `fprintf()`. By default, the automatic line feed is enabled.

```
void SetLineFeed(int enabled)
```

ARGUMENTS

enabled ‘0’ disables automatic line feed, ‘1’ enables it.

EXAMPLE

The following example demonstrates the output of two `printf` statements with and without automatic line feed.

```
SetLineFeed(1); !can be omitted, as automatic line feed is enabled by default
printf('Hello ');
printf('PowerFactory');

SetLineFeed(0);
printf('Hello ');
printf('PowerFactory');
printf('\n'); !explicit newline
```

SEE ALSO

[printf\(\)](#), [mbprintf\(\)](#), [fprintf\(\)](#)

SetOutputWindowState

Changes the display state of the output window.

```
void SetOutputWindowState(int newState)
```

ARGUMENTS

newState

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

EXAMPLE

The following example shows how to change the display state:

```
!minimize output window
SetOutputWindowState(0);
printf('Window was minimized');
Sleep(1000);

!maximize output window
SetOutputWindowState(1);
printf('Window was maximized');
Sleep(1000);

!restore previous state
SetOutputWindowState(-1);
printf('Previous window state was restored');
```

Warn*

Prints a formatted string as warning to the Output Window and automatically inserts a line-break.

The DPL script stops with an appropriate error when the passed arguments don't match the format string. The line-break can't be disabled.

```
string Warn(string format,
            [int|double|string|object argument0,]
            [...])
```

ARGUMENTS

format C++-like printf() format string. See the [Format String Syntax](#) for more information.

argument... (*optional*)
Arguments matching the format string.

RETURNS

The printed string.

SEE ALSO

[printf\(\)](#), [Info\(\)](#), [Error\(\)](#)

EXAMPLE

The following example writes a warning message to the output window.

```
Warn('No loads attached: using approximation.');
```

Write*

This function is described here for compatibility reasons. In most cases the [printf\(\)](#) is easier to use. It writes out a line of formatted text, using the *DlgSILENT* output language.

```
void Write(string format,
           [object obj,]
           [...])
void Write(string format,
           [set objects,]
           [...])
```

ARGUMENTS

format The format string

obj (*optional*)
A object which is used to get data from.

objects (*optional*)
Objects which are used to get data from.

This function is used to quickly output a line of formatted output, using the same formatting language as is used for defining reports and result-boxes (see **PowerFactory** User Manual).

Because data or parameters of more than one object are often written out, the *DlgSILENT* output language has the special macro “ACC(x)” to distinguish between these objects. Prior to execution, all given objects and all objects in the given sets are listed together in a single list.

The “ACC(x)” macro returns the object with the index “x” in that list. The ACC (“acc”=“access”) macro can be used more than once for the same object.

Interface variables of the DPL script can also be used in the format string by the “DEF” macro. If the DPL script has “ResX” as an interface double, then “DEF:ResX” will access that variable.

SEE ALSO

[printf\(\)](#)

EXAMPLE

In the following example, two name and loading of two lines written to the Output Window.

```
string format;
set objects;

objects.Add(LineA); ! some ElmLne later ACC(1)
objects.Add(LineB); ! another ElmLne later ACC(2)

Write('The following results are found:');
format += '# : #.## # , # : #.## # ';
format += '$N,ACC(1):loc_name,ACC(1):c:loading,[ACC(1):c:loading,';
format += 'ACC(2):loc_name,ACC(2):c:loading,[ACC(2):c:loading';
Write(format, objects);
```

2.9 String

Overview

- [sprintf*](#)
- [sscanf*](#)
- [sscanfsep*](#)
- [strchg*](#)
- [strcmp*](#)
- [strcpy*](#)
- [strlen*](#)
- [strstr*](#)
- [strtok*](#)
- [tochar*](#)
- [tocharcodepoint*](#)
- [tolower*](#)
- [toupper*](#)

Format String Syntax

The functions [printf\(\)](#), [mbprintf\(\)](#), [sprintf\(\)](#), [mbsprintf\(\)](#), [fprintf\(\)](#) as well as [Error\(\)](#), [Warn\(\)](#), [Info\(\)](#) and [Write\(\)](#) can all be used with the same format string syntax.

The format string must contain a valid place holder for every given argument. The placeholder format is

[flags] [width] [.precision] type

Where type is one of the following specifiers:

d or i For an integer.

- e For a double. The printed format is $[-]d.ddd e [sign]ddd$ where d is a single decimal digit, ddd is one or more decimal digits, ddd is exactly three decimal digits, and $[sign]$ is + or -.
- E Identical to type e except that the exponent is uppercase.
- f For a double. The printed format is $[-]ddd.ddd$, where ddd is one or more decimal digits. The number of digits before the decimal point depends on the magnitude of the number, and the number of digits after the decimal point depends on the requested precision.
- F For a double. Prints a floating point value as a string of fixed width (`width` is required). The place of the point is determined automatically so that always a maximum precision is achieved.
- g For a double. The printed type is e or f, whichever is more compact for the given value and precision. The type e is used only when the exponent of the value is less than -4 or greater than or equal to the precision argument. Trailing zeros are truncated, and the decimal point appears only if one or more digits follow it.
- G Identical to type g, except that the exponent E instead of e, is used (where appropriate).
- s For a string.
- o For an object. Prints the class icon and loc_name of a given object. The loc_name can be clicked to show the element dialog. The width of the icon and name can be limited/extended to a fix number of characters (the icon needs 3 characters). If the specified width is smaller than 3, the width is set to 3.
- b For an object or a string. Prints the class icon of the given object or class name.

The optional `flags` can be of the following specifiers:

- Left align the result within the given field width.
- + Prefix the output value with a sign (+ or -).
- 0 Fill the left of the number with zero's up to the given field width.

The optional `width` specifies the minimum number of characters to be printed.

The optional `.precision` specifies the number of decimals for all floating-point numbers.

EXAMPLE

```
double d;
int i;
string s;

d = 123456789.987654321;
i = 2468;
s = 'hello dpl';

printf('%f|%15.3f|%E|%.2e|%+f|', d, d, d, d, d);
printf('%d|%6d|%-6d|', i, i, i);
printf('%s|%-20s|%20s|', s, s, s);

! string concat is possible:
s = 'this';
s = sprintf('%s %s', s, 'DPL script');

! print and assign in one action:
s = printf('%s %s \smarks{%s} ', s, 'is called', this:loc_name);
printf('%s (again)', s);
```

In addition to placeholders, the printed string may also contain “escape”-sequences for line feeds, tabs, form feeds and colour. The following escape-sequences can be used:

\n	Inserts a line feed.
\t	Inserts a horizontal tab.
\f	Inserts a form feed, for printing purposes.
\\\	Writes a backslash, even when the next character is a n,t,f or c.
%%	Writes a percent sign.
\cx	Inserts a colour change, where “x” is a colour, according to the following table, i.e. \cf changes the colour to brown.

a	black	i	gray
b	black	j	light gray
c	red	k	bordeaux
d	green	l	dark red
e	blue	m	dark green
f	brown	n	light green
g	cyan	o	marine
h	magenta	p	dark blue

EXAMPLE

```
printf('The \cfbrown \cafox jumped \nover\tthe\nlazy\tcat.');
printf('result written to c:\\documents\\pf\\res.txt');
printf('%% = %%6.2f%% %%', 123.34);
```

sprintf*

Returns a formatted string.

The DPL script stops with an appropriate error when the passed arguments don't match the format string.

```
string sprintf(string format,
              [int|double|string|object argument0,]
              [...])
```

ARGUMENTS

format C++-like printf() format string. See the [Format String Syntax](#) for more information.

argument... (optional)

Arguments matching the format string.

RETURNS

The formated string.

DEPRECATED NAMES

ToStr

SEE ALSO

[mbsprintf\(\)](#), [sscanf\(\)](#), [sscanfsep\(\)](#)

EXAMPLE

The following example writes a report to a file not using [fprintf\(\)](#). It redirects the text from the Output Window to a file. The filename is formatted from a path and the name of the current study case.

```

string lines; ! the lines of a report
object studycase;
! Redirect is an ComOp object inside this script
! StopRedirect is an ComCl object inside this script

studycase = GetActiveStudyCase();
Redirect:f = sprintf('%s%s.out', 'c:\\MyDocuments\\results1215\\',
                     studycase:loc_name);
Redirect.Execute();
Form.WriteOut(lines);    ! write the report
StopRedirect.Execute(); ! stop redirection

```

sscanf*

Parses a string of values which are separated by space or tabulator according the given format. Each parsed value is stored into the passed arguments. [sscanfsep\(\)](#) supports other separators.

```

int sscanf(string source,
           string format,
           [int|double|string argument0,]
           [...])

```

ARGUMENTS

source A formatted source string.

format C++-like printf() format string. See the [Format String Syntax](#) for more information. Only int, double and string types are supported.

argument... (*optional*)
Arguments matching the format string.

RETURNS

≥ 0 Number of values successfully parsed and stored.
–1 On error.

SEE ALSO

[sscanfsep\(\)](#), [mbsscanf\(\)](#), [sprintf\(\)](#)

EXAMPLE

The following example assignes the first two fields of string sStr to the string sRes and the double rVal:

```

int result;
double freq;
string unit;

result = sscanf('249.6 Hz', '%f %s', freq, unit);
printf('%f %s (result = %d)', freq, unit, result);

```

sscanfsep*

Parses a string of values which are separated by a separation character according the given format. Each parsed value is stored into the passed arguments. Empty tokens are skipped by default.

```
int sscanfsep(string input,
              string format,
              int|double|string argument0,
              [... ,]
              string separator
              [int skipEmptyToken = 1])
)
```

ARGUMENTS

input Input string containing separated values.

format C++-like printf() format string. See the [Format String Syntax](#) for more information.
Only int, double and string types are supported.

argument...

Arguments matching the format string.

separator Character that separates value in given input string.

skipEmptyToken (optional) **0** Empty tokens are not skipped.

1 Empty tokens are skipped (default).

RETURNS

Number (≥ 0) of values successfully parsed and stored.

SEE ALSO

[sscanf\(\)](#), [sprintf\(\)](#)

EXAMPLE

```
int result;
int i;
string s;

result = sscanfsep('Hello DPL;123', '%s%d', s, i, ';');

printf('result: %d', result);
printf('s: %s', s);
printf('i: %d', i);

!Output:
!result: 2
!s: Hello DPL
!i: 123
```

strchg*

Searches in the string 'str' for the substring 'find' and replaces it with the string 'new'.

```
int strchg(string& str,
           string find,
           string new)
```

ARGUMENTS

str (*in, out*)
 String to be scanned and modified.
find Substring to be found.
new String to be inserted instead of 'find'.

RETURNS

The first index (*geq0*) in 'str' where 'find' was found; -1 if substring was not found.

SEE ALSO

[mbschg\(\)](#)

EXAMPLE

```
int ret;
string str, find, new;

str = 'This is just a test';
find = 'just a';
new = 'a very important';
ret = strchng(str,find,new);
if (ret == -1) { printf('String could not be found!'); }
else {
    printf('%s',str);
}
```

strcmp*

Compares two strings case sensitive. Therefore:

```
int result;
result = strcmp('a' , 'A' ); ! result = 1
result = strcmp('A' , 'a' ); ! result = -1
result = strcmp('a' , 'a' ); ! result = 0
```

```
int strcmp(string str1,
          string str2,
          [int maxCount])
```

ARGUMENTS

str1 The first string.
str2 The second string.
maxCount (optional)
 Maximal number of characters to compare.

SEE ALSO

[mbscmp\(\)](#)

RETURNS

- < 0 str1 is lexicographically less than str2.
- = 0 Strings are equal.
- > 0 str1 is lexicographically greater than str2.

strcpy*

Returns a copy of a substring.

```
string strcpy(string str,
            int startIndex,
            [int maxCount])
```

ARGUMENTS

- str* The string.
- startIndex* The start index (≥ 0).
- maxCount (optional)* Maximal number of characters to copy (> 0).

RETURNS

The copied substring.

SEE ALSO

[mbscpy\(\)](#)

EXAMPLE

```
string str1, str2;
str1 = 'The brown fox';
str2 = strcpy(str1, 4, 5); ! str2 now equals 'brown'
```

strlen*

Returns the length of a string.

```
int strlen(string str)
```

ARGUMENTS

- str* The string.

RETURNS

Returns the number of single-byte characters of a string.

SEE ALSO

[mbslen\(\)](#)

strstr*

Searches for a substring in a string and returns its position.

```
int strstr(string str,
           string substr)
```

ARGUMENTS

str The string.

substr The substring.

RETURNS

The index of the first character of the substring searched for or -1 when it was not found.

SEE ALSO

[mbssrt\(\)](#)

EXAMPLE

The following example searches for string 'brown' in string str1

```
string str1, str2;
int i;
str1 = 'The brown fox';
i = strstr(str1, 'brown');
! i now equals 4
```

strtok*

Splits the string 'source' into tokens separated by the characters defined in 'delimiters' and returns one of these. Empty tokens are skipped by default.

```
string strtok(string source,
             string delimiters,
             int& index,
             [int tokenNumber = 1],
             [int skipEmptyToken = 1])
```

ARGUMENTS

source Source string to be split.

delimiters String of delimiter characters used for the split.

index (out)

Outputs the index of the returned token in the source string (≥ 0).

number (optional)

Number (≥ 1) of the token to be read (default = 1).

skipEmptyToken (optional) **0** Empty tokens are not skipped.

1 Empty tokens are skipped (default).

RETURNS

Token between separator (tokenNumber-1) and (tokenNumber) as a string. If nothing is read, the token is empty and index is set to -1.

SEE ALSO

[mbstok\(\)](#)

EXAMPLE

The following example searches for different tokens in str

```
string res, str, del;
int pos;
str = 'Das, ist nur, ein Test mit Nr. (555); weiter nichts';
del = ',;();';
res = strtok(str,del,pos);
printf('Token: %s    pos = %d',res,pos);
res = strtok(str,del,pos,2);
printf('Token: %s    pos = %d',res,pos);
res = strtok(str,del,pos,4);
printf('Token: %s    pos = %d',res,pos);
```

tochar*

Converts the given codepoint (number) to the corresponding character.

`string tochar(int codePoint)`

ARGUMENTS

codePoint

Number between 0 and 255 which represents a single-byte character in the current Windows ANSI code page.

RETURNS

A string of length 1 containing the character representation of the given codepoint in the current Windows ANSI code page or an empty string if no single-byte character representation exists.

SEE ALSO

[tocharcodepoint\(\)](#), [tombchar\(\)](#)**tocharcodepoint***

Converts a character to its code point.

`int tocharcodepoint(string str)`

ARGUMENTS

str A string of length 1 containing the single-byte character.

RETURNS

Number between 0 and 255 which represents the given character in the current Windows ANSI code page or 0 if an error occurred.

SEE ALSO

[tocharcodepoint\(\)](#), [tombchar\(\)](#)

tolower*

Creates a copy of a string, where all characters are converted to lowercase.

```
tolower(string str)
```

RETURNS

Copy of original string, but with all characters in lowercase.

SEE ALSO

[toupper\(\)](#), [tombslower\(\)](#)

EXAMPLE

```
string src, scp;
src = 'This is just a test';
scp = tolower(src);
printf('%s', scp);
```

toupper*

Creates a copy of a string, where all characters are converted to uppercase.

```
toupper(string str)
```

RETURNS

Copy of original string, but with all characters in uppercase.

SEE ALSO

[tolower\(\)](#), [tombsupper\(\)](#)

EXAMPLE

```
string src, scp;
src = 'This is just a test';
scp = toupper(src);
printf('%s', scp);
```

3 Object Methods

3.1 General Methods

Overview

AddCopy
ContainsNonAsciiCharacters
CopyData
CreateObject
Energize
GetChildren*
GetClassName*
GetCombinedProjectSource
GetConnectedElements*
GetConnectionCount*
GetContents*
GetControlledNode*
GetCubicle*
GetFullName*
GetImpedance*
GetInom*
GetNet
GetNode*
GetOperator*
GetOwner*
GetParent*
GetReferences*
GetRegion*
GetSize*
GetSupplyingSubstations*
GetSupplyingTransformers*
GetSupplyingTrafstations*
GetSystemGrounding*
GetUnom*
GetVal*
GetVarType*
GetZeroImpedance*
HasResults*
IsCalcRelevant*
IsClass*
IsDeleted*
IsEarthed*
IsEnergized*
IsHidden*

IsInFeeder*
 IsNetworkDataFolder*
 IsNode*
 IsObjectActive*
 IsObjectModifiedByVariation*
 Isolate
 IsOutOfService*
 IsReducible*
 IsShortCircuited*
 Inm*
 MarkInGraphics
 Move
 PasteCopy
 PurgeUnusedObjects
 ReplaceNonAsciiCharacters
 ReportNonAsciiCharacters
 ReportUnusedObjects
 SearchObject*
 SetSize
 SetVal
 ShowEditDialog
 ShowFullName*
 ShowModalSelectTree
 snm*
 SwitchOff
 SwitchOn
 unm*
 VarExists*

AddCopy

Adds a copy of a single object or a set of objects to this object (= target object).

When copying a single object it is possible to give the new name. The new name will be concatenated by the given name parts. This is not possible for a project (IntPrj).

The target object must be able to receive a copy of the objects.

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.

When source object(s) and target object are inside different projects the method [object.PasteCopy\(\)](#) has to be used instead, since it adapts all references automatically.

```
object object.AddCopy(object objectToCopy,
                      [int|string partOfName0,]
                      [...])
object object.AddCopy(set objectsToCopy)
```

ARGUMENTS

objectToCopy

Object to copy.

objectNameParts (optional)

Parts of the name of the new copy which will be concatenated to the object name.

objectsToCopy

Set of objects to copy.

RETURNS

Returns the copy that has been created on success or NULL, when copying a single object.

SEE ALSO

[object.PasteCopy\(\)](#), [object.Move\(\)](#), [Delete\(\)](#)

EXAMPLE

The following example copies a given fuse to all calculation relevant cubicles:

```
set cubics;
object fuse, cubic, copy;
int n;

!get template fuse object
fuse = this.SearchObject('*._RelFuse');
if (.not. fuse) { !just create one, if not given
  fuse = this.CreateObject('RelFuse', 'MyFuse');
}

n = 0;
cubics = GetCalcRelevantObjects('StaCubic');
for (cubic = cubics.First(); cubic; cubic = cubics.Next()) {
  n += 1;
  copy = cubic.AddCopy(fuse, 'Fuse Nr', n);
  printf('Created new fuse %o', copy);
}
```

ContainsNonAsciiCharacters

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

```
int object.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.

```
void object.CopyData(object source)
```

ARGUMENTS

source Object from which parameters are to be copied

RETURNS

0 ok

1 error

EXAMPLE

```
object source,
      target;
int ret;
```

```

ret = target.CopyData(source);
if (ret=1) { ! error
    exit();
}

```

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.

```

object object.CreateObject (string className,
                           [int|string 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.

NULL When no object was created.

EXAMPLE

The following example creates a fuse in a set of cubicles. The new fuses will be named “Fuse0”, “Fuse1”, etc.

```

object target;
set cubs;
int n;
cubs = SEL.GetAll('StaCubic');
target = cubs.First();
n = 0;
while (target) {
    target.CreateObject('RelFuse', 'Fuse', n);
    target = cubs.Next();
    n+=1;
}

```

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 menu.

```

int object.Energize([set& changedSwitches,]
                    [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

[object.SwitchOn\(\)](#), [object.SwitchOff\(\)](#), [object.Isolate\(\)](#)

GetChildren*

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

```
set object.GetChildren(int hiddenMode,
                      [string 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

[object.GetContents\(\)](#)

EXAMPLE

The following example lists all contained terminals for each substation:

```
object obj, substat;
set objs, substats;
!lists all contained terminals for each substation
substats = GetCalcRelevantObjects('* ElmSubstat');
for (substat = substats.First(); substat; substat = substats.Next()) {
    objs = substat.GetChildren(0, '*.');
    printf('Terminals of substation %o', substat);
    for (obj = objs.First(); obj; obj = objs.Next()) {
        printf('    %o', obj);
    }
}
```

GetClassName*

Returns the class name of the object.

```
string object.GetClassName()
```

RETURNS

The class name of the object.

DEPRECATED NAMES

GetClass

GetCombinedProjectSource

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

```
object object.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).

```
set object.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.

DEPRECATED NAMES

GetConnectedElms

GetConnectionCount*

Returns the number of electrical connections.

```
int object.GetConnectionCount()
```

RETURNS

The number of electrical connections.

EXAMPLE

```
int i, count;
object transformer, cubicle, bus;
set transformers;

! list all nodes to which a 3-winding transformer is connected
transformers = GetCalcRelevantObjects('* ElmTr3');
for (transformer=transformers.First(); transformer; transformer=transformers.Next()) {
    count = transformer.GetConnectionCount();
    for (i=0; i<count; i=i+1) {
        cubicle=transformer.GetCubicle(i);
        if (cubicle) {
            bus = cubicle:cBusBar;
            if (bus) {
                bus.ShowFullName();
            }
        }
    }
}
```

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.

```
set object.GetContents([string 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.

EXAMPLE

The following example collects all lines that are stored in network objects.

```
set grids, lines;
object line, grid;
grids = GetCalcRelevantObjects('*.ElmNet');
! get all grids
grid = grids.First();
while (grid) {
    printf('Lines in Grid %o',grid);
    ! get all objects of class ElmLne
    ! in all grid and subfolders of grid
    lines = grid.GetContents('*ElmLne',1);
    line = lines.First();
    while (line) {
        line.ShowFullName();
        line = lines.Next();
    }
    grid = grids.Next();
}
```

GetControlledNode*

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

```
object object.GetControlledNode(int bus,
                                double& targetVoltage,
                                [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, NULL if no controlled terminal exists (or not voltage controlled if check=1)

EXAMPLE

The following example writes the controlled nodes for all calculation relevant objects.

```
set objs;
object obj, node;
double vttarget;

objs = GetCalcRelevantObjects();

! list all regulating units

printf('V Regulating Unit:      Controlled Node:      Target Voltage:');
for (obj = objs.First(); obj; obj = objs.Next()) {
    node = obj.GetControlledNode(-1, vttarget);
    if (node) {
        printf('%20s  %20s  %f', obj, node, vttarget);
    }
}
```

GetCubicle*

Returns the cubicle of an object at the connection with index n, or NULL if there is no cubicle inside the object.

```
object object.GetCubicle(int side)
```

ARGUMENTS

side The connection number.

RETURNS

The cubicle object or NULL.

GetFullName*

Returns the full name of the object as a string.

```
string object.GetFullName([int type])
```

ARGUMENTS

type(optional)

Is used to determine the format of the returned full name:

not given

No special formatting.

= 0

The full name (complete database path including the name and class name) of the object. It becomes a clickable link if printed to the output window.

> 0 (but less or equal to 190)

Formatted exactly to this length and also clickable if printed to the output window.

RETURNS

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

EXAMPLE

```
str = obj.GetFullName();
printf('%s', str);
! Output:
! \Support\IntUser\Example Hierarchy 6.IntPrj\Network Model.IntPrjfilder
! \Network Data.IntPrjfilder\Small Network.ElmNet \400 kV Drakelow\SGT3A.ElmTr3

str = obj.GetFullName(0);
printf('%s', str);
! Output:
! 'Network Model\Network Data\Small Network\400 kV Drakelow\SGT3A.ElmTr3'

str = obj.GetFullName(30);
printf('%s', str);
! Output:
! '400 kV Drakelow\SGT3A.ElmTr3'
```

GetImpedance*

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

```
int object.GetImpedance(double& real,
                        double& imag,
                        double 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.

```
double object.GetInom([int busIndex = 0],  
                      [int inclChar = 0])
```

ARGUMENTS

busIndex (optional)

Bus index, default value is 0.

inclChar (optional)

option to consider thermal rating objects and values modified by characteristics on the (de-)rating factor.

- 0 Not considering thermal rating objects or values modified by characteristics on the (de-)rating factor (default).
- 1 Considering thermal rating objects and values modified by characteristics on the (de-)rating factor.
- 2 Considering thermal rating objects but not values modified by characteristics on the (de-)rating factor.

RETURNS

The nominal current at bus index.

DEPRECATED NAMES

Inom

SEE ALSO

[object.GetUnom\(\)](#)

EXAMPLE

The following example shows the nominal voltages and currents on both sides of all transformers.

```
set lines;  
object obj;  
double curl1, cur2;  
double volt1, volt2;  
  
lines = GetCalcRelevantObjects('* ElmTr2');  
for(obj=lines.First(); obj; obj=lines.Next()) {  
    volt1 = obj.GetUnom(0);  
    volt2 = obj.GetUnom(1);  
    curl1 = obj.GetInom(0);  
    cur2 = obj.GetInom(1);  
    printf('%o: Inom1 = %f, Inom2 = %f', obj, volt1, volt2);  
    printf('%o: Unom1 = %f, Unom2 = %f', obj, curl1, cur2);  
}
```

GetNet

Returns the grid in which the object is located.

```
object object.GetNet()
```

RETURNS

- object** The grid (*ElmNet*) where the object is stored in.
NULL If the current object is not stored in any grid.

GetNode*

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

```
object object.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.
NULL If no node at bus index is found.

GetOperator*

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

```
object object.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 returned.
- Else the operator inherited from its parent is used (recursively applied).
- NULL if none of its parents have an operator set.

GetOwner*

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

```
object object.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 returned.
- Else the owner inherited from its parent is used (recursively applied).
- NULL if none of its parents have an owner set.

GetParent*

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

```
object object.GetParent()
```

RETURNS

object The parent folder object.

NULL On the root database folder e.g. parent of a user.

SEE ALSO

[object.GetContents\(\)](#)

EXAMPLE

The following example returns the folder in which a line is stored.

```
set objects;
object line, folder;

objects = GetCalcRelevantObjects();
line = objects.Firstmatch('ElmLne');

folder = line.GetParent();
if (folder) {
    folder.ShowFullName();
}
```

GetReferences*

Returns a set containing all objects with references to the object the method was called on. By default, references from IntSubset objects or hidden objects are ignored.

```
set object.GetReferences([string filter = '*',]
                        [int includeSubsets = 0,]
                        [int includeHiddenObjects = 0])
```

ARGUMENTS

filter (optional)

Object filter to get only objects whose name matches this filter string, e.g. ".ElmLne".
(default: '*')

includeSubsets (optional)

Forces references from IntSubset objects to be evaluated. These are normally not included for performance reasons. (default: 0)

includeHiddenObjects (optional)

Include also hidden objects. By default they are not included. In contrast hidden objects are always included in the 'Reference List' output of the data browser.
(default: 0)

RETURNS

Set of objects with references to the object the method was called on.

EXAMPLE

The following example returns all relevant objects of class 'TypBar':

```
set objs, refs;
object obj1, obj2;

objs = GetCalcRelevantObjects('*_TYPBAR*'); !get all relevant objects of class TypBar

for (obj1 = objs.First(); obj1; obj1 = objs.Next()) {
    printf('Object referencing to %o', obj1);

    refs = obj1.GetReferences();
    for (obj2 = refs.First(); obj2; obj2 = refs.Next()) {
        obj2.ShowFullName();
    }
}
```

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 \Leftrightarrow *elm1*.GetRegion() == *elm2*.GetRegion().

A region is simply identified by a number that can be access via this function.

```
int object.GetRegion()
```

EXAMPLE

The following example shows the region index for all calculation relevant network elements:

```
set elements;
object obj;
int region, t;
string clasz;

elements = GetCalcRelevantObjects();
for (obj = elements.First(); obj; obj = elements.Next()) {

    clasz = obj.GetClass();
    t = strcmp('Elm', clasz, 3);
    if (t <> 0) {
        continue; !only for network elements
    }

    region = obj.GetRegion();
    if (region > -1) {
        printf('Element %o belongs to region %d', obj, region);
    }
    else {
        printf('Region for element %o is unknown', obj);
    }
}
```

RETURNS

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

GetSize*

Returns the size of the variable “VarName” when this variable is a vector or a matrix.

```
int object.GetSize(string VarName,
                  int& rows,
                  [int& cols])
```

ARGUMENTS

VarName The name of the variable.

rows (out)
The number of rows for a vector or matrix.

cols (optional, out)
The number of columns for a matrix.

RETURNS

- 0** When “VarName” is a valid variable name.
- 1** Otherwise.

SEE ALSO

[object.GetVal\(\)](#)

EXAMPLE

The following example prints the matrix resistances from a tower model with 2 circuits.

```
int err;
double x;
int r, rows, c, cols;
string str;
err = Tower.GetSize('R_c', rows, cols);
if (.not.err) {
    r=0;
    while (r<rows) {
        str = '';
        c = 0;
        while (c<cols) {
            err = Tower.GetVal(x, 'R_c', r,c);
            if (.not.err) str = sprintf('%str %f', str, x); {
                c+=1;
            }
            printf(str);
            r+=1;
        }
    }
}
```

Example Output :

0.067073	0.016869	0.016594	0.016851	0.016576	0.016372	0.016869	0.066832
0.016701	0.016576	0.016445	0.016408	0.016594	0.016701	0.066738	0.016372
0.016408	0.016516	0.016851	0.016576	0.016372	0.067073	0.016869	0.016594
0.016576	0.016445	0.016408	0.016869	0.066832	0.016701	0.016372	0.016408
0.016516	0.016594	0.016701	0.066738				

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.

```
set object.GetSupplyingSubstations()
```

RETURNS

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

SEE ALSO

[ElmTr2.GetSuppliedElements\(\)](#), [ElmTr3.GetSuppliedElements\(\)](#), [ElmSubstat.GetSuppliedElements\(\)](#),
[ElmTrfstat.GetSuppliedElements\(\)](#), [object.GetSupplyingTransformers\(\)](#), [object.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.

```
set object.GetSupplyingTransformers()
```

RETURNS

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

SEE ALSO

[ElmTr2.GetSuppliedElements\(\)](#), [ElmTr3.GetSuppliedElements\(\)](#), [ElmSubstat.GetSuppliedElements\(\)](#),
[ElmTrfstat.GetSuppliedElements\(\)](#), [object.GetSupplyingSubstations\(\)](#), [object.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.

```
set object.GetSupplyingTrfstations()
```

RETURNS

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

SEE ALSO

[ElmTr2.GetSuppliedElements\(\)](#), [ElmTr3.GetSuppliedElements\(\)](#), [ElmSubstat.GetSuppliedElements\(\)](#),
[ElmTrfstat.GetSuppliedElements\(\)](#), [object.GetSupplyingTransformers\(\)](#), [object.GetSupplyingSubstations\(\)](#)

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 object.GetSystemGrounding()
```

RETURNS

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

DEPRECATED NAMES

GetSystemGround

GetUnom*

Returns the nominal voltage of the object.

```
double object.GetUnom([int busIndex = 0])
```

ARGUMENTS

busIndex (optional)

Bus index, default value is 0.

RETURNS

The nominal voltage at bus index.

DEPRECATED NAMES

Unom

SEE ALSO

[object.GetInom\(\)](#)

EXAMPLE

The following example collects all high voltage lines. The value VoltageLevel is an input parameter. The script also needs a general selection defined.

```
set lines, highvoltages;
object obj;
double voltage;

highvoltages.Clear();

lines = SEL.AllLines();
obj = lines.First();
while (obj) {
    voltage = obj.GetUnom();
    if (voltage>VoltageLevel) {
        highvoltages.Add(obj);
    }
    obj = lines.Next();
}

for (obj=highvoltages.First(); obj; obj=highvoltages.Next()) {
    obj.ShowFullName();
}
```

GetVal*

Returns the value of the given variable in the currently set unit (unit could differ from `object.SetVal()`). The row or column can be given for vector or matrix variables.

```
int object.GetVal(int& | double& | string& | object& | set& value,
                  string variable,
                  [int row,]
                  [int col,]
                  [string key])
```

ARGUMENTS

value (out)

Outputs the value of the variable. The value is always in the currently set unit of the variable (as seen in the edit dialog).

variable Name of the variable to get the value for.*row (optional)*

Row number for a vector or matrix variable (≥ 0). For negative numbers 0 is used.

col (optional)

Column number for a matrix variable (≥ 0). For negative numbers 0 is used.

key (optional)

The key parameter is used to get the real, imaginary part, angle, magnitude of a complex parameter (and complex vector, matrix), or for a double vector parameter to get statistic values like min, max, average, sum, ...

'r','i','mag','phi','phirad' real part, imaginary part, magnitude, angle in degree, angle in radians

'min','max','sum','avg','std','varia' minimum, maximum, sum, avarage, standard deviation, variance...

RETURNS

0 Value successfully obtained.

1 On error e.g. variable does not exist or row/column number is out of range.

SEE ALSO

[object.SetVal\(\)](#)

GetVarType*

Outputs the data type of an attribute of the object.

```
int object.GetVarType(string name,
                      string& type)
```

ARGUMENTS

name Name of the attribute to get the type for.

type (out) Outputs the data type of the attribute as string, if it exists.

RETURNS

0 Variable exists.

1 On error.

EXAMPLE

```

set terminals;
object terminal;
string attribute, type;

!get one of the calc relevant terminals
terminals = GetCalcRelevantObjects('ElmTerm');
terminal = terminals.First();

attribute = 'loc_name';
terminal.GetVarType(attribute, type);
printf('Data type of attribute "%s" is "%s"', attribute, type);

attribute = 'uknom';
terminal.GetVarType(attribute, type);
printf('Data type of attribute "%s" is "%s"', attribute, type);

```

GetZeroImpedance*

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

```

int object.GetZeroImpedance(double& real,
                            double& imag,
                            double 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()

HasResults*

Checks if the object has calculated result parameters.

```

int object.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

EXAMPLE

The following example checks if lines have results.

```
set lines;
object line;
int iret;

lines = GetCalcRelevantObjects('* ElmLne');
for(line = lines.First(); line; line = lines.Next()) {
    iret = line.HasResults();
    if (iret) {
        printf('Line %o has results', line);
    }
    else {
        printf('Line %o DOES NOT have results', line);
    }
}
```

IsCalcRelevant*

Returns whether the object is relevant for calculation.

In contrast to [GetCalcRelevantObjects\(\)](#) it also returns 1 for objects in the active study case e.g. simulation events (Evt*).

```
int object.IsCalcRelevant()
```

RETURNS

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

DEPRECATED NAMES

IsRelevant

SEE ALSO

[GetCalcRelevantObjects\(\)](#)

EXAMPLE

The following example checks if a line is used in the calculation.

```
int isCalcRelevant;
isCalcRelevant = MyLine.IsCalcRelevant();
```

```
if (isCalcRelevant) {
    MyLine.ShowFullName();
}
```

IsClass*

Returns whether the object is of a certain class.

```
int object.IsClass(string className)
```

ARGUMENTS

className

The name of the class.

RETURNS

- 1** Object is of the given class.
- 0** Object is not of the given class.

SEE ALSO

[object.GetClassName\(\)](#)

EXAMPLE

The following example write all overloaded lines and transformers to the output window, where a different maximum loading is used for lines or transformers.

```
set objects;
object obj;
int result;
objects = GetCalcRelevantObjects();
obj = objects.First();
while (obj) {
    result = obj.IsClass('ElmLne');
    if (result) {
        if (obj:c:loading > 0.85) {
            obj.ShowFullName();
        }
    }
    else {
        result = obj.IsClass('ElmTr2');
        if (result) {
            if (obj:c:loading > 0.95) {
                obj.ShowFullName();
            }
        }
    }
    obj = objects.Next();
}
```

IsDeleted*

Returns 1 if the object is deleted.

```
int object.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 object.IsEarthed()
```

RETURNS

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

EXAMPLE

The following example shows the earthed elements:

```
set elements;
object obj;
int status, t;
string clasz;

elements = GetCalcRelevantObjects();
for (obj = elements.First(); obj; obj = elements.Next()) {
    clasz = obj.GetClass();
    t = strcmp('Elm', clasz, 3);
    if (t <> 0) {
        continue; !only for network elements
    }
    status = obj.IsEarthed();
    if (status = 0) {
        printf('Component %o is not earthed.', obj);
    }
    else if (status > 0) {
        printf('Component %o is earthed.', obj);
    }
}
```

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 object.IsEnergized()
```

RETURNS

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

EXAMPLE

The following example shows the energizing status of all elements:

```
set elements;
object obj;
int status, t;
string clasz;

elements = GetCalcRelevantObjects();

for (obj = elements.First(); obj; obj = elements.Next()) {
    clasz = obj.GetClass();
    t = strcmp('Elm', clasz, 3);
    if (t <> 0) {
        continue; !only for network elements
    }

    status = obj.IsEnergized();
    if (status = 0) {
        printf('Component %o is de-energized.', obj);
    }
    else if (status > 0) {
        printf('Component %o is energized.', obj);
    }
    else if (status < 0) {
        printf('Energizing status for %o is unknown.', obj);
    }
}
```

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 object.IsHidden()
```

RETURNS

- | | |
|----------|--------------------------------|
| 0 | not hidden, currently 'active' |
| 1 | hidden, currently 'inactive' |

IsInFeeder*

Checks if the object is part of the given feeder. A network element is considered being part of a feeder if a topological path from the feeder definition to the element exists.

This function is based on load flow calculation results. Therefore, it can only be used after such a calculation has been successfully executed and as long as the results are available.

```
int object.IsInFeeder(object 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 part of that feeder.
- 0** Otherwise

SEE ALSO

[ElmFeeder.GetAll\(\)](#)

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 *ElmAra* 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 object.IsNetworkDataFolder()
```

RETURNS

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

SEE ALSO

[GetDataFolder\(\)](#)

IsNode*

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

```
int object.IsNode()
```

RETURNS

- 1** Object is a node.
- 0** Otherwise.

IsObjectActive*

Check if an object is active for specific time.

```
int object.IsObjectActive(int time)
```

ARGUMENTS

time Time in seconds since 01.01.1970 00:00:00.

RETURNS

- 0** Object is not active (hidden or deleted).
- 1** Object is active.

IsObjectModifiedByVariation*

Check if an object is active for specific time.

```
int object.IsObjectModifiedByVariation(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 menu.

```
int object.Isolate([set& changedSwitches,
                    [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

[object.SwitchOn\(\)](#), [object.SwitchOff\(\)](#), [object.Energize\(\)](#)

IsOutOfService*

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

```
int object.IsOutOfService()
```

RETURNS

0 When the object is in service.

1 When the object is out of service.

EXAMPLE

The following example checks if a line is out of service.

```
int i;
i = MyLine.IsOutOfService\(\);
if (i) {
    MyLine.ShowFullName\(\);
}
```

IsReducible*

Checks if object can be reduced during network reduction.

```
int object.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)

EXAMPLE

The following example checks if an object is reducible:

```
set objs;
object obj;
int res;

objs = GetCalcRelevantObjects();
for (obj = objs.First(); obj; obj = objs.Next()) {
    res = obj.IsReducible();
    if (res = 0) {
        printf('Object %o is not reducible.', obj);
        continue;
    }
    if (res = 1) {
        printf('Object %o is reducible.', obj);
        continue;
    }
    if (res = 2) {
        printf('Object %o is currently not reducible.', obj);
        continue;
    }
}
```

IsShortCircuited*

Returns whether an element is short-circuited or not.

```
int object.IsShortCircuited()
```

RETURNS

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

Inm*

Returns the long description of the variable.

```
string object.lnm(string VarName
                  [int& error])
```

ARGUMENTS

VarName The variable name.

error (optional, out)

1 on error, otherwise 0.

RETURNS

The long variable description.

SEE ALSO

[object.snm\(\)](#), [object.unm\(\)](#)

EXAMPLE

The following example prints information about the length of a line.

```
set lines;
object line;
string s1,s2,s3;

lines = GetCalcRelevantObjects('*ElmLne');
for(line = lines.First(); line; line = lines.Next()) {
    s1 = line.lnm('dline');
    s2 = line.snm('dline');
    s3 = line.unm('dline');
    printf('%s (%s) = %5.3f [%s]',s1, s2, line:dline, s3);
}
```

Example output:

Length of Line (Length) = 0.547 [km]

MarkInGraphics

This function is not supported in GUI-less mode.

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.

```
void object.MarkInGraphics([int searchAllDiagramsAndSelect = 0])
```

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. (default)
- 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.

EXAMPLE

The following example will a line in the currently visible diagram or if not found in one of the other diagrams which are currently opened .

```
set lines;
object line;
lines = GetCalcRelevantObjects('*.*ElmLne');
line = lines.First();
if (line) {
    line.MarkInGraphics();
}
```

Move

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

```
int object.Move(object objectToMove)
int object.Move(set objectsToMove)
```

ARGUMENTS

objectToMove
Object to move.

objectToMove
Set of objects to move.

RETURNS

- 0** On success.
- 1** On error.

SEE ALSO

[object.AddCopy\(\)](#), [object.PasteCopy\(\)](#), [Delete\(\)](#)

EXAMPLE

```
object targetobj,startobj;
set allObjs;
! move startobj to targetobj
targetobj.Move(startobj);
! move all objects inside allObjs to targetobj
targetobj.Move(allObjs);
```

PasteCopy

Pastes a copy of a single object or a set of objects to this object (= target object) using the merge tool if source object(s) and target object are inside different projects (equivalent to a manual copy&paste operation).

If the argument ‘resetMissingReferences’ is not given, a dialog on assignment conflicts will pop up, which offers to reset all missing references, to show the merge tool or to cancel the action.

```
int object.PasteCopy(object objectToCopy,
                     [object& newObject],
                     [int resetMissingReferences])
int object.PasteCopy(set objectsToCopy)
```

ARGUMENTS

objectToCopy

Object to be copied.

objectsToCopy

Set of objects to copy.

newObject (optional, out)

The new object copy of *objectToCopy* is assigned on success.

resetMissingReferences (optional)

If set, determines how to handle missing references.

0 No action is taken, the operation is cancelled with an error.

1 Missing references are automatically reset.

RETURNS

0 Object(s) successfully copied.

1 Error.

SEE ALSO

[object.AddCopy\(\)](#), [object.Move\(\)](#), [Delete\(\)](#)

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.

```
void object.PurgeUnusedObjects()
```

SEE ALSO

[object.ReportUnusedObjects\(\)](#)

ReplaceNonAsciiCharacters

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

```
int object.ReplaceNonAsciiCharacters(object map,
                                     string 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.

```
void object.ReportNonAsciiCharacters()
```

ReportUnusedObjects

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

```
void ReportUnusedObjects()
```

SEE ALSO

[object.PurgeUnusedObjects\(\)](#)

SearchObject*

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

```
object object.SearchObject(string name)
```

ARGUMENTS

name string to search

RETURNS

Returns the searched object.

EXAMPLE

The following example searches for a terminal in the Network Model folder.

```
object obj, folder;

folder = GetProjectFolder('netmod');
if (folder) {
    obj = folder.SearchObject('Network Data.IntPrjfolder\Nine_Bus.ElmNet\Bus1.ElmTerm');
    if (obj) {
        obj.ShowFullName();
    }
}
```

SEE ALSO

[object.GetFullName\(\)](#)

SetSize

Sets the size of the variable 'VarName' for an object if this variable is a vector or matrix.

```
int object.SetSize(string VarName,
                   int rows,
                   [int cols])
```

ARGUMENTS

VarName Object variable

rows Row of the variable's matrix or vector

cols (optional) Column of the variable's matrix or vector

RETURNS

0 'VarName' is a valid variable name

1 Variable not found or variable is not a matrix or vector

SEE ALSO

[object.SetVal\(\)](#)

EXAMPLE

The following example will set the size of the row and column to 5:

```
object typSym;
int res1,res2;

typSym.GetSize('satv',oldsize);
printf('Old size of vector: %d',oldsize);

res1 = typSym.SetSize('satv',5);
res2 = typSym.SetSize('satse',5);

if (res1=1.or.res2=1) {
    printf('Error - parameter setse or setv no vector or matrix');
    exit();
}

typSym.GetSize('satv',size);
printf('New size of vector: %d',size);
```

SetVal

Sets the value of the given variable in the default unit (unit could differ from [object.GetVal\(\)](#)). The row or column can be given for vector or matrix variables.

```
int object.SetVal(int|double|string|object|set value,
                  string variable,
                  [int row],
                  [int col])
```

ARGUMENTS

value (out)

Value to set to. Have to be given in the default unit and could differ from the currently set unit.

variable Name of the variable to set the value for.*row (optional)*

Row number for a vector or matrix variable (≥ 0). For negative numbers 0 is used.

col (optional)

Column number for a matrix variable (≥ 0). For negative numbers 0 is used.

RETURNS

0 Value successfully set.

1 On error e.g. variable does not exist or row/column number is out of range.

SEE ALSO

[object.GetVal\(\)](#)

EXAMPLE

The following example sets the size of the row and column to 5:

```
object typSym;
int row, size;
double val;

typSym.GetSize('satv',size);

val = 0;
row = 0;
while (row<size) {
    typSym.SetVal(val,'satv',row);
    typSym.SetVal(0,'satse',row);

    val += 0.1;
    row += 1;
}
```

ShowEditDialog

This function is not supported in GUI-less mode.

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 object.ShowEditDialog()
```

RETURNS

1 Edit dialogue was cancelled by the user.

0 Otherwise.

DEPRECATED NAMES

Edit

EXAMPLE

The following example opens a line dialogue, prior to calculating a load flow.

```
MyLine.ShowEditDialog();
Ldf.Execute();
```

ShowFullName*

Prints the full name (complete database path including the name and class name) of the object as clickable link to the output window. This is useful to inspect or edit the printed object after the script has finished.

```
void object.ShowFullName()
```

SEE ALSO

[object.GetFullName\(\)](#)

EXAMPLE

The following example writes all overloaded lines from the selection to the output window.

```
set lines;
object line;
lines = SEL.AllLines();
line = lines.First();
while (line) {
    if (line:c:loading > 100.0) {
        line.ShowFullName();
    }
    line = lines.Next();
}
```

ShowModalSelectTree

This function is not supported in GUI-less mode.

Shows a modal window with the database object tree of the object on which the function is called on.

```
object object.ShowModalSelectTree([string title,
                                    [string 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

object Selected object.

NULL No object selected e.g. 'Cancel' clicked.

snm*

Returns the short variable name. By default, the short name equals the long variable name. In some cases, the variable also has a short name which is used to save space in reports or dialogues.

```
string object.snm(string VarName
                  [int& error])
```

ARGUMENTS

VarName The variable name

error (optional, out)

1 on error, otherwise 0.

RETURNS

The short name.

SEE ALSO

[object.lnm\(\)](#), [object.unm\(\)](#)

SwitchOff

Performs a "switch off" action on the network element. This action is identical to that in the context menu.

```
int object.SwitchOff([set& changedSwitches,
                      [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

[object.SwitchOn\(\)](#), [object.Isolate\(\)](#), [object.Energize\(\)](#)

SwitchOn

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

```
int object.SwitchOn([set& changedSwitches,
                     [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

[object.SwitchOff\(\)](#), [object.Isolate\(\)](#), [object.Energize\(\)](#)

unm*

Returns the unit of the variable.

```
string object.unm(string VarName,
                  [int& error])
```

ARGUMENTS

VarName The variable name
error (optional, out)
 1 on error, otherwise 0.

RETURNS

The unit name.

SEE ALSO

[object.lnm\(\)](#), [object.snm\(\)](#)

VarExists*

Checks whether the variable exists and whether it is currently valid on this object.

```
int object.VarExists(string name)
```

ARGUMENTS

name Name of the variable.

RETURNS

- 1** Variable exists and is currently valid on this object.
- 0** Variable does not exist e.g. variable never exists or variable currently not exists since the load flow is not calculated.

EXAMPLE

The following example calculates the total length of cables and lines.

```
int exists, unitFound;
double totalLength;
string unit;
object obj;
set objects;

objects = GetCalcRelevantObjects();
obj = objects.First();

while (obj) {
    exists = obj.VarExists('dline');
    if (exists) {
        totalLength += obj:dline;

        unitFound = strlen(unit);
        if (unitFound == 0) {
            unit = obj.unm('dline');
        }
    }
    obj = objects.Next();
}

printf('Total length = %f %s', totalLength, unit);
```

3.2 Network Elements

3.2.1 ElmArea

Overview

[CalculateInterchangeTo*](#)
[DefineBoundary](#)
[GetAll*](#)
[GetBranches*](#)
[GetBuses*](#)
[GetObjs*](#)

CalculateInterchangeTo*

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

```
int ElmArea.CalculateInterchangeTo(object 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

EXAMPLE

```
object AreaA, AreaB; ! externally defined (i.e. input)
AreaA = inputA;
AreaB = inputB;
object Ldf = GetFromStudyCase('ComLdf');
Ldf.Execute();
AreaA.CalculateInterchangeTo(AreaB);
printf('Interchange Pinter from %o to %o is %f MW', AreaA, AreaB, AreaA:c:Pinter);
```

DefineBoundary

Defines boundary with this area as interior part. Resulting cubicles of boundary are busbar-oriented towards the area.

```
object ElmArea.DefineBoundary(int shift)
```

ARGUMENTS

shift Elements outside the area that are within a distance of *shift* many elements to a boundary cubicle of the area are added to the interior part of the resulting boundary

RETURNS

The defined boundary is returned in case of success. Otherwise NULL, if an error appeared in the definition of the boundary.

EXAMPLE

```
external object: "thisArea"
    double shift;
    object newBoundary;
    shift = 2;
    newBoundary = thisArea.DefineBoundary(shift);
    if ({newBoundary <> NULL}) {
        printf('Defined boundary %o by shifting %d elements!', newBoundary, shift);
    }
```

GetAll*

Returns all objects which belong to this area.

```
set ElmArea.GetAll()
```

RETURNS

The set of contained objects.

EXAMPLE

```
set all,areas;
object prj,area,obj;
! output elements in the area
prj = GetActiveProject();
if (prj) {
    areas = prj.GetContents('*.*ElmArea',1);
    areas.SortToVar(0,'loc_name');
    for (area=areas.First(); area; area=areas.Next()) {
        printf('Elements in area %s',area:loc_name);
        all = area.GetAll();
        for (obj=all.First(); obj; obj=all.Next()) {
            printf('%s\\%s',obj:r:fold_id:loc_name,obj:loc_name);
        }
    }
}
```

GetBranches*

Returns all branches which belong to this area.

```
set ElmArea.GetBranches()
```

RETURNS

The set of branch objects.

EXAMPLE

```
set all,areas;
object prj,area,obj;
! output elements in the area
prj = GetActiveProject();
if (prj) {
    areas = prj.GetContents('*.*ElmArea',1);
    areas.SortToVar(0,'loc_name');
```

```

for (area=areas.First(); area; area=areas.Next()) {
    printf('Branches in area %s',area:loc_name);
    all = area.GetBranches();
    for (obj=all.First(); obj; obj=all.Next()) {
        printf('%s\\%s',obj:r:fold_id:loc_name,obj:loc_name);
    }
}
}

```

GetBuses*

Returns all buses which belong to this area.

```
set ElmArea.GetBuses()
```

RETURNS

The set of objects.

GetObjs*

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

```
set ElmArea.GetObjs(string classname)
```

ARGUMENTS

classname

Name of the class (i.e. "ElmTr2").

RETURNS

The set of objects.

EXAMPLE

```

set all,areas;
object prj,area,obj;
! output cubicles in the area
prj = GetActiveProject();
if (prj) {
    areas = prj.GetContents('*ElmArea',1);
    areas.SortToVar(0,'loc_name');
    for (area=areas.First(); area; area=areas.Next()) {
        printf('Cubicles in area %s',area:loc_name);
        all = area.GetObjs('StaCubic');
        for (obj=all.First(); obj; obj=all.Next()) {
            printf('%s\\%s',obj:r:fold_id:loc_name,obj:loc_name);
        }
    }
}

```

3.2.2 ElmAsm

Overview

CalcEfficiency
 GetAvailableGenPower*
 GetElecTorque*
 GetGroundingImpedance*
 GetMechTorque*
 GetMotorStartingFlag*
 GetStepupTransformer*
 IsPQ*

CalcEfficiency

Calculate efficiency for connected storage model at given active power value.

```
double ElmAsm.CalcEfficiency(double activePowerMW)
```

ARGUMENTS

activePowerMW

Active power value to calculate efficiency for.

RETURNS

Returns the resulting efficiency in p.u.

EXAMPLE

```
set objs;
object obj;
int err;
double eff, Ptherm, Pelect;

Pelect = 30.0;
objs = GetCalcRelevantObjects('*.ElmAsm');

obj = objs.First();
if (obj) {
    eff = obj.CalcEfficiency(Pelect);
    Ptherm = Pelect/eff;
}
```

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.

```
double ElmAsm.GetAvailableGenPower()
```

RETURNS

Available generation power

EXAMPLE

```
set generators;
object generator;
double totalpower, power;

generators = GetCalcRelevantObjects('*.ElmAsm');

totalpower = 0; ! initialize cumulative generation

! get cumulative generation
for (generator = generators.First(); generator; generator = generators.Next()) {
    power = generator.GetAvailableGenPower();
    totalpower += power;
}
printf('Cummulative generation is %f', totalpower);
```

GetElecTorque*

Calculates the electrical torque for a given speed and voltage.

```
double ElmAsm.GetElecTorque(double speed,
                            double uReal,
                            [double addZReal,]
                            [double 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 ElmAsm.GetGroundingImpedance(int busIdx,
                                  double& resistance,
                                  double& reactance)
```

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.

```
double ElmAsm.GetMechTorque(double speed,
                            double 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.

```
object ElmAsm.GetStepupTransformer(double hvVoltage,
                                    int ignSwtStatus
                                   )
```

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.

3.2.3 ElmAsm sc

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.

```
double ElmAsmsc.GetAvailableGenPower()
```

RETURNS

Available generation power

EXAMPLE

```
set generators;
object generator;
double totalpower, power;

generators = GetCalcRelevantObjects('*.ElmAsmsc');

totalpower = 0; ! initialize cummulative generation

! get cummulative generation
for (generator = generators.First(); generator; generator = generators.Next()) {
    power = generator.GetAvailableGenPower();
    totalpower += power;
}
printf('Cummulative generation is %f', totalpower);
```

GetGroundingImpedance*

Returns the impedance of the internal grounding.

```
int ElmAsmsc.GetGroundingImpedance(int busIdx,
                                     double& resistance,
                                     double& reactance)
```

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.

```
object ElmAsmsc.GetStepupTransformer(double hvVoltage,
                                      int ignSwtStatus
                                      )
```

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).

3.2.4 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.

EXAMPLE

The following example checks whether all calculated backbones are still valid.

```
object oBbone;
object folder;
set elementBbones;
int valid;

folder = GetDataFolder('IntBbone');
elementBbones = folder.GetContents('*.*ElmBbone');

for ( oBbone = elementBbones.First(); oBbone; oBbone = elementBbones.Next() ) {
    valid = oBbone.CheckBbPath(0);
    if (valid==0) {
        printf('Backbone %o is valid.', oBbone);
    }
}
```

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.

EXAMPLE

This example calculates backbones aretrieves all calculated backbones

```
int order;
object backboneCmd;
object backbone;
object folder;
set allBackbones;

backboneCmd = GetFromStudyCase('ComBbone');
if ( backboneCmd<>nullptr ) {
    backboneCmd:e:iFeedSetting = 0; ! for all feeders
    backboneCmd:e:iCalcMeth = 1; ! Criterion is cross section
    backboneCmd.Execute();

    folder = GetDataFolder('IntBbone');
    allBackbones = folder.GetContents('*.*ElmBbone');

    for ( backbone=allBackbones.First(); backbone; backbone=allBackbones.Next() ) {
        order = backbone.GetBbOrder();
        printf('Backbone %o has order %d', backbone, order);
    }
}
```

GetCompleteBbPath*

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

```
void ElmBbone.GetCompleteBbPath(set<AllElmsOnBb> &AllElmsOnBb,
                                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)

EXAMPLE

This example lists all calculated backbones together with all their elements

```
object oBbone;
object obj;
object folder;
set ElementsOnBbone;
set ElementBbones;

folder = GetDataFolder('IntBbone');
ElementBbones = folder.GetContents('*.*.ElementBbone');

for (oBbone=ElementBbones.First(); oBbone; oBbone=ElementBbones.Next()) {
    ElementsOnBbone.Clear();
    oBbone.GetCompleteBbPath(ElementsOnBbone, 0);
    for (obj=ElementsOnBbone.First(); obj; obj=ElementsOnBbone.Next()) {
        obj.ShowFullName();
    }
}
```

GetFOR

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

```
double ElmBbone.GetFOR()
```

RETURNS

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

EXAMPLE

```

object oBbone;
object folder;
double dFOR;
set ElementBbones;

folder = GetDataFolder('IntBbone');
ElementBbones = folder.GetContents('*.*ElmBbone');

for ( oBbone=ElementBbones.First(); oBbone; oBbone=ElementBbones.Next() ) {
    dFOR = oBbone.GetFOR();
    printf('Backbone %o has a FOR of %f per year.', oBbone, dFOR);
}

```

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.

```
double ElmBbone.GetMeanCs()
```

RETURNS

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

EXAMPLE

```

object oBbone;
object folder;
double meanCs;
set ElementBbones;

folder = GetDataFolder('IntBbone');
ElementBbones = folder.GetContents('*.*ElmBbone');

for ( oBbone=ElementBbones.First(); oBbone; oBbone=ElementBbones.Next() ) {
    meanCs = oBbone.GetMeanCs();
    printf('Backbone %o has a mean cross section of %f mm2.', oBbone, meanCs);
}

```

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.

```
double ElmBbone.GetMinCs([set& ElmsMinCs])
```

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].

EXAMPLE

```

object oBbone;
object obj;
object folder;
double minCs;
set ElmsMinCs;
set ElementBbones;

folder = GetDataFolder('IntBbone');
ElementBbones = folder.GetContents('*.*ElmBbone');

for ( oBbone=ElementBbones.First(); oBbone; oBbone=ElementBbones.Next() ) {
    minCs = oBbone.GetMinCs(ElmsMinCs);
    printf('Backbone %o has a min. cs of %f mm2.\n', oBbone, minCs);
    printf('Its following elements feature this cross section\n');
    for (obj=ElmsMinCs.First(); obj; obj=ElmsMinCs.Next()) {
        printf(' element %o\n', obj);
    }
}

```

GetTieOpenPoint*

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

```
object ElmBbone.GetTieOpenPoint()
```

RETURNS

The switching device (ElmCoup or StaSwitch) or NULL if backbone is invalid.

EXAMPLE

```

object oBbone;
object folder;
object oTie;
set ElementBbones;

folder = GetDataFolder('IntBbone');
ElementBbones = folder.GetContents('*.*ElmBbone');

for ( oBbone=ElementBbones.First(); oBbone; oBbone=ElementBbones.Next() ) {
    oTie = oBbone.GetTieOpenPoint();
    if ( oTie<>nullptr ) {
        oTie.ShowFullName();
    }
}

```

GetTotLength*

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

```
double ElmBbone.GetTotLength()
```

RETURNS

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

EXAMPLE

```

object oBbone;
object folder;
double length;
set ElementBbones;

folder = GetDataFolder('IntBbone');
ElementBbones = folder.GetContents('*.*ElmBbone');

for ( oBbone=ElementBbones.First(); oBbone; oBbone=ElementBbones.Next() ) {
    length = oBbone.GetTotLength();
    printf('Backbone %o has a length of %f km.', oBbone, length);
}

```

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.

3.2.5 ElmBmu**Overview**

[Apply](#)
[Update](#)

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'.

```
void ElmBmu.Update()
```

3.2.6 ElmBoundary

Overview

[AddCubicle](#)
[CalcShiftedReversedBoundary](#)
[Clear](#)
[GetInterior*](#)
[IsSplitting*](#)
[Resize](#)
[Update](#)

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(object 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 ElmBoundary.CalcShiftedReversedBoundary(double shift,
                                             object& boundary)
```

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.

EXAMPLE

```
external object: "thisBoundary"
double shift;
int ret;
object newBoundary;
shift = 2;
ret = thisBoundary.CalcShiftedReversedBoundary(shift, newBoundary);
if (ret = 0) {
    printf('Defined boundary %o by shifting %d elements!', newBoundary, shift);
}
```

Clear

Removes all boundary cubicles from an existing boundary.

```
void ElmBoundary.Clear()
```

GetInterior*

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

```
set ElmBoundary.GetInterior()
```

RETURNS

Returns the set of interior elements.

EXAMPLE

```
external object 'newBoundary'
set interior;
object o;
interior = newBoundary.GetInterior();
o = interior.First();
while (o) {
    printf('interior object: %o \n', o);
    o = interior.Next();
}
```

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 ElmBoundary.IsSplitting([set& notSplittingCubicles])
```

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

EXAMPLE

```
set cubicles;
object cubicle;
int res;
res = boundary.IsSplitting(cubicles);
if (res) {
    printf('Boundary is splitting');
}
```

```

else {
    printf('Boundary is not splitting because of: ');
    for (cubicle = cubicles.First(); cubicle; cubicle = cubicles.Next()) {
        cubicle.ShowFullName();
    }
}

```

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.

```
void ElmBoundary.Resize(double size,
                        string name
                      )
```

ARGUMENTS

<i>size</i>	size of the referenced vector (number of cubicles)
<i>name</i>	reference to the vector ('orient' or 'cubicles')

RETURNS

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

Update

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

```
void ElmBoundary.Update()
```

3.2.7 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.

```
void ElmBranch.Update()
```

3.2.8 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

EXAMPLE

```
object cabSys;
set cabSysElements;
int err;

cabSysElements = GetCalcRelevantObjects('*.*.ElmCabsys');
cabSys = cabSysElements.First();
err = cabSys.FitParams();
if (err) {
    Error('Could not calculate line parameters for %s.', cabSys);
    exit();
}
```

GetLineCable*

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

```
object ElmCabsys.GetLineCable(int line)
```

ARGUMENTS

line Index of line.

RETURNS

cable type On success.
NULL On error.

EXAMPLE

```
object oElmSys;
set lines;
object otyp;
int indx;
```

```

indx = 1;

oElmSys = GetCalcRelevantObjects('* ElmCabsys');
cableElement = cableElements.First();
otyp = cableElement.GetLineCable(indx);
if (otyp == nullptr) {
    Error('Could not find cable type for the index: %d', indx);
    exit();
}
else {
    printf('Line index %d corresponds to the cable type %o', indx, otyp);
}

```

Update

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

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

RETURNS

- 1** On success.
- 0** On error.

EXAMPLE

```

object cable;
set cableElements;
int ierr;

cableElements = GetCalcRelevantObjects('* ElmCabsys');
cable = cableElements.First();
ierr = cable.Update();

```

3.2.9 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.

```
void ElmComp.SlotUpdate()
```

DEPRECATED NAMES

Slotupd

3.2.10 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
≠ 0	On error

EXAMPLE

The following example gathers all open switches and closes them.

```
int open;
set switches;
object switch;
switches = GetCalcRelevantObjects('*.ElmCoup');

for(switch = switches.First(); switch; switch = switches.Next()) {
    open = switch.IsOpen();
    if (open = 1) {
        switch.Close();
    }
}
```

SEE ALSO

[ElmCoup.Open\(\)](#)

GetRemoteBreakers*

Returns the remote circuit breakers or connected bus bars.

This information is determined by a topological search that starts at given breaker in all directions, generally stopping at

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

If the search reaches a busbar while only reducible components have been passed (see [object.IsReducible\(\)](#)) it stops and returns the connected busbar (no breaker found). In case of non-reducible components have been passed, the search continues until next circuit breaker is found. Only breakers with given breaker state are returned.

```
set ElmCoup.GetRemoteBreakers(int desiredBreakerState,
                               set& foundBreakers,
                               [set& foundBusbars])
```

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
≠ 0	On error

EXAMPLE

The following example gathers all closed switches and opens them.

```
int closed;
set switches;
object switch;
switches = GetCalcRelevantObjects('*.ElmCoup');

for(switch = switches.First(); switch; switch = switches.Next()) {
    closed = switch.IsClosed();
    if (closed = 1) {
        switch.Open();
    }
}
```

SEE ALSO

[ElmCoup.Close\(\)](#)

3.2.11 ElmDsl**Overview**

[ExportToClipboard](#)
[ExportToFile](#)

ExportToClipboard

Export the parameter list to clipboard.

```
void ElmDsl.ExportToClipboard([string 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).

```
void ElmDsl.ExportToFile(string filePath,
                         [string 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).

3.2.12 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

EXAMPLE

```
set feeders;
object folder,feeder;
int ierr;
! calculates the aggregated variables of all the feeders
folder = GetDataFolder('ElmFeeder');
if (folder) {
    feeders = folder.GetContents('*.*ElmFeeder',1);
    for (feeder=feeders.First(); feeder; feeder=feeders.Next()) {
        ierr = feeder.CalcAggrVarsInRadFeed();
        printf('Calculation of aggr. variables in feeder %o: %d', feeder, ierr);
    }
}
```

GetAll*

Returns a set with all objects belonging to this feeder.

```
set 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.

EXAMPLE

```
set all,feeders;
object folder,feeder,obj;
! output elements in the feeders
folder = GetDataFolder('ElmFeeder');
if (folder) {
    feeders = folder.GetContents('*.*ElmFeeder',1);
    feeders.SortToVar(0,'loc_name');
    for (feeder=feeders.First(); feeder; feeder=feeders.Next()) {
        printf('Elements in feeder %s',feeder:loc_name);
        all = feeder.GetAll(1);
        for (obj=all.First(); obj; obj=all.Next()) {
            printf('%s\\%s',obj:r:fold_id:loc_name,obj:loc_name);
        }
    }
}
```

```

    }
}
```

SEE ALSO

[object.IsInFeeder\(\)](#)

GetBranches*

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

```
set 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.

EXAMPLE

```
set aBranches, feeders;
object folder, feeder, obj;
! output elements in the feeders
folder = GetDataFolder('ElmFeeder');
if (folder) {
    feeders = folder.GetContents('.ElmFeeder',1);
    feeders.SortToVar(0,'loc_name');
    for (feeder=feeders.First(); feeder; feeder=feeders.Next()) {
        printf('Branches in feeder %s',feeder:loc_name);
        aBranches = feeder.GetBranches(1);
        for (obj=aBranches.First(); obj; obj=aBranches.Next()) {
            printf('%s\\%s',obj:r:fold_id:loc_name,obj:loc_name);
        }
    }
}
```

GetBuses*

Returns a set with all buses belonging to this feeder.

```
set 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.

EXAMPLE

```
set nodes,feeders;
object folder,feeder,obj;
! output elements in the feeders
folder = GetDataFolder('ElmFeeder');
if (folder) {
    feeders = folder.GetContents('*.*ElmFeeder',1);
    feeders.SortToVar(0,'loc_name');
    for (feeder=feeders.First(); feeder; feeder=feeders.Next()) {
        printf('Buses in feeder %s',feeder:loc_name);
        nodes = feeder.GetBuses(1);
        for (obj=nodes.First(); obj; obj=nodes.Next()) {
            printf('%s\\%s',obj:r:fold_id:loc_name,obj:loc_name);
        }
    }
}
```

GetNodesBranches*

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

```
set 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.

EXAMPLE

```
set aAll,feeders;
object folder,feeder,obj;
! output elements in the feeders
folder = GetDataFolder('ElmFeeder');
if (folder) {
    feeders = folder.GetContents('*.*ElmFeeder',1);
    feeders.SortToVar(0,'loc_name');
    for (feeder=feeders.First(); feeder; feeder=feeders.Next()) {
        printf('Branches and Nodes in feeder %s',feeder:loc_name);
        aAll = feeder.GetNodesBranches(1);
        for (obj=aAll.First(); obj; obj=aAll.Next()) {
            printf('%s\\%s',obj:r:fold_id:loc_name,obj:loc_name);
        }
    }
}
```

GetObjs*

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

```
set ElmFeeder.GetObjs(string 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.

EXAMPLE

```
set aAll,feeders;
object folder,feeder,obj;
! output elements in the feeders
folder = GetDataFolder('ElmFeeder');
if (folder) {
    feeders = folder.GetContents('*.*.ElmFeeder',1);
    feeders.SortToVar(0,'loc_name');
    for (feeder=feeders.First(); feeder; feeder=feeders.Next()) {
        printf('Cubicles in feeder %s',feeder:loc_name);
        aAll = feeder.GetObjs('StaCubic');
        for (obj=aAll.First(); obj; obj=aAll.Next()) {
            printf('%s\\%s',obj:r:fold_id:loc_name,obj:loc_name);
        }
    }
}
```

3.2.13 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.

3.2.14 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 ElmFilter.GetGroundingImpedance(int busIdx,
                                     double& resistance,
                                     double& reactance)
```

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.

3.2.15 ElmGenstat

Overview

CalcEfficiency
Derate
Disconnect
GetAvailableGenPower*
GetGroundingImpedance*
GetStepupTransformer*
IsConnected*
Reconnect
ResetDerating

CalcEfficiency

Calculate efficiency for connected storage model at given active power value.

```
double ElmGenstat.CalcEfficiency(double activePowerMW)
```

ARGUMENTS

activePowerMW
Active power value to calculate efficiency for.

RETURNS

Returns the resulting efficiency in p.u.

EXAMPLE

```
set objs;
object obj;
int err;
double eff, Ptherm, Pelect;

Pelect = 30.0;
objs = GetCalcRelevantObjects('*.ElmGenstat');

obj = objs.First();
if (obj) {
    eff = obj.CalcEfficiency(Pelect);
    Ptherm = Pelect/eff;
}
```

Derate

Derates the value of the Max. Active Power Rating according to the specified value given in MW.

The following formula is used: $P_{max_uc} = P_{max_uc} - "Deratingvalue"$.

```
void ElmGenstat.Derate(double 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)) |

EXAMPLE

```
set objs;
object obj;
int err;

objs = GetCalcRelevantObjects('*.ElmSym,*.ElmGenstat,*.ElmPvsys');

! disconnect all generators

for (obj = objs.First(); obj; obj = objs.Next()) {
    err = obj.Disconnect();
    if (err) {
        printf('Error disconnecting %s', obj);
    }
}
```

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.

```
double ElmGenstat.GetAvailableGenPower()
```

RETURNS

Available generation power

EXAMPLE

```
set generators;
object generator;
double totalpower, power;

generators = GetCalcRelevantObjects('*.ElmGenstat');

totalpower = 0; ! initialize cumulative generation

! get cumulative generation
for (generator = generators.First(); generator; generator = generators.Next()) {
    power = generator.GetAvailableGenPower();
    totalpower += power;
}
printf('Cummulative generation is %f', totalpower);
```

GetGroundingImpedance*

Returns the impedance of the internal grounding.

```
int ElmGenstat.GetGroundingImpedance(int busIdx,
                                      double& resistance,
                                      double& reactance)
```

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.

```
object ElmGenstat.GetStepupTransformer(double 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

EXAMPLE

```
set objs;
object obj;
int status;

objs = GetCalcRelevantObjects('*.ElmSym,*.ElmGenstat,*.ElmPvsys');

! print connection status for all generators

for (obj = objs.First(); obj; obj = objs.Next()) {
    err = obj.IsConnected();
    if (status) {
        printf('%s is connected', obj);
    }
    else {
        printf('%s is disconnected', obj);
    }
}
```

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

EXAMPLE

```
set objs;
object obj;
int err;
```

```

objs = GetCalcRelevantObjects('*.ElmSym,*.ElmGenstat,*.ElmPvsys');

! reconnect all generators

for (obj = objs.First(); obj; obj = objs.Next()) {
    err = obj.Reconnect();
    if (err) {
        printf('Error connecting %s', obj);
    }
}

```

ResetDerating

Resets the derating value, setting the Max. Active Power Rating according to the rating factor. The following formula is used: $P_{max_uc} = p_{maxratf} * P_n * n_{num}$.

```
void ElmGenstat.ResetDerating()
```

3.2.16 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 ElmGndswt.GetGroundingImpedance(int busIdx,
                                      double& resistance,
                                      double& reactance)
```

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\(\)](#)

3.2.17 ElmLne

Overview

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

AreDistParamsPossible*

Check if the line fulfils 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 fulfil conditions for distributed paramters
- 5 TypLne does not fulfil 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(double dis,
                                    double rem,
                                    object O,)
int ElmLne.CreateFeederWithRoutes(double dis,
                                    double rem,
                                    object O,
                                    int sw0,
                                    int sw1)
```

ARGUMENTS

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

RETURNS

0	Success, feeders created
1	Error

FitParams

Calculates distributed parameters of the line element. 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([int isRMSModel = 0,
                      [int modify = 1])
```

ARGUMENTS

isRMSModel

modify

RETURNS

0	Success
1	Error

EXAMPLE

```
object line;
set lines;
int err;

lines = GetCalcRelevantObjects('*ElmLne');
line = lines.First();
err = line.FitParams();
if (err) {
```

```

    Error('Could not calculate line parameters for %s.', line);
    exit();
}

```

GetIthr*

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

```
double ElmLne.GetIthr()
```

RETURNS

Returns rated short-time current value

EXAMPLE

```

External Object (ElmLne) pre-defined: MyLine
double Ir;
if (MyLine) {
    Ir = MyLine.GetIthr();
}
Info('Rated short-time current is %f kA', Ir);

```

GetType*

Returns the line type object.

```
object ElmLne.GetType()
```

RETURNS

The TypLne object if exists or NULL

EXAMPLE

The following example reports all 'untyped' lines

```

set list;
object line, type;
list = GetCalcRelevantObjects();
line = list.Firstmatch('ElmLne');
while (line) {
    type = line.GetType();
    if (type=nullptr) {
        line.ShowFullName();
    }
    line = list.Nextmatch();
}

```

GetY0m*

The function returns the zero-sequence mutual coupling admittance (G_0m , B_0m) in Ohm of the line and input argument line (object Lne2). When $Lne2 = line$, the function returns the zero-sequence self admittance.

```
int ElmLne.GetY0m(object Lne2,
                    double& G0m,
                    double& B0m)
```

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

EXAMPLE

```
External Object (ElmLne) pre-defined: MyLine
double fG0m, fB0m;
int iret;
if (MyLine) {
    iret = MyLine.GetY0m(MyLine, fG0m, fB0m);
}
if (iret) {
    Warn('No values for G0m and B0m obtained!');
}
else {
    Info('The value for G0m is %f Ohm and for B0m is %f Ohm', fG0m, fB0m);
}
```

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 ElmLne.GetY1m(object Lne2,
                    double& G1m,
                    double& B1m)
```

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

EXAMPLE

```
External Object (ElmLne) pre-defined: MyLine
double fG1m, fB1m;
int iret;
if (MyLine) {
    iret = MyLine.GetY1m(MyLine, fG1m, fB1m);
}
if (iret) {
    Warn('No values for G1m and B1m obtained!');
}
else {
    Info('The value for G1m is %f Ohm and for B1m is %f Ohm', fG1m, fB1m);
}
```

GetZ0m*

Gets the zero-sequence mutual coupling impedance (R_{0m} , X_{0m}) in Ohm of the line and input argument line (object *otherLine*). When *otherLine* = *line*, the function returns the zero-sequence self impedance.

```
int ElmLne.GetZ0m(object otherLine,
                    double& R0m,
                    double& X0m)
```

ARGUMENTS

otherLine Line element

R0m (out)

To be obtained R_{0m} value

X0m (out)

To be obtained X_{0m} value

RETURNS

0 Success, data obtained

1 Error, e.g. no coupling objects defined

EXAMPLE

```
External Object (ElmLne) pre-defined: MyLine
double fR0m, fX0m;
int iret;
if (MyLine) {
    iret = MyLine.GetZ0m(MyLine, fR0m, fX0m);
}
if (iret) {
    Warn('No values for R0m and X0m obtained!');
}
else {
    Info('The value for R0m is %f Ohm and for X0m is %f Ohm', fR0m, fX0m);
}
```

GetZ1m*

The function returns the positive-sequence mutual coupling impedance (R_{1m} , X_{1m}) in Ohm of the line and input argument line (object $Lne2$). When $Lne2 = \text{line}$, the function returns the positive-sequence self impedance.

```
int ElmLne.GetZ1m(object Lne2,
                    double& R1m,
                    double& X1m)
```

ARGUMENTS

Lne2 Line element

R1m (out)
Resulting R_{1m} value

X1m (out)
Resulting X_{1m} value

RETURNS

0 Success, data obtained
1 Error, e.g. no coupling objects defined

EXAMPLE

```
External Object (ElmLne) pre-defined: MyLine
double fR1m, fX1m;
int iret;
if (MyLine) {
    iret = MyLine.GetZ1m(MyLine, fR1m, fX1m);
}
if (iret) {
    Warn('No values for R1m and X1m obtained!');
}
else {
    Info('The value for R1m is %f Ohm and for X1m is %f Ohm', fR1m, fX1m);
}
```

GetZmatDist*

The function gets impedance matrix in phase domain (only amplitudes), for a line with distributed parameters, short-circuit ended.

```
int ElmLne.GetZmatDist(double frequency,
                        int exact,
                        object 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

EXAMPLE

The following example reports all lines with routes.

```
set list;
object line;
int i;
list = GetCalcRelevantObjects();
line = list.Firstmatch('ElmLne');
while (line) {
    i = line.HasRoutes();
    if (i) line.ShowFullName();
    line = list.Nextmatch();
}
```

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

EXAMPLE

The following example reports all lines with sections.

```
set list;
object line;
int i;
list = GetCalcRelevantObjects();
line = list.Firstmatch('ElmLne');
while (line) {
    i = line.HasRoutesOrSec();
    if (i=2) line.ShowFullName(); {
```

```

        line = list.Nextmatch();
    }
}

```

IsCable*

Checks if this line is a cable.

```
int ElmLne.IsCable()
```

RETURNS

- 1** Line is a cable
- 0** Line is not a cable

EXAMPLE

The following example reports the loading of all cables.

```

set list;
object cable;
int i;
list = GetCalcRelevantObjects();
cable = list.Firstmatch('ElmLne');
while (cable) {
    i = cable.IsCable();
    if (i) {
        Write('# : #.## $N', @ACC(1):loc_name, @ACC(1):c:loading, 0);
    }
    cable = list.Nextmatch();
}

```

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

EXAMPLE

```

set list;
object line;
int i;
list = GetCalcRelevantObjects();
line = list.Firstmatch('ElmLne');
while (line) {
    i = line.IsNetCoupling();
    if (i) {
        Write('# : #.## $N', @ACC(1):loc_name, @ACC(1):c:loading, line);
    }
}

```

```

    line = list.Nextmatch();
}

```

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.

```
double 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

EXAMPLE

```

double length;
object line;
set lines;

lines = GetCalcRelevantObjects('*.ElmLne');
line = lines.First();
if (.not. line) {
  Error('no line found');
  exit();
}

length = line.MeasureLength();
printf('Measured length of %o: %f', line, length);

```

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()
```

EXAMPLE

Suggested code order:

ResetCalculation() Makes sure that all previous calculations are deleted

Line.SetDetailed() Sets detailed flag for the line

Ldf.Execute() Calculation of the load flow

```
External Object (ElmLne) pre-defined: MyLine
object cmdLdf;
cmdLdf = GetFromStudyCase('ComLdf');
if (MyLine .and. cmdLdf) {
    ResetCalculation(); ! Resets previous calculation
    MyLine.SetDetailed(); ! Sets detailed flag on the line dropper
    Ldf.Execute(); ! Performs load flow calculation
    ! The detailed flag is automatically reset after the load flow calculation
    ...
}
```

3.2.18 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

EXAMPLE

The following example reports the loading of all cables.

```
set list;
object cable;
int i;
list = GetCalcRelevantObjects();
cable = list.Firstmatch('ElmLnesec');
while (cable) {
    i = cable.IsCable();
    if (i>0) {
        Write('# : #.# $N', @ACC(1):loc_name, @ACC(1):c:loading, 0);
    }
    cable = list.Nextmatch();
}
```

3.2.19 ElmNec

Overview

[GetGroundingImpedance*](#)

GetGroundingImpedance*

Returns the impedance of the internal grounding.

```
int ElmNec.GetGroundingImpedance(int busIdx,
                                  double& resistance,
                                  double& reactance)
```

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.

3.2.20 ElmNet

Overview

[Activate](#)
[CalculateInterchangeTo*](#)
[Deactivate](#)
[DefineBoundary](#)

Activate

Adds a grid to the active study case. Can only be applied if there are no currently active calculation (i.e. running contingency analysis).

```
int ElmNet.Activate()
```

RETURNS

- 0 on success
- 1 on error

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(object net)
```

ARGUMENTS

- net* Connected grid

RETURNS

- < 0 error
- = 0 grids are not connected, no interchange exists
- > 0 ok

EXAMPLE

```
external grid objects: "from", "to"
ElmNet pnetA, pnetB;
object Ldf;

pnetA = from;
pnetB = to;

Ldf = GetFromStudyCase('ComLdf');
Ldf.Execute(); ! Valid load flow calculation

int res;
res = pnetA.CalculateInterchangeTo(pnetB);
if (res > 0) {
    printf('Pinter: %d', pnetA:c:Pinter);
    printf('Qinter: %d', pnetA:c:Qinter);
    printf('ExportP: %d', pnetA:c:ExportP);
    printf('ExportQ: %d', pnetA:c:ExportQ);
    printf('ImportP: %d', pnetA:c:ImportP);
    printf('ImportQ: %d', pnetA:c:ImportQ);
}
```

Deactivate

Removes a grid from the active study case. Can only be applied if there are no currently active calculation.

```
int ElmNet.Deactivate()
```

RETURNS

- 0 on success
- 1 on error

DefineBoundary

Defines boundary with this grid as interior part. Resulting cubicles of boundary are busbar-oriented towards the grid.

```
object ElmNet.DefineBoundary(int shift)
```

ARGUMENTS

shift Elements outside the grid that are within a distance of shift many elements to a boundary cubicle of the grid are added to the interior part of the resulting boundary

RETURNS

The defined boundary is returned in case of success. Otherwise NULL, if an error appeared in the definition of the boundary.

EXAMPLE

```
external object: "thisGrid"
    double shift;
    object newBoundary;
    shift = 2;
    newBoundary = thisGrid.DefineBoundary(shift);
    if ({newBoundary <> NULL}) {
        printf('Defined boundary %o by shifting %d elements!', newBoundary, shift);
    }
```

3.2.21 ElmPvsys

Overview

[CalcEfficiency](#)
[Derate](#)
[Disconnect](#)
[GetAvailableGenPower*](#)
[GetGroundingImpedance*](#)
[IsConnected*](#)
[Reconnect](#)
[ResetDerating](#)

CalcEfficiency

Calculate efficiency for connected storage model at given active power value.

```
double ElmPvsys.CalcEfficiency(double activePowerMW)
```

ARGUMENTS

activePowerMW
 Active power value to calculate efficiency for.

RETURNS

Returns the resulting efficiency in p.u.

EXAMPLE

```

set objs;
object obj;
int err;
double eff, Ptherm, Pselect;

Pselect = 30.0;
objs = GetCalcRelevantObjects('*.*ElmPvsys');

obj = objs.First();
if (obj) {
    eff = obj.CalcEfficiency(Pselect);
    Ptherm = Pselect/eff;
}

```

Derate

Derates the value of the Max. Active Power Rating according to the specified value given in MW.

The following formula is used: $P_{max_uc} = P_{max_uc} - "Deratingvalue"$.

```
void ElmPvsys.Derate(double 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))

EXAMPLE

```

set objs;
object obj;
int err;

objs = GetCalcRelevantObjects('*.*ElmSym,*.*ElmGenstat,*.*ElmPvsys');

! disconnect all generators

for (obj = objs.First(); obj; obj = objs.Next()) {
    err = obj.Disconnect();
    if (err) {
        printf('Error disconnecting %s', obj);
    }
}

```

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.

```
double ElmPvsys.GetAvailableGenPower()
```

RETURNS

Available generation power

EXAMPLE

```
set generators;
object generator;
double totalpower, power;

generators = GetCalcRelevantObjects('*.ElmPvsys');

totalpower = 0; ! initialize cummulative generation

! get cummulative generation
for (generator = generators.First(); generator; generator = generators.Next()) {
    power = generator.GetAvailableGenPower();
    totalpower += power;
}
printf('Cummulative generation is %f', totalpower);
```

GetGroundingImpedance*

Returns the impedance of the internal grounding.

```
int ElmPvsys.GetGroundingImpedance(int busIdx,
                                     double& resistance,
                                     double& reactance)
```

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

EXAMPLE

```
set objs;
object obj;
int status;

objs = GetCalcRelevantObjects('*.ElmSym,*.ElmGenstat,*.ElmPvsys');

! print connection status for all generators

for (obj = objs.First(); obj; obj = objs.Next()) {
    err = obj.IsConnected();
    if (status) {
        printf('%s is connected', obj);
    }
    else {
        printf('%s is disconnected', obj);
    }
}
```

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

EXAMPLE

```

set objs;
object obj;
int err;

objs = GetCalcRelevantObjects('*.ElmSym,*.ElmGenstat,*.ElmPvsys');

! reconnect all generators

for (obj = objs.First(); obj; obj = objs.Next()) {
    err = obj.Reconnect();
    if (err) {
        printf('Error connecting %s', obj);
    }
}

```

ResetDerating

Resets the derating value, setting the Max. Active Power Rating according to the rating factor.
The following formula is used: $P_{max_uc} = p_{maxratf} * P_n * n_{gnum}$.

```
void ElmPvsys.ResetDerating()
```

3.2.22 ElmRelay**Overview**

```

CheckRanges*
GetCalcRX*
GetMaxFdetectCalI*
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 ElmRelay.GetCalcRX(int inSec,
                      int unit,
                      double& real,
                      double& imag)
```

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 ElmRelay.GetMaxFdetectCalCI(double& Iabs,
                                 int earth,
                                 int iunit
                                )
```

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 occurred, the output is invalid.

GetSlot*

Returns the element in the slot with the given name.

```
object ElmRelay.GetSlot(string 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 NULL.

GetUnom*

Returns the nominal voltage of the local bus of the relay.

```
double 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 occurred.

SetImpedance

Sets the the given impedance to the distance blocks matching the criteria.

```
int ElmRelay.SetImpedance(double real,
                           double imag,
                           int inSec,
                           int zone,
                           int unit
                           )
int ElmRelay.SetImpedance(double real,
                           double imag,
                           double lineAngle,
                           double 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.

```
void ElmRelay.SetMaxI()
```

SetMaxIearth

Sets the “Max. Earth Fault Current” of the relay to the currently measured value.

```
void ElmRelay.SetMaxIearth()
```

SetMinI

Sets the “Min. Phase Fault Current” of the relay to the currently measured value.

```
void ElmRelay.SetMinI()
```

SetMinIearth

Sets the “Min. Earth Fault Current” of the relay to the currently measured value.

```
void 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(int 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.

```
void ElmRelay.SlotUpdate()
```

DEPRECATED NAMES

slotupd

3.2.23 ElmRes

Overview

[AddVariable](#)
[Clear](#)
[FindColumn*](#)
[FindMaxInColumn*](#)
[FindMaxOfVariableInRow*](#)
[FindMinInColumn*](#)
[FindMinOfVariableInRow*](#)
[FinishWriting](#)
[Flush](#)
[GetDescription*](#)
[GetFirstValidObject*](#)

```

GetFirstValidObjectVariable*
GetFirstValidVariable*
GetNextValidObject*
GetNextValidObjectVariable*
GetNextValidVariable*
GetNumberOfColumns*
GetNumberOfRows*
GetObj*
GetObject*
GetObjectValue*
GetRelCase*
GetSubElmRes*
GetUnit*
GetValue*
GetVariable*
InitialiseWriting
Load*
Release*
SetAsDefault
SetObj
SetSubElmResKey
SortAccordingToColumn*
Write
WriteDraw

```

AddVariable

Adds a variable to the list of monitored variables for the Result object.

```

void ElmRes.AddVariable(object element,
                      string varname
)

```

ARGUMENTS

element An object.

varname Variable name for object O.

DEPRECATED NAMES

AddVars

EXAMPLE

The following script gets the result file named “DPL” from the active study case, defines the variables for recording (variables from summary grid), calculates a load flow and records the corresponding variables inside the result file.

```

object sumGrid,
      ldfCmd, !
      elmRes;

elmRes = GetFromStudyCase('DPL.ElmRes');
if (nullptr=elmRes) {
  exit(1);
}
ldfCmd = GetFromStudyCase('ComLdf');
if (nullptr=ldfCmd) {
  exit(1);
}

```

```

sumGrid = GetSummaryGrid();
if (nullptr=sumGrid){
    exit(1);
}
elmRes.AddVariable(sumGrid, 'c:GenP');
elmRes.AddVariable(sumGrid, 'c:GenQ');
ldfCmd.Execute();
elmRes.InitialiseWriting();
elmRes.Write();
elmRes.Flush();

```

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.

EXAMPLE

The following example deletes the data of all ElmRes stored in the active study case.

```

object case,
    elmRes;
set     resObjs;
case = GetActiveStudyCase();
if (case) {
    resObjs = case.GetContents('.ElmRes',0); ! get all ElmRes stored in case
    for (elmRes=resObjs.First(); elmRes; elmRes=resObjs.Next()) {
        elmRes.Clear();
    }
}

```

FindColumn*

Returns the index of the first header column matching the given object and/or variable name.

```

int ElmRes.FindColumn(object obj,
                      [string varName]
)
int ElmRes.FindColumn(object obj,
                      [int startCol]
)
int ElmRes.FindColumn(string varName,
                      [int startCol]
)

```

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.

EXAMPLE

```
object case,
    elmRes,
    terminal;
set resObjs,
    relevantObjs;
int index;

case = GetActiveStudyCase();
if (nullptr=case) {
    exit(1); ! there is no active study case
}
resObjs = case.GetContents('Quasi-Dynamic Simulation AC.ElmRes', 0);
elmRes = resObjs.First();
if (nullptr=elmRes) {
    exit(1);
}
relevantObjs = GetCalcRelevantObjects('*.*.ElmTerm');
terminal = relevantObjs.First();

elmRes.Load();

index = elmRes.FindColumn('m:u'); ! start at first variable
printf('First index of variable m:u %d',index);
index = elmRes.FindColumn('m:u', index+1); ! start after index already found
printf('Second index of variable m:u %d',index);

index = elmRes.FindColumn(terminal); ! start at first variable
printf('First index of object %s: %d',terminal:loc_name, index);
index = elmRes.FindColumn(terminal, index+1); ! start after index already found
printf('Second index of object %s: %d',terminal:loc_name, index);

index = elmRes.FindColumn(terminal, 'm:u');
printf('Index of variable m:u of %s: %d',terminal:loc_name, index);

elmRes.Release();
```

FindMaxInColumn*

Find the maximum value of the variable in the given column.

```
int ElmRes.FindMaxInColumn(int column,
                           [double& value])
```

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.

EXAMPLE

```
object case,
      elmRes;
set    resObjs;
int    row,
      column;
double value;

column = 1;
case = GetActiveStudyCase();
if (nullptr==case) {
    exit(1); ! there is no active study case
}
resObjs = case.GetContents('All calculations.ElmRes',0); ! get all ElmRes stored in case
elmRes = resObjs.First();
if (elmRes) {
    elmRes.Load();
    row = elmRes.FindMaxInColumn(column,value);
    if (row<0) {
        Info('The maximum of column %d was not found',column);
    }
    else {
        Info('The maximum of column %d is %f (row: %d)',column, value, row);
    }
    row = elmRes.FindMinInColumn(column,value);
    if (row<0) {
        Info('The minimum of column %d was not found',column);
    }
    else {
        Info('The minimum of column %d is %f (row: %d)',column, value, row);
    }
    elmRes.Release();
}
```

FindMaxOfVariableInRow*

Find the maximum value for the given row and variable.

```
int ElmRes.FindMaxOfVariableInRow(string variable,
                                    int row,
                                    [double& maxValue])
```

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.

EXAMPLE

The following example reports the maximum and minimum of variable m:u in the first row.

```
object case,
        elmRes,
        terminal;
set    resObjs;
int    index;
double voltage;

case = GetActiveStudyCase();
if (nullptr=case) {
    exit(1); ! there is no active study case
}
resObjs = case.GetContents('Quasi-Dynamic Simulation AC.ElmRes',0);
elmRes = resObjs.First();
if (elmRes) {
    elmRes.Load();
    index = elmRes.FindMaxOfVariableInRow('m:u',0, voltage);
    printf('Maximum m:u at column %d: %f',index,voltage);
    index = elmRes.FindMinOfVariableInRow('m:u',0, voltage);
    printf('Mainnum m:u at column %d: %f',index,voltage);
    elmRes.Release();
}
```

FindMinInColumn*

Find the minimum value of the variable in the given column.

```
int ElmRes.FindMinInColumn(int column,
                           [double& value])
```

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.

EXAMPLE

See example of [ElmRes.FindMaxInColumn](#).

FindMinOfVariableInRow*

Find the minimum value for the given row and variable.

```
int ElmRes.FindMinOfVariableInRow(string variable,
                                    int row,
                                    [double& minValue])
```

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.

EXAMPLE

See example of [ElmRes.FindMinOfVariableInRow](#).

FinishWriting

Finishes the writing of values to a result file.

```
void ElmRes.FinishWriting()
```

DEPRECATED NAMES

Close

EXAMPLE

See example of [ElmRes.InitialiseWriting](#).

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()
```

EXAMPLE

The following example writes a result object and prints the data written to the file. The DPL command contains two variables on the advanced options page:

```
double x x-value
double y y-value
```

These variables were selected in the variable definitions inside the result object which itself is stored in the DPL command. The DPL script code is as follows:

```

int xFailed,yFailed,iX,iY,row;
double xValue,yValue;
! write results to ElmRes named Results (stored in Contents)
for (x=-16; x<16; x=x+0.1) {
    y= x*x;
    Results.Write();
}

! read the data
Results.Flush(); ! if this line omitted some sample might be missing in the output
Results.Load();
iX = Results.FindColumn('b:x');! get index of column x
iY = Results.FindColumn('b:y');! get index of column y
! report values in output window
row = 0;
xFailed = Results.GetValue(xValue,row,iX); ! get the x-value in the first line
yFailed = Results.GetValue(yValue,row,iY); ! get the y-value in the first line
printf('%d %d', ix,iY);
while ({xFailed=0}.and.{yFailed=0}) {
    printf('%6.2f / %6.2f',xValue,yValue);! print the values to the output window
    row = row+1; ! next row
    xFailed = Results.GetValue(xValue,row,iX); ! get the x-value in line row
    yFailed = Results.GetValue(yValue,row,iY); ! get the z-value in line row
}
Results.Release(); ! release the result file data from memory

```

GetDescription*

Get the description of a column.

```

string 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.

EXAMPLE

See example of [ElmRes.GetVariable](#).

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,
                               [string classNames,]
                               [string variableName,]
                               [double limit,]
                               [int limitOperator,]
                               [double limit2,]
                               [int limitOperator2]
                               )
int ElmRes.GetFirstValidObject(int row,
                               set 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

EXAMPLE

```
! Find first line or generator whose loading is $>$= 80%
iCol = ResFirstValidObject(oRes, iRow, 'ElmLne,ElmSym', 'c:loading', 80, 4);
```

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([string 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

EXAMPLE

```
iCol = elmRes.GetFirstValidObjectVariable('c:loading,c:loading_st');
```

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,
                                [string 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

EXAMPLE

The following example outputs all valid indices of m:u in the first row of the results.

```

object case,
      elmRes,
      colObj;
set resObjs;
int columnIndex;
case = GetActiveStudyCase();
if (nullptr=case) {
    exit(1); ! there is no active study case
}
resObjs = case.GetContents('*.ElmRes',0);
elmRes = resObjs.First();
if (nullptr=elmRes) {
    exit(1);
}
elmRes.Load();
columnIndex = elmRes.GetFirstValidVariable(0, 'm:u');
while(columnIndex>=0) {
    printf('First Index of m:u: %d',columnIndex);
    columnIndex = elmRes.GetNextValidVariable('m:u');
}
elmRes.Release();

```

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([string classNames,
                               [string variableName,
                               [double limit,
                               [int limitOperator,
                               [double limit2,
                               [int limitOperator2]
                               )
int ElmRes.GetNextValidObject(set 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

EXAMPLE

```
! Find next line or generator whose loading is $>$= 80%
iCol = ResNextValidObject(oRes, 'ElmLne,ElmSym', 'c:loading', 80, 4);
```

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([string 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

EXAMPLE

```
iCol = elmRes.GetNextValidObjectVariable('c:loading,c:loading_st');
```

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([string 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

EXAMPLE

See example of [ElmRes.GetFirstValidVariable](#).

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.

EXAMPLE

See example of [ElmRes.InitialiseWriting](#).

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.

EXAMPLE

See example of [ElmRes.InitialiseWriting](#).

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.

```
object ElmRes.GetObj(int index)
```

ARGUMENTS

index index of the object.

RETURNS

The object found or NULL.

GetObject*

Get object of given column.

```
object 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'.

EXAMPLE

The following example prints the name of the object in the default column and of the first other column of the result file being used for simulation.

```
object case,
      elmRes,
      colObj;
set    resObjs;

case = GetActiveStudyCase();
if (nullptr=case) {
  exit(1); ! there is no active study case
}
resObjs = case.GetContents('All calculations.ElmRes',0);
elmRes = resObjs.First();
if (nullptr=elmRes) {
  exit(1);
}
elmRes.Load();
colObj = elmRes.GetObject();
Info('Object in default column: %s',colObj:loc_name);
colObj = elmRes.GetObject(0);
Info('Object in first other column: %s',colObj:loc_name);
elmRes.Release();
```

GetObjectValue*

Returns a value from a result object for row iX of curve col.

```
int ElmRes.GetObjectValue(object& o,
                         int ix,
                         [int col])
```

ARGUMENTS

o (out) The object 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 analysis, the value is invalid in case that it was not written, because it was below the recording limit. Result files created using DPL/Python are always full and will not return invalid values.

EXAMPLE

See example of [ElmRes.SortAccordingToColumn](#).

GetRelCase*

Get the contingency object for the given case number from the reliability result file.

```
object ElmRes.GetRelCase(int caseNumber)
```

ARGUMENTS

caseNumber

The reliability case number

RETURNS

Returns the contingency of case number. NULL is returned if there is no corresponding contingency.

EXAMPLE

The following example lists all contingencies stored in the result file named Reliability Enumeration stored in the active study case.

```
int caseVar,      ! column which contains contingency variable for casenum
    row,        ! current row
    failed;     ! error handling
object currObj, ! current contingency
    elmRes;   ! result file
double caseIdx; ! case index read from file

elmRes = GetFromStudyCase('Reliability Enumeration.ElmRes');
if (nullptr=elmRes) {
    Error('Result file Reliability Enumeration.ElmRes not found');
    exit(1);
}
elmRes.Load();
! get index of case column; process error
caseVar = elmRes.FindColumn('b:casenum');
if (caseVar<0) {
    Error('Result file is not a reliability result file or empty');
    exit();
}
```

```

! search the row which contains the given contingency (cnt)
SetLineFeed(0);
printf('Case Number Contingency\n');
row = 0;
failed = elmRes.GetValue(caseIdx, row, caseVar);
while (.not.failed) {
    currObj = elmRes.GetRelCase(caseIdx);
    printf(' %06d      ', caseIdx);
    if (currObj) {
        printf('%o\n', currObj);
    }
    else {
        printf('-----\n');
    }
    row+=1;
    failed = elmRes.GetValue(caseIdx, row, caseVar);
}
SetLineFeed(1);
elmRes.Release();

```

GetSubElmRes*

Get sub-result file stored inside this.

```

object ElmRes.GetSubElmRes(int value)
object ElmRes.GetSubElmRes(object obj)

```

ARGUMENTS

value The cnttime to look for

obj The pResElm to look for

RETURNS

NULL 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.

```

string ElmRes.GetUnit([int column])

```

ARGUMENTS

column (optional)

The column index. The unit of the default variable is returned if the parameter is not passed to the function.

RETURNS

Returns the unit which is empty in case that the column index is not part of the data.

EXAMPLE

See example of [ElmRes.GetVariable](#).

GetValue*

Returns a value from a result object for row iX of curve col.

```
int ElmRes.GetValue(double& d,
                     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 analysis, the value is invalid in case that it was not written, because it was below the recording limit. Result files created using DPL/Python are always full and will not return invalid values.

EXAMPLE

See example of [ElmRes.SortAccordingToColumn](#).

GetVariable*

Get variable name of column

```
string 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.

EXAMPLE

```
object case,
       elmRes;
string name,
      unit,
      short,
      long;

set    resObjs;
int   variables,
      column;
```

```

case = GetActiveStudyCase();
if (nullptr==case) {
    exit(1); ! there is no active study case
}
resObjs = case.GetContents('*.*.ElmRes',0); ! get all ElmRes stored in case
elmRes = resObjs.First();
if (nullptr==elmRes) {
    exit(1);
}
elmRes.Load();
variables = elmRes.GetNumberOfColumns();
if (variables>0) {
    printf('Name; unit; Short Description; Long Description');
    for (column=-1; column<variables; column+=1) {
        ! -1 returns default column
        name = elmRes.GetVariable(column);
        unit = elmRes.GetUnit(column);
        short= elmRes.GetDescription(column,1);
        long = elmRes.GetDescription(column);
        printf('%s; %s; %s; %s', name, unit, short, long);
    }
}
elmRes.Release();

```

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(string variableName,
                            string unit,
                            string description,
                            [string 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\(\)](#)

EXAMPLE

The following example gets the result file named DPI from the study case, the default variable to Length, runs a load flow and writes a line to the file. The default variable is set to 10 km.

```
object sumGrid, ! summary grid
    ldfCmd, ! load flow command
    elmRes; ! result file

sumGrid = GetSummaryGrid();
if (nullptr=sumGrid){
    exit(1);
}
elmRes = GetFromStudyCase('DPL.ElmRes');
if (elmRes) {
    ldfCmd = GetFromStudyCase('ComLdf');
    if (ldfCmd) {
        ldfCmd.Execute();
        elmRes.InitialiseWriting('len', 'km', 'Length', 'Len');
        elmRes.Write(10);
        elmRes.FinishWriting();
    }
}
```

Load*

Loads the data of a result object (**ElmRes**) in memory for reading.

```
void ElmRes.Load()
```

EXAMPLE

```
object case,
    elmRes;
set resObjs;
int variables,rows;

case = GetActiveStudyCase();
if (nullptr=case) {
    exit(1); ! there is no active study case
}
resObjs = case.GetContents('*.*.ElmRes',0); ! get all ElmRes stored in case
elmRes = resObjs.First();
if (nullptr=elmRes) {
    exit(1);
}
elmRes.Load();
variables = elmRes.GetNumberOfColumns();
rows      = elmRes.GetNumberOfRows();
Info('The result file %s contains %d columns and %d rows',elmRes, variables, rows);
elmRes.Release();
```

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.

```
void ElmRes.Release()
```

EXAMPLE

See example of [ElmRes.Load](#).

SetAsDefault

Sets this results object as the default results object. Plots using the default result file will use this file for displaying data.

```
void ElmRes.SetAsDefault()
```

SetObj

Adds an object to the objects assigned to the result file

```
int ElmRes.SetObj(object 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.

EXAMPLE

The following example disables terminal by terminal and calculates a load flow for every case. The outaged terminal is written using SetObj

```
int failed,
    outOfService,
    index;
object ldf, bus;
set terminals;

ldf = GetFromStudyCase('ComLdf'); ! get load flow command
terminals=GetCalcRelevantObjects('ElmTerm');
for (bus = terminals.First(); bus; bus = terminals.Next()) {
    outOfService = bus:outserv;
    if (.not.outOfService) {
        bus:outserv = 1;
        failed = ldf.Execute();
        if (.not.failed) {
            index = Results.SetObj(bus);
            Results.Write();
            printf('%o: %d',bus,index);
        }
        bus:outserv = 0;
    }
}
```

SetSubElmResKey

Assigns a value or an object to the according ElmRes parameter.

```
void ElmRes.SetSubElmResKey(int value)
void ElmRes.SetSubElmResKey(object 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.

EXAMPLE

The following example outputs column 1 and 2 for the first 50 rows. The output is repeated after sorting the data according to column 1.

```
object case,
      elmRes;
set   resObjs;
int   row,
      rows,
      column,
      failed;
double value1, value2;

case = GetActiveStudyCase();
if (nullptr=case) {
    exit(1); ! there is no active study case
}
resObjs = case.GetContents('*.*.ElmRes',0); ! get all ElmRes stored in case
elmRes = resObjs.First();
if (nullptr=elmRes) {
    exit(1);
}
elmRes.Load();
column = 1;
rows = 50;
Info('Orginal data in first %d rows of column %d and %d',rows, column, column+1);
failed = elmRes.GetValue(value1,row,column);
failed = elmRes.GetValue(value2,row,column+1);
row = 0;
while (.not.failed) {
```

```

row += 1;
printf('%04d %f %f', row, value1, value2); ! report row starting with 1
failed = elmRes.GetValue(value1, row, column);
failed = elmRes.GetValue(value2, row, column+1);
if (row > rows) {
    break;
}
failed = elmRes.SortAccordingToColumn(column);
Info('Sorted data in first %d rows in column %d and %d', rows, column, column+1);
failed = elmRes.GetValue(value1, row, column);
failed = elmRes.GetValue(value2, row, column+1);
row = 0;
while (.not.failed) {
    row += 1;
    printf('%04d %f %f', row, value1, value2); ! report row starting with 1
    failed = elmRes.GetValue(value1, row, column);
    failed = elmRes.GetValue(value2, row, column+1);
    if (row > rows) {
        break;
    }
}
elmRes.Release();

```

Write

Writes the current results to the result object.

```
int ElmRes.Write([double defaultValue])
```

RETURNS

0 on success

SEE ALSO

[ElmRes.WriteDraw\(\)](#), [ElmRes.InitialiseWriting\(\)](#), [ElmRes.FinishWriting\(\)](#)

EXAMPLE

See example of [ElmRes.InitialiseWriting](#).

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

EXAMPLE

The following example performs load-flows for a number of load levels and writes the results to the result object

```
int failed;
object ldf;
```

```

load:plini = minLoad;
ldf = GetFromStudyCase('ComLdf');
while (load:plini<maxLoad) {
    failed = ldf.Execute();
    if (failed) {
        break;
    }
    load:plini+=step;
    Results.WriteDraw();
}

```

3.2.24 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 ElmShnt.GetGroundingImpedance(int busIdx,
                                    double& resistance,
                                    double& reactance)

```

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.

3.2.25 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.

```
object ElmStactrl.GetControlledHVNode ([int index = 0])
```

ARGUMENTS

index (optional)

Index of machine (starting from 0 – ...). Default is 0.

RETURNS

object Busbar/Terminal ()

NULL not found

EXAMPLE

```
set objs
object controlledNode,gen;
int index,gens;

objs = GetCalcRelevantObjects('ElmStactrl');

for (obj = objs.First(); obj; obj = objs.Next()) {
    obj.GetSize('psym',gens); ! get no. of machines
    for (index=0;index<gens;index=index+1) {
        oStaCtrl.GetVal(gen,'psym',index);
        controlledNode = oStaCtrl.GetControlledHVNode(index);
        if (controlledNode) {
            printf('Generator: %o, Controlled HV-Node: %o',gen, controlledNode);
        }
        else {
            printf('Generator: %o, Controlled HV-Node: Not found', gen);
        }
    }
}
```

GetControlledLVNode*

Returns the corresponding voltage controlled LV node for the machine at specified index. Switch status are always considered.

```
object ElmStactrl.GetControlledLVNode (int index)
```

ARGUMENTS

index Index of machine (starting from 0 – ...).

RETURNS

object Terminal ()

NULL not found

EXAMPLE

```
set objs
object controlledNode,gen;
int index,gens;
```

```

objs = GetCalcRelevantObjects('ElmStactrl');

for (obj = objs.First(); obj; obj = objs.Next()) {
    obj.GetSize('psym', gens); ! get no. of machines
    for (index=0;index<gens;index=index+1) {
        oStaCtrl.GetVal(gen, 'psym', index);
        controlledNode = oStaCtrl.GetControlledLVNode(index);
        if (controlledNode) {
            printf('Generator: %o, Controlled LV-Node: %o', gen, controlledNode);
        }
        else {
            printf('Generator: %o, Controlled LV-Node: Not found', gen);
        }
    }
}

```

GetStepupTransformer*

Performs a topological search to find the step-up transformer of the machine at the specified index.

```

object 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

NULL 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()

```

3.2.26 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 ElmSubstat.GetSplit(int index,
                        set& mainNodes,
                        [set& connectionCubicles,
                         [set& allElements]
                        )
```

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.)

EXAMPLE

```
set nodes;
set cubicles;
set elements;
int return, index;
object obj;

return = 0;
while (return <> 1) { !loop from 0 to n until there is no more split
    return = station.GetSplit(index, nodes, cubicles, elements);

    printf('Split %d:', index);

    printf('Major Nodes:');
    obj = nodes.First();
    while(obj) {
        obj.ShowFullName();
        obj = nodes.Next();
    }

    printf('Connection Cubicles:');
    obj = cubicles.First();
    while(obj) {
        obj.ShowFullName();
        obj = cubicles.Next();
    }

    printf('All Elements:');
    obj = elements.First();
    while(obj) {
        obj.ShowFullName();
        obj = elements.Next();
    }

    index = index + 1;
}
```

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 ElmSubstat.GetSplitCal(int index,
                           set& nodes,
                           [set& connectionCubicles,
                            [set& elements]])
```

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.)

EXAMPLE

```
set nodes;
set cubicles;
set elements;
int return, index;
object obj;

return = 0;
while (return <> 1) { !loop from 0 to n until there is no more split
    return = station.GetSplitCal(index, nodes, cubicles, elements);

    if (return < 1) {
        if (return = 0) {
            printf('Split %d:', index);
        }
        else {
            printf('(Pseudo)Split %d:', index);
        }
        printf('Major Nodes:');
        obj = nodes.First();
        while(obj) {
```

```

    obj.ShowFullName();
    obj = nodes.Next();
}

printf('Connection Cubicles:');
obj = cubicles.First();
while(obj) {
    obj.ShowFullName();
    obj = cubicles.Next();
}

printf('Elements:');
obj = elements.First();
while(obj) {
    obj.ShowFullName();
    obj = elements.Next();
}
}

index = index + 1;
}

```

SEE ALSO

[ElmSubstat.GetSplit\(\)](#)

GetSplitIndex*

This function returns the index of the split that contains passed object.

```
int ElmSubstat.GetSplitIndex(object 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

EXAMPLE

```

set stations, terms;
object substation, term;
string name;
int index;

!iterates over all substations and checks belonging
!of terminals to splits of each substation

stations = GetCalcRelevantObjects('*.ElmSubstat');

for (substation = stations.First(); substation; substation = stations.Next()) {
    name = substation.GetFullName(0);
    printf('\nSubstation: %s', name);

    terms = substation.GetContents('*.ElmTerm');

    for (term = terms.First(); term; term = terms.Next()) {

```

```

index = substation.GetSplitIndex(term);
name = term.GetFullName(0);
if (index < 0) {
    printf(' %s is not contained in any split of that substation', name);
}
else {
    printf(' %s belongs to split %d', name, index);
}
}
}

```

SEE ALSO

[ElmSubstat.GetSplit\(\)](#)**GetSuppliedElements***

Returns the network components that are supplied by the transformer (located in the station). 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.

```
set 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(object ra)
```

ARGUMENTS

ra Given running arrangement

RETURNS

1	If given running arrangement was successfully overwritten;
0	otherwise

EXAMPLE

```
int res;
res = objsubstation.OverwriteRA(objra);
if (res = 1) {
    printf('%o was successfully overwritten', objra);
}
else {
    printf('%o was not overwritten', objra)
}
```

ResetRA

This function resets the running arrangement selection for the substation it was called on.

```
void 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).

```
object ElmSubstat.SaveAsRA(string 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 NULL.

EXAMPLE

```
object myRA;
myRA = objsubstation.SaveAsRA('MyRA');
if (myRA) {
    myRA.ShowFullName();
}
else {
    printf('No RA created.');
}
```

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(object 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)

EXAMPLE

```
int res;
res = objsubstation.SetRA(objra);
if (res = 1) {
    printf('%o was successfully set', objra);
}
else {
    printf('%o was not set', objra);
}
```

3.2.27 ElmSvs**Overview**

[GetStepupTransformer*](#)

GetStepupTransformer*

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

```
object ElmSvs.GetStepupTransformer(double 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).

3.2.28 ElmSym

Overview

[CalcEfficiency](#)
[Derate](#)
[Disconnect](#)
[GetAvailableGenPower*](#)
[GetGroundingImpedance*](#)
[GetMotorStartingFlag*](#)
[GetStepupTransformer*](#)
[IsConnected*](#)
[Reconnect](#)
[ResetDerating](#)

CalcEfficiency

Calculate efficiency for connected storage model at given active power value.

```
double ElmSym.CalcEfficiency(double activePowerMW)
```

ARGUMENTS

- activePowerMW*
Active power value to calculate efficiency for.

RETURNS

Returns the resulting efficiency in p.u.

EXAMPLE

```
set objs;
object obj;
int err;
double eff, Ptherm, Pelect;

Pelect = 30.0;
objs = GetCalcRelevantObjects('*.ElmSym');

obj = objs.First();
if (obj) {
    eff = obj.CalcEfficiency(Pelect);
    Ptherm = Pelect/eff;
}
```

Derate

Derates the value of the Max. Active Power Rating according to the specified value given in MW.

The following formula is used: $P_{max_uc} = P_{max_uc} - "Deratingvalue"$.

```
void ElmSym.Derate (double 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))

EXAMPLE

```
set objs;
object obj;
int err;

objs = GetCalcRelevantObjects ('*.ElmSym, *.ElmGenstat, *.ElmPvsys');

! disconnect all generators

for (obj = objs.First(); obj; obj = objs.Next()) {
    err = obj.Disconnect();
    if (err) {
        printf('Error disconnecting %s', obj);
    }
}
```

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.

```
double ElmSym.GetAvailableGenPower()
```

RETURNS

Available generation power

EXAMPLE

```
set generators;
object generator;
double totalpower, power;

generators = GetCalcRelevantObjects('*.ElmSym');

totalpower = 0; ! initialize cummulative generation

! get cummulative generation
for (generator = generators.First(); generator; generator = generators.Next()) {
    power = generator.GetAvailableGenPower();
    totalpower += power;
}
printf('Cummulative generation is %f', totalpower);
```

GetGroundingImpedance*

Returns the impedance of the internal grounding.

```
int ElmSym.GetGroundingImpedance(int busIdx,
                                  double& resistance,
                                  double& reactance)
```

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.

```
object ElmSym.GetStepupTransformer(double 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

EXAMPLE

```
set objs;
object obj;
int status;

objs = GetCalcRelevantObjects('*.ElmSym,*.ElmGenstat,*.ElmPvsyss');

! print connection status for all generators
```

```

for (obj = objs.First(); obj; obj = objs.Next()) {
    err = obj.IsConnected();
    if (status) {
        printf('%s is connected', obj);
    }
    else {
        printf('%s is disconnected', obj);
    }
}

```

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

EXAMPLE

```

set objs;
object obj;
int err;

objs = GetCalcRelevantObjects('*.ElmSym,*.ElmGenstat,*.ElmPvsys');

! reconnect all generators

for (obj = objs.First(); obj; obj = objs.Next()) {
    err = obj.Reconnect();
    if (err) {
        printf('Error connecting %s', obj);
    }
}

```

ResetDerating

Resets the derating value, setting the Max. Active Power Rating according to the rating factor. The following formula is used: $P_{max_uc} = p_{maxratf} * P_n * n_{gnum}$.

```
void ElmSym.ResetDerating()
```

3.2.29 ElmTerm

Overview

[GetBusType*](#)
[GetCalcRelevantCubicles*](#)
[GetConnectedBrkCubicles*](#)
[GetConnectedCubicles*](#)
[GetConnectedMainBuses*](#)
[GetConnectionInfo*](#)
[GetEquivalentTerminals*](#)
[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** PQ busbar.
- 2** PV busbar.
- 3** Slack busbar.

GetCalcRelevantCubicles*

This function gets calculation relevant cubicles of this terminal.

```
set ElmTerm.GetCalcRelevantCubicles()
```

RETURNS

Set of calculation relevant cubicles.

GetConnectedBrkCubicles*

Function gets the set of cubicles connected with the breaker and this terminal.

```
set ElmTerm.GetConnectedBrkCubicles([int 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.

```
set ElmTerm.GetConnectedCubicles ([int 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.

```
set ElmTerm.GetConnectedMainBuses ([int 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 ElmTerm.GetConnectionInfo (double& closedSwitches,
                               double& allSwitches,
                               double& nonSwitchingDevices,
                               double& closedAndNonSwitchingDevices,
                               double& allDevices,
                               double& connectedNodes,
                               double& mainNodes)
```

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.

GetEquivalentTerminals*

Returns a set of all terminals that are equivalent to current one. Euqivalent means that those terminals are topologically connected only by

- closed switching devices (ElmCoup, RelFuse) or
- lines of zero length (line droppers).

```
set ElmTerm.GetEquivalentTerminals()
```

RETURNS

All terminals that are equivalent to current one. Current one is also included so the set is never empty.

EXAMPLE

```
set terminals;
object curTerm;
int t;

printf('List of equivalent terminals:');
terminals = term.GetEquivalentTerminals();
for (curTerm = terminals.First(); curTerm; curTerm=terminals.Next()) {
    printf('%o', curTerm);
}
printf('End');

!checking correctness of function
for (curTerm = terminals.First(); curTerm; curTerm=terminals.Next()) {
    t = term.IsEquivalent(curTerm);
    if (t) {
        printf('%o and %o are equivalent', term, curTerm);
    }
    else {
        !will never happen
        Error('%o and %o are not equivalent', term, curTerm);
    }
}
```

SEE ALSO

[ElmTerm.IsEquivalent\(\)](#)

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).

```
double ElmTerm.GetMinDistance(object term,
                           [int considerSwitches,]
                           [set& path,]
                           [set limitToNodes,]
                           [set elementsToIgnore]
                           )
```

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).

elementsToIgnore(optional)

If given, these objects (e.g. nodes) are ignored in the search and not traversed (=considered as not existing).

RETURNS

< 0 If there is no path between the two terminals

≥ 0 Distance of shortest path in km

EXAMPLE

```
!assume terminal1 and terminal2 are objects of class
double distance;
set path;
object obj;

distance = terminal1.GetMinDistance(terminal2, 0, path);

printf('Shortest distance: %f', distance);
printf('Path:' );

for(obj = path.First(); obj; obj = path.Next()) {
    obj.ShowFullName();
}
```

GetNextHVBus*

This function returns the nearest connected busbar that has a higher voltage level. To detect this bus, a breadth-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. Two winding transformers with equal voltage on both sides are ignored (simply passed by the search).

```
object ElmTerm.GetNextHVBus()
```

RETURNS

object First busbar found.

NULL If no busbar was found.

EXAMPLE

```
object obj, result;
set bars;

bars = SEL.AllBars(); !gets all currently selected bars
obj = bars.First(); !takes first one

result = obj.GetNextHVBus();
if (result) {
    result.ShowFullName();
}
```

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.

```
string ElmTerm.GetnodeName()
```

RETURNS

Node name as described above. Never empty.

DEPRECATED NAMES

GetEllaNodeName

GetSepStationAreas*

Function gets all separate areas within the substation linked to this terminal. In this manner, area is any part between two nodes.

```
set ElmTerm.GetSepStationAreas([int 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.

EXAMPLE

```
int ival;
double dval;
object oTerm;
set allTerminals;
allTerminals = GetAll('ElmTerm');
oTerm = allTerminals.First();
while (oTerm) {
    ival = oTerm.HasCreatedCalBus();
    if (ival) {
        dval = oTerm:m:u;
        printf('Voltage of %o is %f p.u.', oTerm, dval);
    }
    oTerm = allTerminals.Next();
}
```

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 ($R \leq \text{maxR}$ and $X \leq \text{maxX}$)

```
int ElmTerm.IsElectrEquivalent(object terminal,
                               double maxR,
                               double maxX
)
```

ARGUMENTS

- terminal* Terminal to which the 'method called terminal' is connected to.
- maxR* Given threshold for the resistance of branch elements (must be given in Ohm).
- 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

EXAMPLE

```
int res;
res = Busbar1.IsElectrEquivalent(Busbar2, 0.05, 0.05);
if (res = 1) {
    printf('%o is electrical equivalent to %o', Busbar1, Busbar2);
}
else {
    printf('%o is not electrical equivalent to %o', Busbar1, Busbar2);
}
```

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 (object 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 NULL 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)

EXAMPLE

```
int res;
! Assume Busbar1 and Busbar2 are two objects of class ElmTerm
res = Busbar1.IsEquivalent(Busbar2);
if (res = 1) {
    printf('%o is equivalent to %o', Busbar1, Busbar2);
```

```

    }
else {
    printf('%o is not equivalent to %o', Busbar1, Busbar2);
}

```

SEE ALSO

[ElmTerm.GetEquivalentTerminals\(\)](#) [ElmTerm.IsElectrEquivalent\(\)](#)

IsInternalNodeInStation*

Function checks if the terminal is an internal node located 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.

DEPRECATED NAMES

IsInternalNodeInSubStation

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.

```
void ElmTerm.UpdateSubstationTerminals(int volt,
                                         int phs
                                         )
```

ARGUMENTS

- volt* Updates nominal voltages (<> 0)
- phs* Updates phase technology (<> 0)

3.2.30 ElmTr2**Overview**

[CreateEvent](#)
[GetGroundingImpedance*](#)
[GetSuppliedElements*](#)
[GetTapPhi*](#)
[GetTapRatio*](#)
[GetZ0pu*](#)
[GetZpu*](#)
[IsQuadBooster*](#)
[NTap*](#)

CreateEvent

For the corresponding transformer, a Tap Event (EvtTap) is created for the simulation.

```
int ElmTr2.CreateEvent ([int tapAction,
                        [int 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 ElmTr2.GetGroundingImpedance (int busIdx,
                                    double& resistance,
                                    double& reactance)
```

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.

```
set 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.

```
double 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 transformer for given tap position

GetTapRatio*

Gets the voltage ratio of the transformer for given tap position.

```
double 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 transformer 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.

```
void ElmTr2.GetZ0pu(int itappos,
                     double& r0pu,
                     double& x0pu,
                     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.

```
void ElmTr2.GetZpu(int itappos,
                    double& rpu,
                    double& xpu,
                    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'

EXAMPLE

```
set objsTr2;
object obj;
int isQuadBooster;
objsTr2 = GetCalcRelevantObjects('*.ElmTr2');

printf('This is a list of QBs in the network:');
for (obj = objsTr2.First(); obj; obj = objsTr2.Next())
{
    isQuadBooster = obj.IsQuadBooster();
    if(isQuadBooster=1) {
        obj.ShowFullName();
    }
}
```

NTap*

Gets the transformer tap position.

```
int ElmTr2.NTap()
```

RETURNS

The tap position.

3.2.31 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.

```
void ElmTr3.CreateEvent ([int tapAction = 2,]
                        [int tapPos = 0,]
                        [int busIdx = 0]
                        )
```

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.

GetGroundingImpedance*

Returns the impedance of the internal grounding.

```
int ElmTr3.GetGroundingImpedance (int busIdx,
                                   double& resistance,
                                   double& reactance)
```

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.

```
set 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.

```
double ElmTr3.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 transformer for given tap position and side

GetTapRatio*

Gets the voltage ratio of the transformer for given tap position and side.

```
double ElmTr3.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 transformer for given tap position and side

GetTapZDependentSide*

Get tap side used for the dependent impedance

```
int 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.

```
void ElmTr3.GetZ0pu(int itappos,
                     int iSide,
                     double& r0pu,
                     double& x0pu,
                     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.

```
void ElmTr3.GetZpu(int itappos,
                    int iSide,
                    double& rpu,
                    double& xpu,
                    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 of a given bus.

```
int ElmTr3.NTap(int bus)
```

ARGUMENTS

bus 0=HV, 1=MV, 2=LV

RETURNS

The tap position.

3.2.32 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.

```
void ElmTr4.CreateEvent ([int tapAction = 2,]
                        [int tapPos = 0,]
                        [int busIdx = 0]
                      )
```

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.

GetGroundingImpedance*

Returns the impedance of the internal grounding.

```
int ElmTr4.GetGroundingImpedance (int busIdx,
                                    double& resistance,
                                    double& reactance)
```

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.

```
set 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.

```
double 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 transformer for given tap position and side

GetTapRatio*

Gets the voltage ratio of the transformer for given tap position and side.

```
double 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 transformer for given tap position and side

GetTapZDependentSide*

Get tap side used for the dependent impedance

```
int 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.

```
void ElmTr4.GetZ0pu(int itappos,
                     int iSide,
                     double& r0pu,
                     double& x0pu,
                     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.

```
void ElmTr4.GetZpu(int itappos,
                    int iSide,
                    double& rpu,
                    double& xpu,
                    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(double busIdx)
```

ARGUMENTS

busIdx 0=HV, 1=MV, 2=LV

RETURNS

The tap position.

3.2.33 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 ElmTrfstat.GetSplit(int index,
                       set& mainNodes,
                       [set& connectionCubicles,]
                       [set& allElements]
                      )
```

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.)

EXAMPLE

```
set nodes;
set cubicles;
set elements;
int return, index;
object obj;

return = 0;
while (return <> 1) { !loop from 0 to n until there is no more split
    return = station.GetSplit(index, nodes, cubicles, elements);

    printf('Split %d:', index);

    printf('Major Nodes:');
    obj = nodes.First();
    while(obj) {
        obj.ShowFullName();
        obj = nodes.Next();
    }

    printf('Connection Cubicles:');
    obj = cubicles.First();
    while(obj) {
        obj.ShowFullName();
        obj = cubicles.Next();
    }

    printf('All Elements:');
    obj = elements.First();
    while(obj) {
        obj.ShowFullName();
        obj = elements.Next();
    }

    index = index + 1;
}
```

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 ElmTrfstat.GetSplitCal(int index,
                           set& nodes,
                           [set& connectionCubicles,
                            set& elements])
```

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 ≥ 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.)

EXAMPLE

```
set nodes;
set cubicles;
set elements;
int return, index;
object obj;

return = 0;
while (return <> 1) { !loop from 0 to n until there is no more split
    return = station.GetSplitCal(index, nodes, cubicles, elements);

    if (return < 1) {
        if (return = 0) {
            printf('Split %d:', index);
        }
        else {
            printf('(Pseudo)Split %d:', index);
        }
        printf('Major Nodes:');
        obj = nodes.First();
        while(obj) {
```

```

    obj.ShowFullName();
    obj = nodes.Next();
}

printf('Connection Cubicles:');
obj = cubicles.First();
while(obj) {
    obj.ShowFullName();
    obj = cubicles.Next();
}

printf('Elements:');
obj = elements.First();
while(obj) {
    obj.ShowFullName();
    obj = elements.Next();
}
}

index = index + 1;
}

```

SEE ALSO

[ElmTrfstat.GetSplit\(\)](#)

GetSplitIndex*

This function returns the index of the split that contains passed object.

```
int ElmTrfstat.GetSplitIndex(object 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

EXAMPLE

```

set stations, terms;
object substation, term;
string name;
int index;

!iterates over all substations and checks belonging
!of terminals to splits of each substation

stations = GetCalcRelevantObjects('*.ElmSubstat');

for (substation = stations.First(); substation; substation = stations.Next()) {
    name = substation.GetFullName(0);
    printf('\nSubstation: %s', name);

    terms = substation.GetContents('*.ElmTerm');

    for (term = terms.First(); term; term = terms.Next()) {

```

```

index = substation.GetSplitIndex(term);
name = term.GetFullName(0);
if (index < 0) {
    printf(' %s is not contained in any split of that substation', name);
}
else {
    printf(' %s belongs to split %d', name, index);
}
}
}

```

SEE ALSO

[ElmTrfstat.GetSplit\(\)](#)**GetSuppliedElements***

Returns the network components that are supplied by the transformer (located in the station). 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.

```
set 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\(\)](#)

3.2.34 ElmVac

Overview

[GetGroundingImpedance*](#)

GetGroundingImpedance*

Returns the impedance of the internal grounding. Single phase voltage source connected to neutral are considered as grounding devices themselves; i.e. instead of the dedicated grounding parameters, the R1,X1 parameters are used.

```
int ElmVac.GetGroundingImpedance(int busIdx,
                                  double& resistance,
                                  double& reactance)
```

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.

3.2.35 ElmVoltreg

Overview

[CreateEvent*](#)

[GetGroundingImpedance*](#)

[GetZpu*](#)

[NTap*](#)

CreateEvent*

For the corresponding voltage regulator, a Tap Event (EvtTap) is created for the simulation.

```
void ElmVoltreg.CreateEvent([int tapAction = 2,
                             [int tapPos = 0]
                             ])
```

ARGUMENTS

tapAction (optional)

0=increase tap; 1=decrease tap; 2=set tap to tapPos; 3=manual; 4=automatic

tapPos (optional)

Position of tap

GetGroundingImpedance*

Returns the impedance of the internal grounding.

```
int ElmVoltreg.GetGroundingImpedance(int busIdx,
                                      double& resistance,
                                      double& reactance)
```

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.

```
void ElmVoltreg.GetZpu(int itappos,
                      double& rpu,
                      double& xpu,
                      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(int tap)
```

ARGUMENTS

tap 0=Tap 1, 1=Tap 2, 2=Tap 3

RETURNS

The tap position.

3.2.36 ElmXnet

Overview

[CalcEfficiency](#)
[Disconnect](#)
[GetGroundingImpedance*](#)
[GetStepupTransformer*](#)
[Reconnect](#)

CalcEfficiency

Calculate efficiency for connected storage model at given active power value.

```
double ElmXnet.CalcEfficiency(double activePowerMW)
```

ARGUMENTS

activePowerMW

Active power value to calculate efficiency for.

RETURNS

Returns the resulting efficiency in p.u.

EXAMPLE

```
set objs;
object obj;
int err;
double eff, Ptherm, Pselect;

Pselect = 30.0;
objs = GetCalcRelevantObjects('*.ElmXnet');

obj = objs.First();
if (obj) {
    eff = obj.CalcEfficiency(Pselect);
    Ptherm = Pselect/eff;
}
```

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 ElmXnet.Disconnect()
```

RETURNS

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

EXAMPLE

```

set objs;
object obj;
int err;

objs = GetCalcRelevantObjects('*.ElmXnet,*.ElmGenstat,*.ElmPvsys');

! disconnect all generators

for (obj = objs.First(); obj; obj = objs.Next()) {
    err = obj.Disconnect();
    if (err) {
        printf('Error disconnecting %s', obj);
    }
}

```

GetGroundingImpedance*

Returns the impedance of the internal grounding.

```

int ElmXnet.GetGroundingImpedance(int busIdx,
                                    double& resistance,
                                    double& reactance)

```

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

```

object ElmXnet.GetStepupTransformer(double 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).

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 ElmXnet.Reconnect()
```

RETURNS

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

EXAMPLE

```
set objs;
object obj;
int err;

objs = GetCalcRelevantObjects('*.ElmXnet,*.ElmGenstat,*.ElmPvsys');

! reconnect all generators

for (obj = objs.First(); obj; obj = objs.Next()) {
    err = obj.Reconnect();
    if (err) {
        printf('Error connecting %s', obj);
    }
}
```

3.2.37 ElmZone**Overview**

[CalculateInterchangeTo*](#)
[DefineBoundary](#)
[GetAll*](#)
[GetBranches*](#)
[GetBuses*](#)
[GetObjs*](#)
[SetLoadScaleAbsolute](#)

CalculateInterchangeTo*

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

```
int ElmZone.CalculateInterchangeTo(object zone)
```

ARGUMENTS

- zone** zone to which the interchange is calculated

RETURNS

- < 0 calculation error (i.e. no valid load flow, empty zone...)
- 0 no interchange power to the given zone
- 1 interchange power calculated

EXAMPLE

```
object ZoneA, ZoneB; ! externally defined (i.e. input)
ZoneA = inputA;
ZoneB = inputB;
object Ldf = GetFromStudyCase('ComLdf');
Ldf.Execute();
ZoneA.CalculateInterchangeTo(ZoneB);
printf('Interchange Pinter from %o to %o is %f MW', ZoneA, ZoneB, ZoneA:c:Pinter);
```

DefineBoundary

Defines boundary with this zone as interior part. Resulting cubicles of boundary are busbar-oriented towards the zone.

```
object ElmZone.DefineBoundary(int shift)
```

ARGUMENTS

- shift** Elements outside the zone that are within a distance of shift many elements to a boundary cubicle of the zone are added to the interior part of the resulting boundary

RETURNS

The defined boundary is returned in case of success. Otherwise NULL, if an error appeared in the definition of the boundary.

EXAMPLE

```
external object: "thisZone"
double shift;
object newBoundary;
shift = 2;
newBoundary = thisZone.DefineBoundary(shift);
if ({newBoundary <> NULL}) {
    printf('Defined boundary %o by shifting %d elements!', newBoundary, shift);
}
```

GetAll*

Returns all objects which belong to this zone.

```
set ElmZone.GetAll()
```

RETURNS

The set of objects.

EXAMPLE

```

set all,zones;
object prj,zone,obj;
! output elements in the zone
prj = GetActiveProject();
if (prj) {
    zones = prj.GetContents('*.ElmZone',1);
    zones.SortToVar(0,'loc_name');
    for (zone=zones.First(); zone; zone=zones.Next()) {
        printf('Elements in zone %s',zone:loc_name);
        all = zone.GetAll();
        for (obj=all.First(); obj; obj=all.Next()) {
            printf('%s\\%s',obj:r:fold_id:loc_name,obj:loc_name);
        }
    }
}

```

GetBranches*

Returns all branches which belong to this zone.

```
set ElmZone.GetBranches()
```

RETURNS

The set of branch objects.

EXAMPLE

```

set all,zones;
object prj,zone,obj;
! output elements in the zone
prj = GetActiveProject();
if (prj) {
    zones = prj.GetContents('*.ElmZone',1);
    zones.SortToVar(0,'loc_name');
    for (zone=zones.First(); zone; zone=zones.Next()) {
        printf('Branches in zone %s',zone:loc_name);
        all = zone.GetBranches();
        for (obj=all.First(); obj; obj=all.Next()) {
            printf('%s\\%s',obj:r:fold_id:loc_name,obj:loc_name);
        }
    }
}

```

GetBuses*

Returns all buses which belong to this zone.

```
set ElmZone.GetBuses()
```

RETURNS

The set of objects.

GetObjs*

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

```
set ElmZone.GetObjs(string classname)
```

ARGUMENTS

classname

name of the class (i.e. "ElmTr2")

RETURNS

The set of contained objects.

EXAMPLE

```
set all,zones;
object prj,zone,obj;
! output cubicles in the zone
prj = GetActiveProject();
if (prj) {
    zones = prj.GetContents('.ElmZone',1);
    zones.SortToVar(0,'loc_name');
    for (zone=zones.First(); zone; zone=zones.Next()) {
        printf('Cubicles in zone %s',zone:loc_name);
        all = zone.GetObjs('StaCubic');
        for (obj=all.First(); obj; obj=all.Next()) {
            printf('%s\\%s',obj:r:fold_id:loc_name,obj:loc_name);
        }
    }
}
```

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.

```
void ElmZone.SetLoadScaleAbsolute(double Pin)
```

ARGUMENTS

Pin active power in MW used for the load scaling factor.

3.3 Station Elements

3.3.1 StaCt

Overview

[SetPrimaryTap](#)

SetPrimaryTap

Determines the best matching primary tap for the connected branch, so that $I_{Nom,Branch} \cdot mltFactor \leq I_{Pri}$. If no tap satisfies the equation, the largest tap is used.

```
int StaCt.SetPrimaryTap([double mltFactor])
```

ARGUMENTS

mltFactor (optional)
Multiplication factor (default 1.0)

RETURNS

0	Correctly set.
1	Error.

3.3.2 StaCubic

Overview

AddBreaker
GetAll*
GetBranch*
GetConnectedMajorNodes*
GetConnections*
GetNearestBusbars*
GetPathToNearestBusbar*
GetRemoteBorderCubicles
GetSwitch
IsClosed*
IsConnected*
RemoveBreaker

AddBreaker

This function creates a new switch (StaSwitch) inside this cubicle. The switch is created only if no switch previously existed.

A switch object created by this function is always of usage 'circuit-breaker' and its state is 'closed'.

```
object StaCubic.AddBreaker()
```

RETURNS

- StaSwitch object in case a new switch was created
- NULL if no object was created. This means a StaSwitch object already exists.

EXAMPLE

```
set cubics;
object cubic, staSwitch;
cubics = GetCalcRelevantObjects('*.*StaCubic');
!create StaSwitches in all cubicles that do not contain a switch yet
for(cubic = cubics.First(); cubic; cubic = cubics.Next()) {
    staSwitch = cubic.AddBreaker();
    if (staSwitch) {
        staSwitch.ShowFullName();
    }
}
```

```
}
```

GetAll*

This function returns a set of network components that are collected by a topological traversal starting from this cubicle.

```
set 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.

EXAMPLE

For a defined variable "cubic" pointing to a cubicle in a net:

```
set components;
object component;

components = cubic.GetAll(); !same as cubic.GetAll(1, 0);

!Pass open switches:
!components = cubic.GetAll(1, 1);

!Walk in opposite direction:
!components = cubic.GetAll(0); !equivalent to cubic.GetAll(0, 0);

!Walk in opposite direction and ignore switch states:
!components = cubic.GetAll(0, 1);

components.MarkInGraphics();

for (component = components.First(); component; component = components.Next()) {
    component.ShowFullName();
}
```

GetBranch*

Function gets the branch of this cubicle.

```
object 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.

```
set StaCubic.GetConnectedMajorNodes ([double 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.

EXAMPLE

```
object substat, cub, obj;
set nodes, buses, cubicles, elements, allCubicles, terms;
int index, return;
string name;

!displays all connected major nodes for all connection cubicles
!of all substations

nodes = GetCalcRelevantObjects('* ElmSubstat');

for (substat = nodes.First(); substat; substat = nodes.Next()) {
    allCubicles.Clear();

    index = 0;
    return = 0;
    while (return <> 1) {
        return = substat.GetSplit(index, buses, cubicles, elements);
        if (return = 0) {
            allCubicles.Add(cubicles);
        }
        index = index +1;
    }
    for(cub = allCubicles.First(); cub; cub = allCubicles.Next()) {
        name = cub.GetFullName(0);
        printf('\nMajor Nodes cubicle %s is connected to:', name);
```

```
    terms = cub.GetConnectedMajorNodes(0);
    for (obj = terms.First(); obj; obj = terms.Next()) {
        obj.ShowFullName();
    }
}
```

GetConnections*

Function gets all elements connected with this cubicle.

```
set 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.

```
void StaCubic.GetNearestBusbars(set< connectedBusbars,
                                set< connectableBusbars,
                                int searchDirection,
                                [int excludeZPUs])
```

ARGUMENTS

connectedBushars (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

`excludeZPLs (optional)`

ZPI (optional)
Whether ZPI (ElmZpu) should be ignored in the search. Default=0

EXAMPLE

```

set switches, connected, connectable;
object switch, cubicle, element;
int direction;

!Search in all directions
direction = 0;

switches = GetCalcRelevantObjects('ElmCoup');

for(switch = switches.First(); switch; switch = switches.Next()){
    cubicle = switch:bus1;

    if(cubicle){
        cubicle.GetNearestBusbars(connected, connectable, direction);

        printf('Connected busbars to %o:', cubicle);
        for(element = connected.First(); element; element = connected.Next()){
            printf('%o', element);
        }

        printf('Connectable busbars to %o:', cubicle);
        for(element = connectable.First(); element; element = connectable.Next()){
            printf('%o', 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.

```
set StaCubic.GetPathToNearestBusbar(object nearestBusbar, [int excludeZPUs])
```

ARGUMENTS*nearestBusbar*

Nearest busbar in relation to cubicle.

excludeZPUs (optional)

Whether ZPU (ElmZpu) should be ignored in the search. Default=0.

RETURNS

Net elements of the path from cubicle to busbar.

EXAMPLE

```

set switches, connected, connectable, path;
object switch, cubicle, busbar, element;
int direction;

!Search in all directions
direction = 0;

```

```

switches = GetCalcRelevantObjects('ElmCoup');

for(switch = switches.First(); switch; switch = switches.Next()){
    cubicle = switch:bus1;

    if(cubicle){
        cubicle.GetNearestBusbars(connected, connectable, direction);

        for(busbar = connected.First(); busbar; busbar = connected.Next()){
            path = cubicle.GetPathToNearestBusbar(busbar);

            printf('Path from %o to busbar %o:', cubicle, busbar);
            for(element = path.First(); element; element = path.Next()){
                printf('%o', element);
            }
        }
    }
}

```

GetRemoteBorderCubicles

This function returns all border cubicles of opposing substations that can be reached when starting a topological search on a cubicle. Search is always branch oriented.

```

void StaCubic.GetRemoteBorderCubicles(set& outCubicles,
                                      set& outNodes,
                                      int considerSwitchStates)

```

ARGUMENTS

outCubicles (out)

outNodes (out)

considerSwitchStates (out)

EXAMPLE

```

set nodes, cubicles;
set remoteCubicles, remoteNodes;
int return, index, i, count;
object obj, obj2, obj3;

return = 0;
while (1){ !loop from 0 to n until there is no more split

    return = substation.GetSplit(index, nodes, cubicles, elements);
    if (return = 1) {
        break;
    }

    printf('-----');
    printf('Split %d:', index);
    printf('\nMajor Nodes:');

```

```

obj = nodes.First();
while(obj) {
    obj.ShowFullName();
    obj = nodes.Next();
}

printf('\nConnection Cubicles:');
obj = cubicles.First();
while(obj) {
    obj.ShowFullName();
    obj.GetRemoteBorderCubicles(remoteCubicles, remoteNodes, 0);
    count = remoteCubicles.Count();
    printf('  Remote:');
    for(i=0; i<count; i++) {
        obj2 = remoteCubicles.Obj(i);
        obj3 = remoteNodes.Obj(i);
        printf('    cubicle: %component node %component', obj2, obj3);
    }
    obj = cubicles.Next();
}
index = index + 1;
}

```

GetSwitch

This function returns the switch stored in the cubicle.

```
object StaCubic.GetSwitch()
```

RETURNS

Switch object.

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(object 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.

RemoveBreaker

This function deletes all switches stored in the cubicle.

```
void StaCubic.RemoveBreaker()
```

EXAMPLE

```
set cubics;
object cubic;
cubics = GetCalcRelevantObjects('*.StaCubic');
!delete StaSwitches from all cubicles
for(cubic = cubics.First(); cubic; cubic = cubics.Next()) {
    cubic.RemoveBreaker();
}
```

3.3.3 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.

```
void StaExtbrkmea.CopyExtMeaStatusToStatusTmp()
```

GetMeaValue

Returns the value for the switch position currently stored in the measurement object.

```
int StaExtbrkmea.GetMeaValue(double& value)
```

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.GetStatus()
```

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.

```
void 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.

```
void 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.

```
void 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(double 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

```
void 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.

```
void 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.

```
void 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.

```
void 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

3.3.4 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.

```
void StaExtcmdmea.CopyExtMeaStatusToStatusTmp()
```

GetMeaValue

Returns the value for command interpreted as floating point value.

```
int StaExtcmdmea.GetMeaValue(double& value)
```

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.GetStatus()
```

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.

```
void 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.

```
void 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.

```
void 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(double 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

```
void 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.

```
void 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.

```
void 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.

```
void 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 |

3.3.5 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.

```
void 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.

```
void StaExtdatmea.CreateEvent()
```

GetMeaValue

Returns the value stored in the measurement object.

```
int StaExtdatmea.GetMeaValue(double& value,
                               int unused,
                               int applyMultipliator)
```

ARGUMENTS

value (out)

Value, optionally modified by configured multiplicator

unused Not used.

applyMultipliator

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.GetStatus()
```

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.

```
void 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.

```
void 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.

```
void 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(double 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

```
void 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.

```
void 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.

```
void 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.

```
void 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 |

3.3.6 StaExtfmea

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.

```
void StaExtfmea.CopyExtMeaStatusToStatusTmp()
```

GetMeaValue

Returns the value for frequency currently stored in the measurement object.

```
int StaExtfmea.GetMeaValue(double& value)
```

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.GetStatus()`**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.

`void 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.

```
void 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.

```
void 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(double 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

```
void 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.

```
void 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.

```
void 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.

```
void 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

3.3.7 StaExtfuelmea**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.

```
void StaExtfuelmea.CopyExtMeaStatusToStatusTmp()
```

GetMeaValue

Returns the value for fuel currently stored in the measurement object.

```
int StaExtfuelmea.GetMeaValue(double& value,
                               int unused,
                               int applyMultipliator)
```

ARGUMENTS

value (out)

Value for fuel, optionally multiplied by configured multiplicator

unused Not used.*applyMultipliator*

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.GetStatus()
```

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.

```
void 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.

```
void 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.

```
void 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(double 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

```
void 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.

```
void 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.

```
void 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.

```
void 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

3.3.8 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.

```
void StaExtimea.CopyExtMeaStatusToStatusTmp()
```

GetMeaValue

Returns the value for current currently stored in the measurement object.

```
int StaExtimea.GetMeaValue(double& value,
                           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.GetStatus()
```

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.

```
void 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 [StaExtmea.SetStatus\(\)](#) for details on the status bits.

```
void StaExtmea.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 [StaExtmea.SetStatus\(\)](#) for details on the status bits.

```
void StaExtmea.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 StaExtmea.SetMeaValue(double 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

```
void StaExtmea.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 [StaExtmea.SetStatus\(\)](#) for details on the status bits.

```
void StaExtmea.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 [StaExtmea.SetStatus\(\)](#) for details on the status bits.

```
void StaExtmea.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 [StaExtmea.setStatus\(\)](#) for details on the status bits.

```
void StaExtmea.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 StaExtmea.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 StaExtmea.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

3.3.9 StaExtpfmea

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.

```
void StaExtpfmea.CopyExtMeaStatusToStatusTmp()
```

GetMeaValue

Returns the value for power factor currently stored in the measurement object.

```
int StaExtpfmea.GetMeaValue(double& value,
                             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.GetStatus()
```

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.

```
void 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.

```
void 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.

```
void 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(double 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

```
void 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.

```
void 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.

```
void 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.

```
void 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

3.3.10 StaExtpmea

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.

```
void StaExtpmea.CopyExtMeaStatusToStatusTmp ()
```

GetMeaValue

Returns the value for active power stored in the measurement object.

```
int StaExtpmea.GetMeaValue(double& value,
                           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.GetStatus ()
```

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.

```
void 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.

```
void 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.

```
void 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(double 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

```
void 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.

```
void 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.

```
void 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.

```
void 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

3.3.11 StaExtqmea

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.

```
void StaExtqmea.CopyExtMeaStatusToStatusTmp ()
```

GetMeaValue

Returns the value for reactive power currently stored in the measurement object.

```
int StaExtqmea.GetMeaValue(double& value,
                           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.GetStatus ()
```

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.

```
void 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.

```
void 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.

```
void 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(double 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

```
void 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.

```
void 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.

```
void 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.

```
void 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

3.3.12 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.

```
void 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.GetStatus()
```

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.

```
void 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.

```
void 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.

```
void 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(double 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

```
void 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.

```
void 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.

```
void 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.

```
void 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 |

3.3.13 StaExtapmea

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.

```
void StaExttapmea.CopyExtMeaStatusToStatusTmp()
```

GetMeaValue

Returns the value for tap position and tap info currently stored in the measurement object.

```
int StaExttapmea.GetMeaValue(double& value,
                               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.GetStatus()
```

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.

```
void 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.

```
void 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.

```
void 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(double 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

```
void 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.

```
void 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.

```
void 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.

```
void 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

3.3.14 StaExtv3mea

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.

```
void StaExtv3mea.CopyExtMeaStatusToStatusTmp()
```

GetMeaValue

Returns the value for voltage currently stored in the measurement object.

```
int StaExtv3mea.GetMeaValue(double& value,
                             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.GetStatus()
```

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.

```
void 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.

```
void 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.

```
void 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(double 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

```
void 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.

```
void 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.

```
void 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.

```
void 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

3.3.15 StaExtvmea

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.

```
void StaExtvmea.CopyExtMeaStatusToStatusTmp ()
```

GetMeaValue

Returns the value for voltage currently stored in the measurement object.

```
int StaExtvmea.GetMeaValue(double& value,
                           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.GetStatus ()
```

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.

```
void 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.

```
void 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.

```
void 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(double 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

```
void 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.

```
void 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.

```
void 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.

```
void 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

3.3.16 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
≠ 0 On error

EXAMPLE

The following example gathers all open switches and closes them.

```
int open;
set switches;
object switch;
switches = GetCalcRelevantObjects('*.*.StaSwitch');

for(switch = switches.First(); switch; switch = switches.Next()) {
    open = switch.IsOpen();
    if (open = 1) {
        switch.Close();
    }
}
```

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
- ≠ 0** On error

EXAMPLE

The following example gathers all closed switches and opens them.

```
int closed;
set switches;
object switch;
switches = GetCalcRelevantObjects('*.*.StaSwitch');

for(switch = switches.First(); switch; switch = switches.Next()) {
    closed = switch.IsClosed();
    if (closed = 1) {
        switch.Open();
    }
}
```

SEE ALSO

[StaSwitch.Close\(\)](#)

3.4 Commands

Overview

[Execute](#)

Execute

Executes the command.

```
int Com*.Execute()
```

3.4.1 ComAddlabel

Overview

[Execute](#)

Execute

This function executes the Add Statistic Labels command itself for a given plot and curve.

```
int ComAddlabel.Execute(object plot, 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 NULL.
- 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 occurred (plot is incomplete).

EXAMPLE

The following script searches for the plot named T1 on page Voltage and adds a statistic label for the second curve.

```
object command,
desktop,
page,
plot;

command = GetFromStudyCase('ComAddlabel');
desktop = GetGraphBoard();
page = desktop.GetPage('Voltages', 0);
plot = page.GetVI('T1', 'VisPlot', 0);
command.Execute(plot, 1);
```

3.4.2 ComAddon

Overview

[CreateModule](#)
[DefineDouble](#)
[DefineDoubleMatrix](#)
[DefineDoublePerConnection](#)
[DefineDoubleVector](#)
[DefineDoubleVectorPerConnection](#)
[DefineInteger](#)
[DefineIntegerPerConnection](#)
[DefineIntegerVector](#)
[DefineIntegerVectorPerConnection](#)
[DefineObject](#)
[DefineObjectPerConnection](#)
[DefineObjectVector](#)
[DefineObjectVectorPerConnection](#)
[DefineString](#)
[DefineStringPerConnection](#)
[DeleteModule](#)
[FinaliseModule](#)
[GetActiveModule](#)
[ModuleExists](#)
[SetActiveModule](#)

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(string class,  
                           string name,  
                           string desc,  
                           string unit,  
                           double initial)
```

ARGUMENTS

<i>class</i>	The type of objects for which the new parameter is to be created, e.g. ElmLne for the line.
<i>name</i>	Name of the new parameter
<i>desc</i>	Parameter description
<i>unit</i>	Parameter's unit
<i>initial</i>	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,
                                string name,
                                string desc,
                                string unit,
                                double initial,
                                int rows,
                                int columns)
```

ARGUMENTS

<i>class</i>	The type of objects for which the new parameter is to be created, e.g. ElmLne for the line.
<i>name</i>	Name of the new parameter.
<i>desc</i>	Parameter description.
<i>unit</i>	Parameter's unit.
<i>initial</i>	Default value for all entries of new parameter.
<i>rows</i>	Number of initial rows. Number of rows will be 0 if a value smaller than 0 is given.
<i>columns</i>	Number of initial columns. Number of columns 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.

DefineDoublePerConnection

Creates a new floating-point-number parameter for every connection for the given type of objects.

```
int ComAddon.DefineDoublePerConnection(string class,
                                       string name,
                                       string desc,
                                       string unit,
                                       double initial)
```

ARGUMENTS

<i>class</i>	The type of objects for which the new parameter is to be created, e.g. ElmLne for the line.
<i>name</i>	Name of the new parameter
<i>desc</i>	Parameter description
<i>unit</i>	Parameter's unit
<i>initial</i>	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(string class,
                                string name,
                                string desc,
                                string unit,
                                double initial,
                                int size)
```

ARGUMENTS

<i>class</i>	The type of objects for which the new parameter is to be created, e.g. ElmLne for the line.
<i>name</i>	Name of the new parameter
<i>desc</i>	Parameter description
<i>unit</i>	Parameter's unit
<i>initial</i>	Default value of new parameter
<i>size</i>	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(string class,
                                             string name,
                                             string desc,
                                             string unit,
                                             double initial,
                                             int size)
```

ARGUMENTS

<i>class</i>	The type of objects for which the new parameter is to be created, e.g. ElmLne for the line.
<i>name</i>	Name of the new parameter
<i>desc</i>	Parameter description
<i>unit</i>	Parameter's unit
<i>initial</i>	Default value of new parameter
<i>size</i>	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\(\)](#)

DefinInteger

Creates a new integer parameter for the given type of objects.

```
int ComAddon.DefineInteger(string class,
                           string name,
                           string desc,
                           string unit,
                           int initial)
```

ARGUMENTS

<i>class</i>	The type of objects for which the new parameter is to be created, e.g. ElmLne for the line.
<i>name</i>	Name of the new parameter
<i>desc</i>	Parameter description
<i>unit</i>	Parameter's unit
<i>initial</i>	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(string class,
                                         string name,
                                         string desc,
                                         string unit,
                                         int initial)
```

ARGUMENTS

<i>class</i>	The type of objects for which the new parameter is to be created, e.g. ElmLne for the line.
<i>name</i>	Name of the new parameter
<i>desc</i>	Parameter description
<i>unit</i>	Parameter's unit
<i>initial</i>	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(string class,
                                 string name,
                                 string desc,
                                 string unit,
                                 int initial,
                                 int size)
```

ARGUMENTS

<i>class</i>	The type of objects for which the new parameter is to be created, e.g. ElmLne for the line.
<i>name</i>	Name of the new parameter
<i>desc</i>	Parameter description
<i>unit</i>	Parameter's unit
<i>initial</i>	Default value of new parameter
<i>size</i>	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(string class,
                                              string name,
                                              string desc,
                                              string unit,
                                              int initial,
                                              int size)
```

ARGUMENTS

<i>class</i>	The type of objects for which the new parameter is to be created, e.g. ElmLne for the line.
<i>name</i>	Name of the new parameter

<i>desc</i>	Parameter description
<i>unit</i>	Parameter's unit
<i>initial</i>	Default value of new parameter
<i>size</i>	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(string class,
                          string name,
                          string desc,
                          object initial)
```

ARGUMENTS

<i>class</i>	The type of objects for which the new parameter is to be created, e.g. ElmLne for the line.
<i>name</i>	Name of the new parameter
<i>desc</i>	Parameter description
<i>initial</i>	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(string class,
                                         string name,
                                         string desc,
                                         object initial)
```

ARGUMENTS

<i>class</i>	The type of objects for which the new parameter is to be created, e.g. ElmLne for the line.
<i>name</i>	Name of the new parameter
<i>desc</i>	Parameter description
<i>initial</i>	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(string class,
                                string name,
                                string desc,
                                object initial,
                                int size)
```

ARGUMENTS

<i>class</i>	The type of objects for which the new parameter is to be created, e.g. ElmLne for the line.
<i>name</i>	Name of the new parameter.
<i>desc</i>	Parameter description.
<i>initial</i>	Default value of new parameter.
<i>size</i>	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(string class,
                                             string name,
                                             string desc,
                                             object initial,
                                             int size)
```

ARGUMENTS

<i>class</i>	The type of objects for which the new parameter is to be created, e.g. ElmLne for the line.
<i>name</i>	Name of the new parameter.
<i>desc</i>	Parameter description.
<i>initial</i>	Default value of new parameter.
<i>size</i>	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(string class,
                           string name,
                           string desc,
                           string unit,
                           string initial)
```

ARGUMENTS

<i>class</i>	The type of objects for which the new parameter is to be created, e.g. ElmLne for the line.
<i>name</i>	Name of the new parameter
<i>desc</i>	Parameter description
<i>unit</i>	Parameter's unit
<i>initial</i>	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(string class,
                                         string name,
                                         string desc,
                                         string unit,
                                         string initial)
```

ARGUMENTS

<i>class</i>	The type of objects for which the new parameter is to be created, e.g. ElmLne for the line.
<i>name</i>	Name of the new parameter
<i>desc</i>	Parameter description
<i>unit</i>	Parameter's unit
<i>initial</i>	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 method 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.

`string 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\(\)](#)

3.4.3 ComAmpacity**Overview**

[ExecuteAmpacityCalc](#)

ExecuteAmpacityCalc

The function executes ampacity calculation with or without the tabular report at the end.

```
int ComAmpacity.ExecuteAmpacityCalc([int ireport = 1])
```

ARGUMENTS

ireport (optional)

Show report or not, default = 1.

EXAMPLE

```
object cmdAmpacity;
int iret, ireport;
cmdAmpacity = GetFromStudyCase('ComAmpacity');
if (cmdAmpacity) {
    ireport = 0;
    iret = cmdAmpacity.ExecuteAmpacityCalc(ireport);
    if (iret = 0) {
        ! tabular report not displayed since ireport = 0
    }
}
```

3.4.4 ComAuditlog

Overview

[Check](#)

Check

Checks integrity of Audit Log.

```
int ComAuditlog.Check()
```

RETURNS

number of errors

3.4.5 ComBoundary

Overview

[GetCreatedBoundaries](#)

GetCreatedBoundaries

Gets created boundaries after the command has been executed.

```
set ComBoundary.GetCreatedBoundaries()
```

RETURNS

Container of created boundaries.

3.4.6 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(object terminal,
                               double phtech,
                               double 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 success.
- 1** An error occurred during load flow execution.

EXAMPLE

```

! This example connects a 0.25 Mvar three-phase shunt to obus
! and executes load flow.
! Results can be compared to a load flow study after actually
! inserting and connecting a 0.4 kV, 0.25 Mvar shunt to the
! specified bus.
! NOTES:
! 1. Open Application Example "LV Distribution Network"
! 2. Open the "Optimal Capacitor Placement" command and
!    select feeder "FD_242". Close command.
! 3. Define "obus" as an external object in the "Basic Options"
!    page of the DPL Command
! 4. Select terminal "ND_2406" (0.4 kV System) as "obus"
object comcapo, comldf;
int iret;

iret = 0;
comcapo = GetFromStudyCase('ComCapo');
comldf = GetFromStudyCase('ComLdf');

if ({comldf}.and.{comcapo}) {
    comldf:iopt_net = 0; ! balanced load flow
    ! obus is an external terminal
    iret = comcapo.ConnectShuntToBus(obus,0,0.25);
}

printf('Error: %d', iret);

```

LossCostAtBusTech

Returns the losses cost of the selected terminal and configuration calculated during the sensitivity analysis or the optimization.

```
double ComCapo.LossCostAtBusTech(object terminal,
                                  double 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

EXAMPLE

```

! This example gets the losses cost for the buses in the
! feeder with single-phase (A-B) configuration considered
! in the sensitivity analysis or in the optimization process.
! NOTES:
! 1. Open Application Example "LV Distribution Network"
! 2. Open the "Optimal Capacitor Placement" command and
!    select feeder "FD_242".
! 3. In the "Available capacitors" page of the dialog,
!    add one single phase, 0.05 Mvar capacitor and
!    add one three-phase, 0.05 Mvar capacitor
!    both with cost of 10 $/kWh.
! 4. Close command.
! 5. Define "ofeed" as an external object in the "Basic Options"
!    page of the DPL Command
! 6. Select feeder "FD_242" as "ofeed"
object comcapo, pObj;
set aNodes;
int dret;

dret = 0;
comcapo = GetFromStudyCase('ComCapo');
comcapo:iopt_meth = 1; ! Sensitivity analysis
comcapo:iopt_ldf = 1; ! Unbalanced
comcapo.Execute();

if (comcapo) {
    aNodes = ofeed.GetBuses(1);
    aNodes.SortToName(0);
    for (pObj=aNodes.First(); pObj; pObj=aNodes.Next()) {
        dret = comcapo.LossCostAtBusTech(pObj,1); ! A-B configuration
        if (dret >= 0.) {
            printf('%o: Losses Cost: %f', pObj, dret);
        }
    }
}

```

TotalLossCost

Returns the total cost calculated after the sensitivity analysis or the optimization.

```
double 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.

EXAMPLE

```
! This example gets the different costs calculated
! after the optimization process.
! NOTES:
! 1. Open Application Example "LV Distribution Network"
! 2. Open the "Optimal Capacitor Placement" command and
!    select feeder "FD_242".
! 3. In the "Available capacitors" page of the dialog,
!    add one single phase, 0.05 Mvar capacitor and
!    add one three-phase, 0.05 Mvar capacitor,
!    both with cost of 10 $/kWh.
! 4. Close command.
object comcapo;
int dret;

comcapo = GetFromStudyCase('ComCapo');
comcapo:iopt_meth = 0; ! Optimization
comcapo:iopt_ldf = 1; ! Unbalanced
comcapo:iopt_con = 0; ! Only do report
comcapo.Execute();

if (comcapo) {
    dret = comcapo.TotalLossCost(0);
    printf('Losses in MW: %f', dret);
    dret = comcapo.TotalLossCost(1);
    printf('Cost of Losses: %f', dret);
    dret = comcapo.TotalLossCost(2);
    printf('Cost of V Violations: %f', dret);
    dret = comcapo.TotalLossCost(3);
    printf('Cost of Shunts: %f', dret);
}
```

3.4.7 ComCheck

Overview

[GetNextLoop](#)

GetNextLoop

Get the next loop for any non-radial feeder.

```
set ComCheck.GetNextLoop()
```

RETURNS

Returns the elements in of the loop.

3.4.8 ComCimdbexp

Overview

[Execute](#)

Execute

Executes the export. In case of a validation error the export is not performed.

```
int ComCimdbexp.Execute([int validate])
```

ARGUMENTS

validate (optional)

Option to execute CIM Data Validation before export. If not provided, the validation is executed. Possible values are

- 0 Do not validate
- 1 Validate

RETURNS

- 0 OK
- 1 Error: export failed

3.4.9 ComCimdbimp

Overview

[Execute](#)

[ImportAndConvert](#)

Execute

Executes the import.

```
int ComCimdbimp.Execute([int validate])
```

ARGUMENTS

validate (optional)

Option to execute CIM Data Validation after import. If not provided, the validation is executed. Possible values are

- 0** Do not validate
- 1** Validate

RETURNS

- 0** OK
- 1** Error: import failed

ImportAndConvert

Imports CIM data from file path provided in the 'CIM Data Import' command without storing CIM data into database, and converts CIM to Grid using the 'CIM to Grid Conversion' command object provided in the function call as template. The CIM to Grid Conversion will be executed with default settings if the command object is not provided in the function call.

```
int ComCimdbimp.ImportAndConvert([int validate],
                                  [object cimToGrid])
```

ARGUMENTS

validate (optional)

Option to execute CIM Data Validation after import. If not provided, the validation is executed. Possible values are

- 0** Do not validate
- 1** Validate

cimToGrid (optional)

ComCimtogrid object with preconfigured settings for converting imported CIM data. If not provided, the default CIM to Grid Conversion settings will be used.

RETURNS

- 0** OK
- 1** Error: import failed
- 2** Error: conversion failed

3.4.10 ComCimvalidate**Overview**

[Execute](#)
[GetClassType](#)
[GetDescriptionText](#)
[GetInputObject](#)
[GetModel](#)
[GetModelId](#)
[GetNumberOfValidationMessages](#)
[GetObject](#)

`GetObjectId`
`GetProfile`
`GetSeverity`
`GetType`

Execute

Executes the validation. Creates no validation report if no errors, or warnings were found.

```
int ComCimvalidate.Execute([int openReport])
```

ARGUMENTS

openReport (*optional*)

Option to open report after validation. If not provided, the report is opened.
Possible values are

- 0** Do not open report
- 1** Open report

RETURNS

- 0** OK
- 1** Error: validation failed

GetClassType

Returns the object class type from the selected validation message.

```
string ComCimvalidate.GetClassType(int messageIndex)
```

ARGUMENTS

messageIndex

Index of the validation message.

RETURNS

Object class type.

GetDescriptionText

Returns the description from the selected validation message.

```
string ComCimvalidate.GetDescriptionText(int messageIndex)
```

ARGUMENTS

messageIndex

Index of the validation message.

RETURNS

Message description.

GetInputObject

Returns the object selected for validation.

```
object ComCimvalidate.GetInputObject()
```

RETURNS

Pointer to *CimArchive*, *CimModel*, or *SetSelect*.

GetModel

Returns the *CimModel* object from the selected validation message.

```
object ComCimvalidate.GetModel(int messageIndex)
```

ARGUMENTS

messageIndex

Index of the validation message.

RETURNS

Pointer to *CimModel*.

GetModelId

Returns the model ID from the selected validation message.

```
string ComCimvalidate.GetModelId(int messageIndex)
```

ARGUMENTS

messageIndex

Index of the validation message.

RETURNS

Model ID.

GetNumberOfValidationMessages

Returns the number of validation messages generated.

```
int ComCimvalidate.GetNumberOfValidationMessages()
```

RETURNS

Number of validation messages.

GetObject

Returns the *CimObject* object from the selected validation message.

```
object ComCimvalidate.GetObject(int messageIndex)
```

ARGUMENTS

messageIndex

Index of the validation message.

RETURNS

Pointer to *CimObject*.

GetObjectId

Returns the object ID from the selected validation message.

```
string ComCimvalidate.GetObjectId(int messageIndex)
```

ARGUMENTS

messageIndex

Index of the validation message.

RETURNS

Object ID.

GetProfile

Returns the model profile from the selected validation message.

```
string ComCimvalidate.GetProfile(int messageIndex)
```

ARGUMENTS

messageIndex

Index of the validation message.

RETURNS

Model profile.

GetSeverity

Returns the severity of the selected validation message.

```
string ComCimvalidate.GetSeverity(int messageIndex)
```

ARGUMENTS

messageIndex

Index of the validation message.

RETURNS

Message severity.

GetType

Returns the type of the selected validation message.

```
string ComCimvalidate.GetType(int messageIndex)
```

ARGUMENTS

messageIndex

Index of the validation message.

RETURNS

Message type.

3.4.11 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 |

3.4.12 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(object 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.

```
void ComContingency.GetGeneratorEvent(int currentTimestep,
                                      int loadEvent,
                                      object& generator,
                                      double& changedP,
                                      double& changedQ )
```

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 ComContingency.GetInterruptedPowerAndCustomersForStage(double timeOfStageInMinutes,
    double& interruptedPower,
    double& newInterruptedPower,
    double& interruptedCustomers,
    double& newInterruptedCustomers)
```

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.

```
int ComContingency.GetInterruptedPowerAndCustomersForTimeStep(int currentTimeStep,
    double& interruptedPower,
    double& newInterruptedPower,
    double& interruptedCustomers,
    double& newInterruptedCustomers)
```

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.

```
void ComContingency.GetLoadEvent (int currentTimestep,
                                 int loadEvent,
                                 object& load,
                                 double& changedP,
                                 double& changedQ,
                                 int& isTransfer)
```

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 transferred

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.

```
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.

```
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.

```
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).

```
object ComContingency.GetObj (int index)
```

ARGUMENTS

index Element order index, 0 for the first object.

RETURNS

object Interupted element for given index.

NULL 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.

```
void ComContingency.GetSwitchEvent (int currentTimestep,
                                    int switchEvent,
                                    object& switchToBeActuated,
                                    int& isClosed,
                                    int& sectionalizingStep)
```

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.

```
int 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.

```
double ComContingency.GetTotalInterruptedPower ()
```

JumpToLastStep

Gets the last trace execution for this contingency.

```
int ComContingency.JumpToLastStep ([int timeDelay])
```

ARGUMENTS

timeDelay (optional)

time delay in seconds between trace steps

RETURNS

0 On success.

1 On error.

RemoveEvents

Removes events from this contingency.

```
void ComContingency.RemoveEvents ([double emitMessage])
void ComContingency.RemoveEvents (string whichEvents)
void ComContingency.RemoveEvents (double emitMessage,
                                 string whichEvents
                               )
void ComContingency.RemoveEvents (string whichEvents,
                                 double emitMessage
                               )
```

ARGUMENTS

emitMessage(optional)

0: no info message shall be issued after event removal

whichEvents(optional)

'lod' removed load events, '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.

3.4.13 ComCoordreport**Overview**

[DevicesToReport](#)
[HasResultsForDirectionalBackup](#)
[HasResultsForNonDirectionalBackup](#)
[HasResultsForOverreach](#)
[HasResultsForZone](#)
[MaxZoneNumberFor](#)
[ResultForDirectionalBackupVariable](#)
[ResultForNonDirectionalBackupVariable](#)
[ResultForOverreachVariable](#)
[ResultForZoneVariable](#)
[TopologyForDirectionalBackupVariable](#)
[TopologyForNonDirectionalBackupVariable](#)
[TopologyForOverreachVariable](#)

[TopologyForZoneVariable](#)
[TransferDirectionalBackupResultsTo](#)
[TransferNonDirectionalBackupResultsTo](#)
[TransferOverreachResultsTo](#)
[TransferResultsTo](#)
[TransferZoneResultsTo](#)

DevicesToReport

Returns the devices with stored results, i.e. which can be reported.

```
set ComCoordreport.DevicesToReport()
```

RETURNS

Container with reportable devices.

HasResultsForDirectionalBackup

Checks whether there is a stored directional backup result for the given device.

```
int ComCoordreport.HasResultsForDirectionalBackup(object device)
```

ARGUMENTS

device Device for which to check.

RETURNS

- 1** A directional backup result is stored for this device.
- 0** The device has no stored results or no result for the directional backup.

SEE ALSO

[ComCoordreport.HasResultsForZone\(\)](#), [ComCoordreport.HasResultsForOverreach\(\)](#), [ComCoordreport.HasResultsForNonDirectionalBackup\(\)](#)

HasResultsForNonDirectionalBackup

Checks whether there is a stored non-directional backup result for the given device.

```
int ComCoordreport.HasResultsForNonDirectionalBackup(object device)
```

ARGUMENTS

device Device for which to check.

RETURNS

- 1** A non-directional backup result is stored for this device.
- 0** The device has no stored results or no result for the non-directional backup.

SEE ALSO

[ComCoordreport.HasResultsForZone\(\)](#), [ComCoordreport.HasResultsForOverreach\(\)](#), [ComCoordreport.HasResultsForDirectionalBackup\(\)](#)

HasResultsForOverreach

Checks whether there is a stored overreach zone result for the given device.

```
int ComCoordreport.HasResultsForOverreach(object device)
```

ARGUMENTS

device Device for which to check.

RETURNS

- 1** An overreach zone result is stored for this device.
- 0** The device has no stored results or no result for the overreach zone.

SEE ALSO

[ComCoordreport.HasResultsForZone\(\)](#), [ComCoordreport.HasResultsForDirectionalBackup\(\)](#),
[ComCoordreport.HasResultsForNonDirectionalBackup\(\)](#)

HasResultsForZone

Checks whether there is a stored result for the given device and zone number.

```
int ComCoordreport.HasResultsForZone(object device,  
                                    int zoneNumber)
```

ARGUMENTS

device Device for which to check.

zoneNumber
Zone number to check (1-4).

RETURNS

- 1** A result is stored for this device and zone number.
- 0** The device has no stored results or no result for this zone number.

SEE ALSO

[ComCoordreport.HasResultsForOverreach\(\)](#), [ComCoordreport.HasResultsForDirectionalBackup\(\)](#),
[ComCoordreport.HasResultsForNonDirectionalBackup\(\)](#)

MaxZoneNumberFor

Returns the highest zone number in the stored results for the given device.

```
int ComCoordreport.MaxZoneNumberFor(object device)
```

ARGUMENTS

device Device for which to retrieve the zone number.

RETURNS

Highest zone number in the stored results.

ResultForDirectionalBackupVariable

Provides access to the directional backup result for a given device and variable.

```
int ComCoordreport.ResultForDirectionalBackupVariable(object device,
                                                    string variable,
                                                    double& result)
```

ARGUMENTS

device Device for which to retrieve the result.

variable Variable for which to retrieve the result:

dir Tripping direction

Tp Polygonal delay

Tc Circular delay

result (out)

Value of the stored result.

RETURNS

0 The result is valid.

1 The result was not calculated or not found.

2 The corresponding topological search failed.

3 The corresponding topological search ended prematurely (e.g. end of network reached).

SEE ALSO

[ComCoordreport.ResultForZoneVariable\(\)](#), [ComCoordreport.ResultForOverreachVariable\(\)](#),
[ComCoordreport.ResultForNonDirectionalBackupVariable\(\)](#)

ResultForNonDirectionalBackupVariable

Provides access to the non-directional backup result for a given device and variable.

```
int ComCoordreport.ResultForNonDirectionalBackupVariable(object device,
                                                       string variable,
                                                       double& result)
```

ARGUMENTS

device Device for which to retrieve the result.

variable Variable for which to retrieve the result:

dir Tripping direction

Tp Polygonal delay

Tc Circular delay

result (out)

Value of the stored result.

RETURNS

0 The result is valid.

1 The result was not calculated or not found.

2 The corresponding topological search failed.

3 The corresponding topological search ended prematurely (e.g. end of network reached).

SEE ALSO

[ComCoordreport.ResultForZoneVariable\(\)](#), [ComCoordreport.ResultForOverreachVariable\(\)](#),
[ComCoordreport.ResultForDirectionalBackupVariable\(\)](#)

ResultForOverreachVariable

Provides access to the overreach result for a given device and variable.

```
int ComCoordreport.ResultForOverreachVariable(object device,
                                              string variable,
                                              double& result)
```

ARGUMENTS

device Device for which to retrieve the result.

variable Variable for which to retrieve the result:

- X** Polygonal reactance
- R** Polygonal phase-phase resistance
- RE** Polygonal phase-earth resistance
- Z** Circular impedance
- phi** Circular angle
- dir** Tripping direction
- Tp** Polygonal delay
- Tc** Circular delay

result (out)

Value of the stored result.

RETURNS

- 0** The result is valid.
- 1** The result was not calculated or not found.
- 2** The corresponding topological search failed.
- 3** The corresponding topological search ended prematurely (e.g. end of network reached).

SEE ALSO

[ComCoordreport.ResultForZoneVariable\(\)](#), [ComCoordreport.ResultForDirectionalBackupVariable\(\)](#),
[ComCoordreport.ResultForNonDirectionalBackupVariable\(\)](#)

ResultForZoneVariable

Provides access to the result for a given device, zone number and variable.

```
int ComCoordreport.ResultForZoneVariable(object device,
                                         int zoneNumber
                                         string variable,
                                         double& result)
```

ARGUMENTS

device Device for which to retrieve the result.

zoneNumber

Zone number for which to retrieve the result (1-4).

variable Variable for which to retrieve the result:

X	Polygonal reactance
R	Polygonal phase-phase resistance
RE	Polygonal phase-earth resistance
Z	Circular impedance
phi	Circular angle
dir	Tripping direction
Tp	Polygonal delay
Tc	Circular delay

result (out)

Value of the stored result.

RETURNS

- 0** The result is valid.
- 1** The result was not calculated or not found.
- 2** The corresponding topological search failed.
- 3** The corresponding topological search ended prematurely (e.g. end of network reached).

SEE ALSO

[ComCoordreport.ResultForOverreachVariable\(\)](#), [ComCoordreport.ResultForDirectionalBackupVariable\(\)](#), [ComCoordreport.ResultForNonDirectionalBackupVariable\(\)](#)

TopologyForDirectionalBackupVariable

Returns the associated directional backup topology for a given device and variable.

```
set ComCoordreport.TopologyForDirectionalBackupVariable(object device,
                                                       string variable)
```

ARGUMENTS

device Device for which to retrieve the topology.

variable Variable for which to retrieve the topology:

Tp	Polygonal delay
Tc	Circular delay

RETURNS

Elements traversed by the topological search determining this variables result.

SEE ALSO

[ComCoordreport.TopologyForZoneVariable\(\)](#), [ComCoordreport.TopologyForOverreachVariable\(\)](#), [ComCoordreport.TopologyForNonDirectionalBackupVariable\(\)](#)

TopologyForNonDirectionalBackupVariable

Returns the associated non-directional backup topology for a given device and variable.

```
set ComCoordreport.TopologyForNonDirectionalBackupVariable(object device,
                                                       string variable)
```

ARGUMENTS

device Device for which to retrieve the topology.

variable Variable for which to retrieve the topology:

Tp	Polygonal delay
Tc	Circular delay

RETURNS

Elements traversed by the topological search determining this variables result.

SEE ALSO

[ComCoordreport.TopologyForZoneVariable\(\)](#), [ComCoordreport.TopologyForOverreachVariable\(\)](#),
[ComCoordreport.TopologyForDirectionalBackupVariable\(\)](#)

TopologyForOverreachVariable

Returns the associated overreach zone topology for a given device and variable.

```
set ComCoordreport.TopologyForOverreachVariable(object device,
                                               string variable)
```

ARGUMENTS

device Device for which to retrieve the topology.

variable Variable for which to retrieve the topology:

X	Polygonal reactance
R	Polygonal phase-phase resistance
RE	Polygonal phase-earth resistance
Z	Circular impedance
phi	Circular angle
Tp	Polygonal delay
Tc	Circular delay

RETURNS

Elements traversed by the topological search determining this variables result.

SEE ALSO

[ComCoordreport.TopologyForZoneVariable\(\)](#), [ComCoordreport.TopologyForDirectionalBackupVariable\(\)](#),
[ComCoordreport.TopologyForNonDirectionalBackupVariable\(\)](#)

TopologyForZoneVariable

Returns the associated topology for a given device, zone number and variable.

```
set ComCoordreport.TopologyForZoneVariable(object device,
                                           int zoneNumber
                                           string variable)
```

ARGUMENTS

device Device for which to retrieve the topology.

zoneNumber

Zone number for which to retrieve the topology (1-4).

variable Variable for which to retrieve the topology:

- X** Polygonal reactance
- R** Polygonal phase-phase resistance
- RE** Polygonal phase-earth resistance
- Z** Circular impedance
- phi** Circular angle
- Tp** Polygonal delay
- Tc** Circular delay

RETURNS

Elements traversed by the topological search determining this variables result.

SEE ALSO

[ComCoordreport.TopologyForOverreachVariable\(\)](#), [ComCoordreport.TopologyForDirectionalBackupVariable\(\)](#),
[ComCoordreport.TopologyForNonDirectionalBackupVariable\(\)](#)

TransferDirectionalBackupResultsTo

Transfers the results of the directional backup for the given variables to the device.

```
int ComCoordreport.TransferDirectionalBackupResultsTo(object device,
                                                    string variables)
```

ARGUMENTS

device Device to transfer the results to.

variable Variables to transfer as semi-colon separated string:

- dir** Tripping direction
- Tp** Polygonal delay
- Tc** Circular delay

RETURNS

0 Transfer did non succeed.

1 Result transfer successful.

SEE ALSO

[ComCoordreport.TransferZoneResultsTo\(\)](#), [ComCoordreport.TransferOverreachResultsTo\(\)](#),
[ComCoordreport.TransferNonDirectionalBackupResultsTo\(\)](#)

TransferNonDirectionalBackupResultsTo

Transfers the results of the non-directional backup for the given variables to the device.

```
int ComCoordreport.TransferNonDirectionalBackupResultsTo(object device,
                                                       string variables)
```

ARGUMENTS

- device* Device to transfer the results to.
- variable* Variables to transfer as semi-colon separated string:
- dir** Tripping direction
 - Tp** Polygonal delay
 - Tc** Circular delay

RETURNS

- 0** Transfer did non succeed.
- 1** Result transfer successful.

SEE ALSO

[ComCoordreport.TransferZoneResultsTo\(\)](#), [ComCoordreport.TransferOverreachResultsTo\(\)](#),
[ComCoordreport.TransferDirectionalBackupResultsTo\(\)](#)

TransferOverreachResultsTo

Transfers the results of the overreach zone for the given variables to the device.

```
int ComCoordreport.TransferOverreachResultsTo(object device,
                                             string variables)
```

ARGUMENTS

- device* Device to transfer the results to.
- variable* Variables to transfer as semi-colon separated string:
- X** Polygonal reactance
 - R** Polygonal phase-phase resistance
 - RE** Polygonal phase-earth resistance
 - Z** Circular impedance
 - phi** Circular angle
 - dir** Tripping direction
 - Tp** Polygonal delay
 - Tc** Circular delay

RETURNS

- 0** Transfer did non succeed.
- 1** Result transfer successful.

SEE ALSO

[ComCoordreport.TransferZoneResultsTo\(\)](#), [ComCoordreport.TransferDirectionalBackupResultsTo\(\)](#),
[ComCoordreport.TransferNonDirectionalBackupResultsTo\(\)](#)

TransferResultsTo

Transfers the results for the given variables to one or more devices.

```
int ComCoordreport.TransferResultsTo(object device,
                                    string variables)

int ComCoordreport.TransferResultsTo(set devices,
                                    string variables)
```

ARGUMENTS

devices Devices to transfer the results to.

variable Variables to transfer as semi-colon separated string:

X	Polygonal reactance
R	Polygonal phase-phase resistance
RE	Polygonal phase-earth resistance
Z	Circular impedance
phi	Circular angle
dir	Tripping direction
Tp	Polygonal delay
Tc	Circular delay

RETURNS

0 At least one transfer did not succeed.

1 Result transfer successful.

TransferZoneResultsTo

Transfers the results of a particular zone for the given variables to the device.

```
int ComCoordreport.TransferZoneResultsTo(object device,
                                         int zoneNumer,
                                         string variables)
```

ARGUMENTS

device Device to transfer the results to.

zoneNumber

Zone number for which to transfer the results (1-4).

variable Variables to transfer as semi-colon separated string:

X	Polygonal reactance
R	Polygonal phase-phase resistance
RE	Polygonal phase-earth resistance
Z	Circular impedance
phi	Circular angle
dir	Tripping direction
Tp	Polygonal delay
Tc	Circular delay

RETURNS

- 0** Transfer did not succeed.
- 1** Result transfer successful.

SEE ALSO

[ComCoordreport.TransferOverreachResultsTo\(\)](#), [ComCoordreport.TransferDirectionalBackupResultsTo\(\)](#),
[ComCoordreport.TransferNonDirectionalBackupResultsTo\(\)](#)

3.4.14 ComDiff

Overview

[Start](#)
[Stop](#)

Start

Starts comparisons of calculation results. See [SetDiffMode\(\)](#) for more information.

```
void ComDiff.Start()
```

SEE ALSO

[ComDiff.Stop\(\)](#), [GetDiffMode\(\)](#), [SetDiffMode\(\)](#)

Stop

Stops comparisons of calculation results. See [SetDiffMode\(\)](#) for more information.

```
void ComDiff.Stop()
```

SEE ALSO

[ComDiff.Start\(\)](#), [GetDiffMode\(\)](#), [SetDiffMode\(\)](#)

3.4.15 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.)

```
void ComDllmanager.Report()
```

3.4.16 ComDpl

Overview

CheckSyntax
 Encrypt
 Execute
 GetExternalObject*
 GetInputParameterDouble*
 GetInputParameterInt*
 GetInputParameterString*
 IsEncrypted*
 ResetThirdPartyModule
 SetExternalObject
 SetInputParameterDouble
 SetInputParameterInt
 SetInputParameterString
 SetResultString
 SetThirdPartyModule

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. An encrypted script can be executed without password but decrypted only with password. If no password is given a 'Choose Password' dialog appears.

```
int ComDpl.Encrypt ([string password = ''],  

                     [int removeObjectHistory = 1],  

                     [int masterCode = 0])
```

ARGUMENTS

password (optional)

 Password for decryption. If no password is given a 'Choose Password' dialog appears.

removeObjectHistory (optional)

 Handling of unencrypted object history in database, e.g. used by project versions or by undo:

- 0** Do not remove.
- 1** Do remove (default).
- 2** Show dialog and ask.

masterCode (optional)

Used for re-selling scripts. 3rd party licence codes already set in the script will be overwritten by this value (default = 0).

RETURNS

- 0** On success.
- 1** On error.

SEE ALSO

[ComDpl.IsEncrypted\(\)](#)

Execute

Executes the DPL script as subscript.

```
int ComDpl.Execute([input parameter])
```

ARGUMENTS

input parameter (optional)

All input parameter from the 'Basic Options' page of the 'Edit' dialog can be given as arguments. If a parameter is not given then the value from the dialog is used. The values from the dialog itself are not modified. These can be modified via

- [ComDpl.SetInputParameterInt\(\)](#)
- [ComDpl.SetInputParameterDouble\(\)](#)
- [ComDpl.SetInputParameterString\(\)](#).

The arguments are given by reference, thus a subscript can change the value of a variable from the main script.

RETURNS

For scripts without the use of [exit\(\)](#) the following values are returned:

- 0** On a successful execution.
- 1** An error occurred.
- 6** User hit the break button.

SEE ALSO

[ComDpl.CheckSyntax\(\)](#)

EXAMPLE

The following example performs a load-flow and calls the DPL subroutine "CheckVoltages" to check the voltage conditions.

```
int err;
err = Ldf.Execute();
if (.not.err) {
    err = CheckVoltages.Execute(0.94, 1.05);
}
if (err) {
    printf('Voltage conditions are violated');
}
```

GetExternalObject*

Gets the external object defined in the ComDpl edit dialog.

```
int ComDpl.GetExternalObject(string name,
                             object& value)
```

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.

This method is intended to get the database value of the input parameter of another script only and not to get the current value of an input parameter of another script executed as subscript as it can be done via 'subscript:input'.

```
int ComDpl.GetInputParameterDouble(string name,
                                   double& value)
```

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.

This method is intended to get the database value of the input parameter of another script only and not to get the current value of an input parameter of another script executed as subscript as it can be done via 'subscript:input'.

```
int ComDpl.GetInputParameterInt(string name,
                               int& value)
```

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.

This method is intended to get the database value of the input parameter of another script only and not to get the current value of an input parameter of another script executed as subscript as it can be done via 'subscript:input'.

```
int ComDpl.GetInputParameterString(string name,
                                    string& value)
```

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\(\)](#)

ResetThirdPartyModule

Resets the third party licence. Only possible for non-encrypted scripts. Requires masterkey licence for third party module currently set.

```
int ComDpl.ResetThirdPartyModule()
```

RETURNS

- 0** On success.
- 1** On error.

SetExternalObject

Sets the external object defined in the ComDpl edit dialog.

This method is intended to change the database value of the external object of another script only and not to change the current value of an external object of another script executed as subscript.

Changing the current value of an external object of another script executed as subscript is not possible. This use-case can only be fulfilled with input parameters of type object.

```
int ComDpl.SetExternalObject(string name,
                             object 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.

This method is intended to change the database value of the input parameter of another script only and not to change the current value of an input parameter of another script executed as subscript.

If you want to change the current value of an input parameter of another script executed as subscript then you can pass the input parameter as argument to [ComDpl.Execute\(\)](#) or set it directly e.g. 'subscript:input = ...'.

```
int ComDpl.SetInputParameterDouble(string 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.

This method is intended to change the database value of the input parameter of another script only and not to change the current value of an input parameter of another script executed as subscript.

If you want to change the current value of an input parameter of another script executed as subscript then you can pass the input parameter as argument to [ComDpl.Execute\(\)](#) or set it directly e.g. 'subscript:input = ...'.

```
int ComDpl.SetInputParameterInt(string 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.

This method is intended to change the database value of the input parameter of another script only and not to change the current value of an input parameter of another script executed as subscript.

If you want to change the current value of an input parameter of another script executed as subscript then you can pass the input parameter as argument to [ComDpl.Execute\(\)](#) or set it directly e.g. 'subscript:input = ...'.

```
int ComDpl.SetInputParameterString(string name,
                                    string 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\(\)](#)

SetResultString

Method for setting the 'sResultStr' calc comment parameter of a ComDpl object. This parameter is used by dpl driven textboxes.

```
void ComDpl.SetResultString(string result)
```

ARGUMENTS

result String to set to 'sResultStr' calc comment parameter.

SetThirdPartyModule

Sets the third party licence to a specific value. Only possible for non-encrypted scripts with no third party licence set so far. Requires masterkey licence for third party module to be set.

```
int ComDpl.SetThirdPartyModule(string companyCode,
                               string moduleCode
                               )
```

ARGUMENTS

companyCode

D isplay name or numeric value of company code.

moduleCode

D isplay name or numeric value of third party module.

RETURNS

- 0** On success.
- 1** On error.

3.4.17 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

3.4.18 ComGenrelinC

Overview

[GetCurrentIteration](#)
[GetMaxNumIterations](#)

GetCurrentIteration

The command returns the current iteration number of the 'Run Generation Adequacy' command (ComGenrelinC).

```
int ComGenrelinC.GetCurrentIteration()
```

RETURNS

Returns the current iteration number.

EXAMPLE

```
object cominc, comexe;
int maxiter, curriter;

cominc = GetFromStudyCase('ComGenrelinC');
comexe = GetFromStudyCase('ComGenrel');
cominc.Execute();
curriter = cominc.GetCurrentIteration();
printf('Current Iter = %d', curriter);

comexe:maxit = 5000;
comexe.Execute();
maxiter = cominc.GetMaxNumIterations();
curriter = cominc.GetCurrentIteration();
printf('Max Iter = %d, Current Iter = %d', maxiter, curriter);

comexe:addit = 1000;
comexe.Execute();
maxiter = cominc.GetMaxNumIterations();
curriter = cominc.GetCurrentIteration();
printf('Max Iter = %d, Current Iter = %d', maxiter, curriter);
```

GetMaxNumIterations

The command returns the maximum number of iterations specified in the 'Run Generation Adequacy' command (ComGenrel).

```
int ComGenrelinc.GetMaxNumIterations()
```

RETURNS

Returns the maximum number of iterations.

EXAMPLE

```
object cominc, comexe;
int maxiter, curriter;

cominc = GetFromStudyCase('ComGenrelinc');
comexe = GetFromStudyCase('ComGenrel');
cominc.Execute();
curriter = cominc.GetCurrentIteration();
printf('Current Iter = %d', curriter);

comexe:maxit = 5000;
comexe.Execute();
maxiter = cominc.GetMaxNumIterations();
curriter = cominc.GetCurrentIteration();
printf('Max Iter = %d, Current Iter = %d', maxiter, curriter);

comexe:addit = 1000;
comexe.Execute();
maxiter = cominc.GetMaxNumIterations();
curriter = cominc.GetCurrentIteration();
printf('Max Iter = %d, Current Iter = %d', maxiter, curriter);
```

3.4.19 ComGridtocim

Overview

[ConvertAndExport](#)
[SetAuthorityUri](#)
[SetBoundaries](#)
[SetGridsToExport](#)

ConvertAndExport

Convert Grid to CIM and export CIM data to given file path without storing into database. If no file path is provided, the file path from the corresponding CIM Data Export command in the study will be used. In case of a validation error the export is not performed.

```
int ComGridtocim.ConvertAndExport([int validate])
int ComGridtocim.ConvertAndExport(string filePath,
                                  [int validate])
```

ARGUMENTS

filePath File path for CIM data.

validate (optional)

Option to execute CIM Data Validation before export. If not provided, the validation

is executed. Possible values are

- 0** Do not validate
- 1** Validate

RETURNS

- 0** OK
- 1** Error: conversion failed
- 2** Error: export failed

SetAuthorityUri

Sets the authority uri for a specific grid.

```
void ComGridtocim.SetAuthorityUri(object grid,
                                string uri)
```

ARGUMENTS

- grid* Grid to set the URI for.
- uri* Model authority URI to be set.

SetBoundaries

Sets the grids as "Boundary Grid" and clears any previous setting.

```
void ComGridtocim.SetBoundaries(set grids)
```

ARGUMENTS

- grids* The grids to be considered as boundaries.

SetGridsToExport

Sets the grids as "Selected" and clears any previous setting.

```
void ComGridtocim.SetGridsToExport(set grids)
```

ARGUMENTS

- grids* The grids to be selected.

3.4.20 ComHostcap

Overview

[CalcMaxHostedPower](#)

CalcMaxHostedPower

The function executes predefined hosting capacity analysis command and returns the max. hosted power (P, Q) at the given terminal. In addition, object where the violation has occurred is returned.

```
int ComHostcap.CalcMaxHostedPower(object terminal,
                                    double& P,
                                    double& Q,
                                    object& violatedElement)
```

ARGUMENTS

terminal Hosting site.

P (out) Max. active power.

Q (out) Max. reactive power.

violatedElement (out)
Element where the limiting violation occurs.

RETURNS

- 1** Selected object is not a terminal.
- 0** No violations. Calculation OK.
- 1** Calculation failed.
- 2** Thermal violation.
- 3** Voltage violation.
- 4** Protection violation.
- 5** Power quality violation.

EXAMPLE

```
external object terminalA;
object cmdHCA, violatedElm;
double dP, dQ;
int absRet;
int iret;
cmdHCA = GetFromStudyCase('ComHostcap');
if (cmdHCA) {
    iret = cmdHCA.CalcMaxHostedPower(terminalA, dP, dQ, violatedElm);
    ! If the calculation was success then the data can be obtained
    absRet = abs(iret);
    if (absRet <> 1) {
        printf('Max. hosted power P, Q is: %.3f MW, %.3f MVAr', dP, dQ);
        if (iret .and. violatedElm) {
            printf('The most critical element: %o', violatedElm);
            printf('Violation type: %d', iret);
        }
    }
}
```

3.4.21 ComImport

Overview

[GetCreatedObjects](#)
[GetModifiedObjects](#)

GetCreatedObjects

Returns the newly created objects after execution of a DGS import.

```
set ComImport.GetCreatedObjects()
```

RETURNS

Collection of objects that have been created during DGS import.

EXAMPLE

The following example returns the created objects after execution of a DGS import:

```
set created;
object obj;

ImportCmd.Execute(); !execute dgs import

printf('Created objects:');
created = ImportCmd.GetCreatedObjects();
for(obj = created.First(); obj; obj = created.Next()) {
    printf('%o', obj);
}
```

GetModifiedObjects

Returns the modified objects after execution of a DGS import.

```
set ComImport.GetModifiedObjects()
```

RETURNS

Collection of objects that have been modified during DGS import.

EXAMPLE

The following example returns the modified objects after execution of a DGS import:

```
set modified;
object obj;

ImportCmd.Execute(); !execute dgs import

printf('\nModified objects:');
modified = ImportCmd.GetModifiedObjects();
for(obj = modified.First(); obj; obj = modified.Next()) {
    printf('%o', obj);
}
```

3.4.22 ComInc

Overview

[ZeroDerivative*](#)

ZeroDerivative*

This function returns 1 if the state variable derivatives are less than the tolerance for the initial conditions, provided that the *Simulation method* is set to *RMS values* and the *Verify initial conditions* option is selected in the *Calculation of initial conditions* command. The tolerance is defined on the *Solver options* page in the *Maximum error for dynamic model equations* field. The function returns 0 if the aforementioned conditions are not met, or if at least one state variable has a derivative larger than the tolerance.

```
int ComInc.ZeroDerivative()
```

RETURNS

- 0** At least one state variable has a derivative larger than the tolerance, or the required command options have not been set.
- 1** All state variable derivatives are less than the tolerance.

EXAMPLE

The following example checks whether the initial conditions for the RMS simulation were calculated and if the command option Verify Initial Conditions has been set.

```
object oInc;
int result;
oInc = GetCaseObject('ComInc');

oInc.Execute();

result = IsRmsValid();
if (result = 1) {
    if (oInc:iopt_show = 1) {
        printf('Zero derivative = %d \n', oInc.ZeroDerivative());
    }
}
```

3.4.23 ComLdf

Overview

[CheckControllers*](#)
[DoNotResetCalc](#)
[EstimateOutage](#)
[Execute](#)
[IsAC*](#)
[IsBalanced*](#)
[IsDC*](#)
[PrintCheckResults](#)
[SetOldDistributeLoadMode](#)

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.

EstimateOutage

Estimate the loading of all branches with outages of given set of branch elements.

```
int ComLdf.EstimateOutage(set 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.

```
void ComLdf.SetOldDistributeLoadMode(int iOldMode)
```

ARGUMENTS

iOldMode The flag showing if the old model is used.

- 0** Use standard mode.
- 1** Use old mode.

3.4.24 ComLink

Overview

[GetMicroSCADASatus](#)
[LoadMicroSCADAFile](#)
[ReceiveData](#)
[SendData](#)
[SentDataStatus](#)
[SetMicroSCADASatus](#)
[SetOPCReceiveQuality](#)
[SetSwitchShcEventMode](#)

GetMicroSCADASatus

Returns the current status of the link when used in MicroSCADA mode.

```
int ComLink.GetMicroSCADASatus ()
```

RETURNS

Returns one of the following values

- 1** undefined
- 0** dispatcher ldf
- 1** online state estimation
- 2** simulation

LoadMicroSCADAFile

Reads in a MicroSCADA snapshot file.

```
int ComLink.LoadMicroSCADAFile(string 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

EXAMPLE

```
object measure;
set measureSet;

!Set temp status for all measurement objects
measureSet = GetCalcRelevantObjects(\textquotesignle *.StaExt*\textquotesignle );
measure = measureSet.First();
while(measure) {
    measure.InitTmp();
    measure = measureSet.Next();
}

!initialization by forced sending all values
Link.SendData(1);
```

SentDataStatus

Outputs status of all items marked for sending to output window.

```
int ComLink.SentDataStatus()
```

RETURNS

Number of items configured for sending.

SetMicroSCADASatus

Sets the current status of the link when used in MicroSCADA mode.

```
void ComLink.SetMicroSCADASatus(double status)
```

ARGUMENTS

<i>status</i>)	-1	undefined
	0	dispatcher ldf
	1	online state estimation
	2	simulation

SetOPCReceiveQuality

Allows to override the actual OPC receive quality by this value. (Can be used for testing.)

```
int ComLink.SetOPCReceiveQuality(int quality)
```

ARGUMENTS

<i>quality</i>	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 or whether shc switch events shall be created instead.

```
void ComLink.SetSwitchShcEventMode(int enabled)
```

ARGUMENTS

<i>enabled</i>

0	Values are directly written to switches
1	For each value change a switch event will be created

3.4.25 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** Cancelled 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.

```
void ComMerge.ExecuteWithActiveProject()
```

GetCorrespondingObject

Searches corresponding object for given object.

```
object ComMerge.GetCorrespondingObject(object 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.

NULL Corresponding object not found.

GetModification

Gets kind of modification between corresponding objects of “Base” and “1st” or “2nd”.

```
int ComMerge.GetModification(object sourceObj,
                             [int target]
                             )
```

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(object 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.

```
set ComMerge.GetModifiedObjects(int modType,
                               [int modSource]
                               )
```

ARGUMENTS

modType

- 1** get unmodified objects
- 2** get modified objects
- 3** get added objects
- 4** get removed objects

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.

```
void 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.

```
void 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(set objs)
int ComMerge.PrintModifications(object obj)
```

ARGUMENTS

- | | |
|-------------|---|
| <i>objs</i> | Set of objects for which the modifications are printed. |
| <i>obj</i> | 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.

```
void ComMerge.Reset()
```

SetAutoAssignmentForAll

Sets the assignment of all compared objects automatically.

```
void 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.

```
void ComMerge.SetObjectsToCompare(object base,
                                [object first,
                                [object 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.

3.4.26 ComMot

Overview

[GetMotorConnections*](#)
[GetMotorSwitch*](#)
[GetMotorTerminal*](#)

GetMotorConnections*

Finds the cables connecting the motor to the switch.

```
set ComMot.GetMotorConnections(object 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.

```
object ComMot.GetMotorSwitch(object motor)
```

ARGUMENTS

motor The motor element

RETURNS

Returns the switch element.

GetMotorTerminal*

Finds the terminal to which the motor will be connected.

```
object ComMot.GetMotorTerminal(object motor)
```

ARGUMENTS

motor The motor element

RETURNS

Returns the terminal element.

3.4.27 ComNmink

Overview

[AddRef](#)
[Clear](#)
[GenerateContingenciesForAnalysis](#)
[GetAll*](#)

AddRef

Adds shortcuts to the objects to the existing selection.

```
void ComNmink.AddRef(object O)
void ComNmink.AddRef(set S)
```

ARGUMENTS

O(optional)
an object

S(optional)
a Set of objects

EXAMPLE

The following prepares and executes an outage simulation for all high loaded lines.

```
PrepOut.Clear();
S = GetCalcRelevantObjects();
O = S.Firstmatch('ElmLne');
while (O) {
    if (O:c:loading > 75) {
        PrepOut.AddRef(O);
    }
    O = S.Nextmatch();
}
PrepOut.Execute();
```

Clear

Delete all contents, i.e. to empty the selection.

```
void ComNmink.Clear()
```

EXAMPLE

The following example creates a selection of all loads.

```
PrepOut.Clear();
objects = GetCalcRelevantObjects();
obj = objects.Firstmatch('ElmLne');
while (obj) {
    if (obj:c:loading > 75) {
        PrepOut.AddRef(obj);
    }
    obj = objects.Nextmatch();
}
PrepOut.Execute();
```

GenerateContingenciesForAnalysis

Generates Contingencies for Contingency Analysis. Similar to calling 'Execute' but does not pop-up the contingency command.

RETURNS

- 0** On success.
- 1** On error.

GetAll*

Returns all objects which are of the class 'ClassName'.

```
set ComNmink.GetAll(string className)
```

ARGUMENTS

className

The object class name.

RETURNS

The set of objects

EXAMPLE

The following example writes all three winding transformers in the preparation command to the output window.

```
set list;
object obj;
S = Prep.GetAll('ElmTr3');
obj = list.First();
while (obj) {
    obj.ShowFullName();
    obj = list.Next();
}
```

3.4.28 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.

```
set ComOmr.GetFeeders()
```

RETURNS

The set of all feeders used for optimisation.

EXAMPLE

The following example calculates a pre-configured Optimal Manual Restoration command and shows the considered feeders in the output window.

```

int errCode;
object comOmr,
      feeder;
set feeders;

comOmr = GetFromStudyCase('ComOmr');

errCode = comOmr.Execute();
if (errCode=0) {
    feeders = comOmr.GetFeeders();
    printf('The following feeders participated in the optimisation:\n');
    for (feeder = feeders.First(); feeder; feeder = feeders.Next()) {
        printf(' - %o\n', feeder);
    }
}

```

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.

```

set ComOmr.GetOMR(object 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.

EXAMPLE

The following example calculates a pre-configured Optimal Manual Restoration command and shows the resulting optimal manual restoration points (together with its connected switches) in the output window.

```

int errCode,
    numRegions,
    isClass,
    regIdx;
object comOmr,
      feeder,
      obj;
set feeders,
    termAndSwitches;

comOmr = GetFromStudyCase('ComOmr');

errCode = comOmr.Execute();

```

```

if ( errCode=0 ) {
    feeders = comOmr.GetFeeders();
    for ( feeder=feeders.First(); feeder; feeder=feeders.Next() ) {
        numRegions = comOmr.GetRegionCount(feeder);
        for ( regIdx=0; regIdx<numRegions; regIdx=regIdx+1 ) {
            termAndSwitches = comOmr.GetOMR(feeder, regIdx);
            for ( obj=termAndSwitches.First(); obj; obj=termAndSwitches.Next() ) {
                isClass = obj.IsClass('ElmTerm');
                if (isClass) {
                    printf('The optimal terminal in region %d is: %o', regIdx, obj);
                }
                isClass = obj.IsClass('ElmCoup');
                if (isClass) {
                    printf('Optimal manual switches in region %d are: %o', regIdx, obj);
                }
                isClass = obj.IsClass('StaSwitch');
                if (isClass) {
                    printf('Optimal manual switches in region %d are: %o', regIdx, obj);
                }
            }
        }
    }
}

```

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(object feeder)
```

ARGUMENTS

feeder Feeder to derive number of regions(pockets) for.

RETURNS

Number of regions(pockets) for the feeder.

EXAMPLE

The following example calculates a pre-configured Optimal Manual Restoration command and shows the number of regions(pockets) for the considered feeders in the output window.

```

int errCode,
    numRegions;
object comOmr,
        feeder;
set feeders;

comOmr = GetFromStudyCase('ComOmr');

errCode = comOmr.Execute();
if ( errCode=0 ) {
    feeders = comOmr.GetFeeders();
    for ( feeder = feeders.First(); feeder ; feeder = feeders.Next() ) {
        numRegions = comOmr.GetRegionCount(feeder);

```

```

        printf(' %o: number of regions: %d\n', feeder, numRegions);
    }
}

```

3.4.29 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 occurred -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 occurred -2 if the link is not connected

3.4.30 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(double 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.

```
object ComOutage.GetObject(int line)
```

ARGUMENTS

line line index, if index exceeds the range NULL is returned

RETURNS

the element of line “line” in the table.

EXAMPLE

The following example shows how to access elements in the table of all ComOutage whose names start with “L”.

```

object command,
outage,
obj;
set outages;
int sizeElements,
line;

command = GetFromStudyCase('*.*.ComSimoutage');
! get rel. command from study case
if (command) {
    ! get all outages of which the names starts with "L"
    outages = command.GetContents('L*.ComOutage');
    ! show the elements of all outages
    for (outage=outages.First(); outage; outage=outages.Next()) {
        outage.GetSize('Elements', sizeElements);
        ! get size of vector Elements
        obj = outage.GetObject(line);
        !outage.GetVal(obj, 'Elements', line); same like GetObject
        obj.ShowFullName();
    }
}

```

RemoveEvents

Remove all events defined in this contingency.

```

void ComOutage.RemoveEvents ([int info])
void ComOutage.RemoveEvents (string type)
void ComOutage.RemoveEvents (int info, string type)
void ComOutage.RemoveEvents (string 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

EXAMPLE

The following example shows how to remove events from ComOutage.

```

object command,
outage;
set outages;

command = GetFromStudyCase('*.*.ComSimoutage');
! get contingency analysis command from study case
if (command) {
    ! get all outages of which the names starts with "L"
    outages = command.GetContents('L*.ComOutage');
    ! remove the events
}

```

```

for (outage=outages.First(); outage; outage=outages.Next()) {
    outage.RemoveEvents(0);! delete all stored events,suppress info message
}
}

```

SetObjs

To fill up the "interrupted components" with given elements.

Sets the list of objects according to S.

```
int ComOutage.SetObjs(set 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.
≠ 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.
≠ 0	On error.

3.4.31 ComPfdimport

Overview

[GetImportedObjects](#)

GetImportedObjects

Returns the imported objects of last execution of command.

```
set ComPfdimport.GetImportedObjects()
```

RETURNS

Returns the imported objects of the last execution of the command.

EXAMPLE

```
object com;
object importedObject;
set importedObjects;

com = GetFromStudyCase('ComPfdimport');

SetConsistencyCheck(0);
com:g_target = GetCurrentUser();
com:g_file = 'D:/test.pdf';
SetConsistencyCheck(1);

com.Execute();
importedObjects = com.GetImportedObjects();
printf('All imported objects:');
for (importedObject = importedObjects.First();
     importedObject;
     importedObject = importedObjects.Next())
{
    printf('%o', importedObject);
}
```

3.4.32 ComPrjconnector

Overview

[GetSuccessfullyConnectedItems](#)
[GetUnsuccessfullyConnectedItems](#)

GetSuccessfullyConnectedItems

Returns a list of elements which were correctly processed in the last run of the command

```
set ComPrjconnector.GetSuccessfullyConnectedItems()
```

GetUnsuccessfullyConnectedItems

Returns a list of elements which were not correctly processed in the last run of the command

```
set ComPrjconnector.GetUnsuccessfullyConnectedItems()
```

3.4.33 ComProtgraphic

Overview

AddToUpdatePages
ClearUpdatePages

AddToUpdatePages

Adds pages (*.SetVipage) to the user-defined selection of plot pages to update.

```
void ComProtgraphic.AddToUpdatePages(set pages)
```

ClearUpdatePages

Clears the user-defined selection of plot pages to update.

```
void ComProtgraphic.ClearUpdatePages()
```

3.4.34 ComPvcycles

Overview

FindCriticalBus

FindCriticalBus

Returns the critical bus for a given PV-Curve result file. The critical bus is the one with the highest gradient at the last iteration. If a bus is found, whose voltage drop is twice as high, as the one with the highest gradient, the bus with the higher voltage drop is the critical one.

```
object ComPvcycles.FindCriticalBus(object resultfile)
```

3.4.35 ComPython

Overview

GetExternalObject*
GetInputParameterDouble*
GetInputParameterInt*
GetInputParameterString*
SetExternalObject
SetInputParameterDouble
SetInputParameterInt
SetInputParameterString

GetExternalObject*

Gets the external object defined in the ComPython edit dialog.

```
int ComPython.GetExternalObject(string name,  
                                object& value)
```

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 ComPython.GetInputParameterDouble(string name,  
                                      double& value)
```

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 ComPython.GetInputParameterInt(string name,  
                                   int& value)
```

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 ComPython.GetInputParameterString(string name,  
                                     string& value)
```

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(string name,  
                               object 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(string 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(string 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(string name,  
                                      string 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\(\)](#)

3.4.36 ComRed

Overview

[ReductionInMemory](#)
[ResetReductionInMemory](#)

ReductionInMemory

Execute network reduction in memory, no change to database. Note: afterwards, the function `ResetReductionInMemory()` must be called to revert the reduction.

```
int ComRed.ReductionInMemory()
```

RETURNS

Returns 0 if successfully executed.

EXAMPLE

```
object redCom;

redCom = GetCaseObject('ComRed');

if (redCom = NULL) {
    Error('No Network Reduction Command found in active study case!');
    exit();
}

redCom.ReductionInMemory();
```

ResetReductionInMemory

Restore the network back to original after reduction in memory.

```
void ComRed.ResetReductionInMemory()
```

EXAMPLE

```
object redCom;

redCom = GetCaseObject('ComRed');

if (redCom = NULL) {
    Error('No Network Reduction Command found in active study case!');
    exit();
}
```

```
redCom.ResetReductionInMemory();
```

3.4.37 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.
- ≠ 0 On error.

EXAMPLE

The following example shows how to call the evaluation of the results.

```
object command,
resultFile;
int err;

command = GetFromStudyCase('.ComRel3');
! get the command from study case
if (command) {
    err=command.AnalyseElmRes(0);
    ! hide error message
    if (err) {
        ! display my own error message
        resultFile = command:p_resenum;
        if (resultFile) {
            Error('Evaluation of results %s failed', resultFile:loc_name);
        }
    }
}
```

ExeEvt

Executes a given event.

```
void ComRel3.ExeEvt ([object event])
```

ARGUMENTS

event The event that shall be executed.

OvlAlleviate

Performs an overload alleviation for given events.

```
int ComRel3.OvlAlleviate ([set 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.

```
void ComRel3.RemoveEvents ()
```

EXAMPLE

The following example shows how to remove events from all contingencies stored inside the reliability command:

```
object command;

command = GetFromStudyCase (*.ComRel3);
! get reliability command from study case
if (command) {
    command.RemoveEvents ();
}
```

RemoveOutages

Removes all contingency definitions (*.ComContingencies) stored inside the reliability command.

```
void ComRel3.RemoveOutages ([int msg])
```

ARGUMENTS

msg(optional)

- 1** Show info message in output window (default value).
- 0** Do not emit messages.

EXAMPLE

The following example removes all ComContingencies objects stored inside the reliability command in the study case.

```
object command;

command = GetFromStudyCase('*.*.ComRel3');
! get the command from study case
if (command) {
    command.RemoveOutages(0);
    ! suppress info message
}
```

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(object 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.

3.4.38 ComRepost**Overview**

[CalcContributions](#)
[GetContributionOfComponent](#)

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 ComRepost.CalcContributions([set 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.

EXAMPLE

The following example shows how to calculate the contributions for all elements.

```
object command;
int err;

command = GetFromStudyCase (*.ComRelpost );
if (command) {
    err=command.CalcContributions ();
}
```

GetContributionOfComponent

Gets the contributions of a component to a certain reliability indice.

```
double ComRelpost.GetContributionOfComponent (int componentNr,
                                              string indice)
```

ARGUMENTS

componentNr 1. Lines

- 2. Cables
- 3. Transformers
- 4. Busbars
- 5. Generators
- 6. Common Modes
- 7. Double Earth Faults

indice Available indices are: 'SAIFI', 'SAIDI', 'ASIFI', 'ASIDI', 'ENS', 'EIC'

RETURNS

The contribution of this component to this reliability indice.

EXAMPLE

See 'Reliability Contributions Report' . The link to this report can be found in the contribution to reliability indices command.

3.4.39 ComRelreport

Overview

[GetContingencies*](#)
[GetContributionOfComponent](#)

GetContingencies*

Gets all contingencies of reliability for reporting.

```
set ComRelpost.GetContingencies()
```

RETURNS

All contingencies of reliability for reporting.

GetContributionOfComponent

Is described in [ComRelpost.GetContributionOfComponent\(\)](#).

```
double ComRelreport.GetContributionOfComponent(int componentNr,  
                                              string indice)
```

3.4.40 ComRes

Overview

[ExportFullRange*](#)
[FileNmResNm](#)

ExportFullRange*

Executes the export command for the whole data range.

```
void ComRes.ExportFullRange()
```

EXAMPLE

The following example exports the complete range of results

```
object obj,  
      export;  
set range;  
export = GetFromStudyCase('*.*.ComRes');  
range = SEL.GetAll('ElmRes');  
obj = range.First();  
while (obj) {  
    export:pResult = obj;  
    export.ExportFullRange();  
    obj = range.Next();  
}
```

FileNmResNm

Sets the filename for the data export to the name of the result object being exported (classes: ElmRes, IntComtrade)

```
void ComRes.FileNmResNm()
```

EXAMPLE

The following example searches the export command in the study case, assigns the first ElmRes found and sets the filename of the export command to the name of the result file object.

```
object export,
      res;
export = GetFromStudyCase('*.ComRes');
if (export=nullptr) {
    exit(1);
}

res = GetFromStudyCase('*.ElmRes');
if (res=nullptr) {
    exit(1);
}
export:pResult = res;
export.FileNmResNm();
```

3.4.41 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(double Zr,
                           double 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 ComShc.GetOverLoadedBranches(double ip,
                                 double ith,
                                 [set& branches]
)
```

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.
- ≠ 0 Number of branches.

EXAMPLE

```
object oShcCmd, oBranch;
double dIupload, dIthload;
set sBranch;
int iret;

dIupload = 500;
dIthload = 100;
oShcCmd = GetFromStudyCase('ComShc');
iret = oShcCmd.GetOverLoadedBranches(dIupload, dIthload, sBranch); ! Shall check all branches
if (iret > 0) {
    oBranch = sBranch.First();
```

```

while (oBranch) {
    printf('Overloaded branch: %o \n', oBranch);
    oBranch = sBranch.Next();
}
}

```

GetOverLoadedBuses

Get overloaded buses after a short-circuit calculation.

```

int ComShc.GetOverLoadedBuses(double ip,
                             double ith,
                             [set& buses])

```

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.

≠ 0 Number of buses.

EXAMPLE

```

object oShcCmd, oBus;
double dIupload, dIthload;
set sBus;
int iret;

dIupload = 500;
dIthload = 100;
oShcCmd = GetFromStudyCase('ComShc');
iret = oShcCmd.GetOverLoadedBuses(dIupload, dIthload, sBus); ! Shall check all buses
if (iret > 0) {
    oBus = sBus.First();
    while (oBus) {
        printf('Overloaded bus: %o \n', oBus);
        oBus = sBus.Next();
    }
}

```

3.4.42 ComShctrace

Overview

[BlockSwitch](#)
[ExecuteAllSteps](#)
[ExecuteInitialStep](#)
[ExecuteNextStep](#)
[GetBlockedSwitches](#)
[GetCurrentTimeStep](#)
[GetDeviceSwitches](#)
[GetDeviceTime](#)
[GetNonStartedDevices](#)
[GetStartedDevices](#)
[GetSwitchTime](#)
[GetTrippedDevices](#)
[NextStepAvailable](#)

BlockSwitch

Blocks a switch from operating for the remainder of the trace.

```
int ComShctrace.BlockSwitch(object 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 occurred, 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 occurred, 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 occurred, step was executed .
- !=0** An error occurred, calculation was reset.

SEE ALSO

[ComShctrce.ExecuteInitialStep\(\)](#)

GetBlockedSwitches

Returns all switches which are currently blocked.

```
set ComShctrace.GetBlockedSwitches()
```

RETURNS

All blocked switches.

GetCurrentTimeStep

Returns the current time step of the trace in seconds.

```
int ComShctrace.GetCurrentTimeStep()
```

RETURNS

The current time step in [s].

GetDeviceSwitches

Returns all switches operated by a protection device.

```
set ComShctrace.GetDeviceSwitches(object 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.

```
int ComShctrace.GetDeviceTime(object 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.

```
set ComShctrace.GetNonStartedDevices()
```

RETURNS

All protection devices which are not started.

GetStartedDevices

Returns all started but not yet tripped protection devices.

```
set ComShctrace.GetStartedDevices()
```

RETURNS

All started but not yet tripped protection devices.

GetSwitchTime

Returns the time a switch device operated or will operate at.

```
int ComShctrace.GetSwitchTime(object device,
                               object 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.

```
set 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.

3.4.43 ComSim

Overview

[GetSimulationTime](#)
[GetTotalWarnA](#)
[GetTotalWarnB](#)
[GetTotalWarnC](#)
[GetViolatedScanModules*](#)
[LoadSnapshot](#)
[SaveSnapshot](#)

GetSimulationTime

Get the actual simulation time if the initial conditions are calculated.

```
int ComSim.GetSimulationTime()
```

RETURNS

Returns the simulation time in seconds. If the initial conditions are not calculated, then the function returns 'nan'.

GetTotalWarnA

Returns the total number of type-A (serious) warnings related to the time-domain simulation.

```
int ComSim.GetTotalWarnA()
```

RETURNS

Returns the total number of type-A (serious) warnings.

GetTotalWarnB

Returns the total number of type-B (moderate) warnings related to the time-domain simulation.

```
int ComSim.GetTotalWarnB()
```

RETURNS

Returns the total number of type-B (moderate) warnings.

GetTotalWarnC

Returns the total number of type-C (minor) warnings related to the time-domain simulation.

```
int ComSim.GetTotalWarnC()
```

RETURNS

Returns the total number of type-C (minor) warnings.

GetViolatedScanModules*

Returns a set of scan modules of which each have at least one violation.

```
set ComSim.GetViolatedScanModules()
```

RETURNS

Returns a set of scan modules of which each have at least one violation.

EXAMPLE

See example [ScnFreq.GetViolationTime\(\)](#).

LoadSnapshot

Load the state of a dynamic simulation from a file.

```
int ComSim.LoadSnapshot([string snapshotFilePath], [int suppressUserMessage])
```

DEPRECATED NAMES

LoadSimulationState

ARGUMENTS

snapshotFilePath (*optional*)

The snapshot file to load. If no file is specified, the last snapshot stored in the memory is used. (default = "")

suppressUserMessage (*optional*)

The pop-up window asking for data overwrite is not displayed if this value is set to 1. (default = 0)

RETURNS

- 0** Saved simulation state has been loaded.
- 1** Saved simulation state cannot be loaded.
- 2** Saved simulation state does not match actual model. Calculation will be reset.

SaveSnapshot

Save the state of a dynamic simulation to memory and to a file.

```
int ComSim.SaveSnapshot(string snapshotFilePath, [int suppressUserMessage])
```

DEPRECATED NAMES

SaveSimulationState

ARGUMENTS

snapshotFilePath (*optional*)

The snapshot file to save. If no file is specified, the last snapshot is saved in the non-persistent memory slot. (default = "")

suppressUserMessage (*optional*)

The pop-up window asking for file overwriting is not displayed if this value is set to 1. (default = 0)

RETURNS

0 OK

1 Error

3.4.44 ComSimoutage

Overview

AddCntcy
 AddContingencies
 AddRas
 ClearCont
 CreateFaultCase
 Execute
 ExecuteAndCheck
 GetNTopLoadedElms*
 MarkRegions
 RemoveAllRas
 RemoveContingencies
 RemoveRas
 Reset
 SetLimits
 Update

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(object o,
                           [string name]
                           )
```

ARGUMENTS

O The ComOutage object

name A name for the outage

RETURNS

0 On success.

1 On error.

EXAMPLE

The following example analyses all selected outage definitions and adds the results to the intermediate results.

```
set outages;
object limit;
outages = SEL.GetAll('ComOutage');
limit = outages.First();
while (limit) {
    CA.AddCntcy(obj);
    limit = outages.Next();
}
```

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.

```
void ComSimoutage.AddContingencies()
```

EXAMPLE

```
object command;

command.AddContingencies();
```

AddRas

Adds a reference to a given IntRas to the ComSimoutage.

```
int ComSimoutage.AddRas(object ras)
```

ARGUMENTS

ras The IntRas object

RETURNS

- 0** If the reference to the IntRas has been added.
- 1** If the reference to the IntRas has already been there or on error.

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(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 elements 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.

ExecuteAndCheck

Executes contingency analysis and checks the violated constraints. The constraints are defined in models, such as maximum and minimum voltage of buses, maximum loading of lines etc. On the first occurred constraint violation, the further contingency execution will be terminated, making the calculation faster, but with less information.

```
int ComSimoutage.ExecuteAndCheck([int workMode,]
                                 [int initMode,]
                                 [int considerOPFFlags,]
                                 [int considerVstep]
                                )
```

ARGUMENTS

workMode (optional)

the mode how the contingency analysis is executed; default is 0

- 0** Normal execution (doesn't check limits, write results file)
- 1** Check constraints (thermal loading, bus voltage range) and terminate execution if violates any limit, writes result file
- 2** Check constraints (thermal loading, bus voltage range) and terminate execution if violates any limit, does not write result file
- 3** Check thermal loading only and terminate execution if violates any limit, writes result file
- 4** Check thermal loading only and terminate execution if violates any limit, does not write result file
- 5** Check voltage limits only and terminate execution if violates any limit, writes result file
- 6** Check voltage limits only and terminate execution if violates any limit, does not write result file
- 7** Neither loading nor voltage limits are checked and terminate execution if any contingency fails, writes result file
- 8** Neither loading nor voltage limits are checked and terminate execution if any contingency fails, does not write result file

initMode (optional)

the mode how the contingency analysis is initialised; default is 0

- 0** full initialisation, new topology rebuild, and update all contingencies
- 1** new topology rebuild, but skip updating all contingencies
- 2** no topology rebuild, and no contingency update, which is the fastest mode

considerOPFflags (optional)

whether the voltage limit settings on OPF page will be considered or not; default is 1

- 0** the settings on OPF page are not considered.
- 1** the voltage limits are considered only when the check boxes on OPF page are ticked.

considerVstep (optional)

whether the voltage step limits will be considered or not; default is 1

- 0** the voltage step limits are not considered.
- 1** the voltage step limits are considered.

RETURNS

- 0** contingencies successfully executed without any violation
- 1** calculation was interrupted by the user
- 2** error occurred during contingency analysis
- 3** initialisation failed, such as base case load flow diverged
- 4** for AC/DC combined method, base case is already critical and calculation stopped
- 5** any loading constraint violated
- 6** any contingency cannot be analysed, non-convergent case
- 7** any voltage constraint violated
- 8** any constraint violated already in the base case

GetNTopLoadedElms*

To get certain number of top loaded components (most close to its limit).

```
void ComSimoutage.GetNTopLoadedElms(int number,
                                     set<elements>)
```

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 identifies energizing mode for each element.

```
int ComSimoutage.MarkRegions(int stage)
```

ARGUMENTS

stage which system stage to be analyzed, 0=stage1=2

RETURNS

0 On success.

1 On error.

RemoveAllRas

Removes all IntRas from to the ComSimoutage. Only deletes the references not the IntRas itself.

```
void ComSimoutage.RemoveAllRas()
```

RemoveContingencies

Removes all contingencies from the command. Functionality as "Remove All" button in dialog.

```
void ComSimoutage.RemoveContingencies()
```

EXAMPLE

```
object command;
command.RemoveContingencies();
```

RemoveRas

Removes the reference to a given IntRas from the ComSimoutage.

```
int ComSimoutage.RemoveRas(object ras)
```

ARGUMENTS

ras The IntRas object

RETURNS

- 0** If the reference to the IntRas has been successfully removed.
- 1** If the reference to the IntRas has not been in the container or on error.

Reset

Resets the intermediate results of the outage simulation.

```
int ComSimoutage.Reset()
```

RETURNS

- 0** On success.
- 1** On error.

SetLimits

Sets the recording limits for the Contingency Analysis.

```
void ComSimoutage.SetLimits(double vlmin,
                            double vlmax,
                            double 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.

3.4.45 ComSvgexport**Overview**

[SetFileName](#)
[SetObject](#)
[SetObjects](#)

SetFileName

Sets SVG file for export.

```
void ComSvgexport.SetFileName(string path)
```

ARGUMENTS

path Path of target SVG file

SetObject

Sets annotation layer or group for export.

```
void ComSvgexport.SetObject(object obj)
```

ARGUMENTS

obj Annotation layer (IntGrflayer) or group (IntGrfgroup) to be exported

SetObjects

Sets annotation layers and groups for export.

```
void ComSvgexport.SetObjects(set objs)
```

ARGUMENTS

objs Set of annotation layers (IntGrflayer) and/or groups (IntGrfgroup) to be exported

3.4.46 ComSvgimport

Overview

[SetFileName](#)
[SetObject](#)

SetFileName

Sets source SVG file for import.

```
void ComSvgimport.SetFileName(string path)
```

ARGUMENTS

path Path of SVG file to be imported

SetObject

Sets target annotation layer or group for import.

```
void ComSvgimport.SetObject(object obj)
```

ARGUMENTS

obj Target annotation layer (IntGrflayer) or group (IntGrfgroup)

3.4.47 ComTableReport

Overview

AddButton
AddCellAction
AddColumn
AddCurve
AddHeader
AddInvisibleFilter
AddListFilter
AddListFilterEntries
AddMultiListFilter
AddPlot
AddRow
AddTable
AddTabularFilter
AddTextFilter
AddXLabel
ClearAll
DisableAutomaticRowNumbering
DisableSingleFilterRefresh
DisplayButtonsCollapsible
DisplayFiltersCollapsible
DisplayFiltersIn2Columns
DisplayHeadersCollapsible
EnableAutomaticRowNumbering
ExportToExcel
ExportToHTML
FindNextCell
FindNextRow
FindPreviousCell
FindPreviousRow
GetCellAccessObject
GetCellAlignment
GetCellBackgroundColor
GetCellColor
GetCellEditObjects
GetCellFontStyle
GetCellFormat
GetCellHelpText
GetCellValueType
GetCheckboxCellValue
GetColumnFilter
GetColumnHeader
GetColumnId
GetDateCellValue
GetDoubleCellValue
GetIntCellValue
GetNumberOfColumns
GetNumberOfRows

GetNumberOfSelectedCells
 GetObjectCellValue
 GetRowCaption
 GetRowId
 GetSelectedCell
 GetSelectedElements
 GetSelection
 GetStringCellValue
 GoToCell
 HasCell
 HasColumnAutoFilter
 IsColumnScrollable
 IsColumnSortable
 IsRowVisible
 RemoveRow
 SetBarLimits
 SetCellAccess
 SetCellEdit
 SetCellValueToBar
 SetCellValueToCheckbox
 SetCellValueToDate
 SetCellValue.ToDouble
 SetCellValue.ToInt
 SetCellValueToObject
 SetCellValue.ToString
 SetColumnHeader
 SetCurveValue
 SetDialogSize
 SetExportHtml
 SetExportXls
 SetListFilterSelection
 SetNumberFormatForPlot
 SetRefresh
 SetSorting
 SetStatusText
 SetTextAxisDistForPlot
 SetTicksForPlot
 SetTitle

AddButton

Adds a button to the button section of the report.

```
int ComTableReport.AddButton(string id,
                            string label,
                            [int width,]
                            [int newLine,]
                            [int refresh]
                            )
```

ARGUMENTS

id Identifier

label Label text

width (optional)
Button width in pixel (default: 100)

newLine (optional)

- 0** Place button floating (default)
- 1** Place button in a new line

refresh (optional)

- 0** Do not refresh report after button was handled (default)
- 1** Refresh report after button was handled

RETURNS

- 0** On error: button with given id already exists.
- 1** Button was added to the report.

AddCellAction

Adds a cell action to a table cell. This means that an entry is added to the right mouse button menu of the cell.

```
int ComTableReport.AddCellAction(string tableId,
                                string columnId,
                                string rowId,
                                string actionId,
                                string displayText,
                                [set actionObjects,]
                                [int refresh]
                                )
```

ARGUMENTS

tableId Identifier of the table

columnId Identifier of the column

rowId Identifier of the row

actionId Identifier for the action

displayText (optional)

Text to be displayed in the right mouse button menu for this action

actionObjects (optional)

Objects needed to handle this action (default: empty). These objects are passed as input to the “Action” script.

refresh (optional)

- 0** Do not refresh report after action was handled
- 1** Refresh report after action was handled (default)

RETURNS

- 0** On error: cell with given ids does not exist.
- 1** Action was added.

AddColumn

Adds a column to the table.

```
void ComTableReport.AddColumn(string tableViewId,
                             string columnId,
                             string caption,
                             [int columnWidth,]
                             [int hasAutoFilter,]
                             [int isSortable,]
                             [int isScrollable]
                             )
```

ARGUMENTS

tableViewId Identifier of the table

columnId Identifier for the column to be inserted

caption Text to be displayed in the column header (lines separated by '\n')

columnWidth (optional)

>0 Initial column width in pixel

-1 Determine column width automatically according to values in the column (default)

hasAutoFilter (optional)

0 No auto filter (default)

1 Display an auto filter in the column header

isSortable (optional)

0 Column is not sortable

1 Column is sortable (default)

isScrollable (optional)

0 Column is fix (all columns left of this column must also be fix)

1 Column is scrollable (default)

EXAMPLE

```
object report;

report = this.GetParent();
if (!report) {
    exit();
}

report.AddTable('table');
report.AddColumn('table', 'scenario', 'Scenario', 100, 1, 1, 0);
```

AddCurve

Adds a curve to a plot.

```
int ComTablereport.AddCurve(string plotId,
                            string yText,
                            [string yUnit,]
                            [int lineWidth,]
                            [int lineStyle]
                            )
```

ARGUMENTS

- plotId* Identifier of the plot
- yText* Description for the curve
- yUnit (optional)*
Unit for the curve
- lineColor (optional)*
Line color for the curve (default: -1 = automatic)
- lineWidth (optional)*
Line width for the curve (default: -1 = automatic)
- lineStyle (optional)*
Line style for the curve (default: -1 = automatic)

RETURNS

- 0** On error.
- >0** Identifier of the inserted curve.

AddHeader

Adds a header to the report.

```
void ComTablereport.AddHeader(string label,
                             string text
                             )
```

ARGUMENTS

- label* Label
- text* Text

AddInvisibleFilter

Adds an invisible filter to the report. This filter can be used for storing objects and/or strings that can then be accessed again at the next refresh of the report.

```
int ComTablereport.AddInvisibleFilter(string id,
                                      string value,
                                      object obj
                                      )
```

ARGUMENTS

- id* Identifier
- value* String value to be stored
- obj* Object to be stored

RETURNS

- 0** On error: filter with given id already exists.
- 1** Filter was added to the report.

AddListFilter

Adds a list filter to the report.

```
int ComTableReport.AddListFilter(string id,
                                 string label,
                                 [string captions,]
                                 [set filterObjects,]
                                 [string selEntry,]
                                 [int showObjects]
int ComTableReport.AddListFilter(string id,
                                 string label,
                                 int showObjects
)
```

ARGUMENTS

id Identifier
label Label text
captions (optional) Captions for list entries ('\n'-separated)
filterObjects (optional) Objects for list entries (default: empty)
selEntry (optional) Selected list entry (default: empty)
showObjects (optional)

- 0** Object dialogues are not accessible (default)
- 1** Object dialogues are accessible

RETURNS

- 0** On error: filter with given id already exists.
- 1** Filter was added to the report.

AddListFilterEntries

Adds entries to an existing list filter of the report.

```
int ComTableReport.AddListFilterEntries(string id,
                                       string captions,
                                       [set filterObjects]
)
```

ARGUMENTS

id Identifier
captions Captions for list entries ('\n'-separated)
filterObjects (optional) Objects for list entries (default: empty)

RETURNS

- 0** On error: filter with given id does not exist.
- 1** Entries were added to the filter.

AddMultiListFilter

Adds a multi-selection list filter to the report.

```
int ComTablereport.AddMultiListFilter(string id,
                                      string label,
                                      [string captions,]
                                      [set filterObjects,]
                                      [string selEntries,]
                                      [int showObjects]
                                      )
int ComTablereport.AddMultiListFilter(string id,
                                      string label,
                                      int showObjects
                                      )
```

ARGUMENTS

id Identifier

label Label text

captions (optional)

Captions for list entries ('\n'-separated; default: empty)

filterObjects (optional)

Objects for list entries (default: empty)

selEntries (optional)

Selected list entries ('\n'-separated; default: empty)

showObjects (optional)

Object dialogues are not accessible (default)

1 Object dialogues are accessible

RETURNS

- 0** On error: filter with given id already exists.
- 1** Filter was added to the report.

AddPlot

Adds a plot to the report.

```
void ComTablereport.AddPlot(string plotId,
                            string xText,
                            [string xUnit,]
                            [string header,]
                            [int textLabels,]
                            [int splitCurves,]
                            [int trueDots,]
                            [int niceTicks,]
                            [int legendPos]
                            )
```

ARGUMENTS

plotId Identifier for the plot

xText Description text for x-axis

xUnit (optional)
Unit for x-axis

header (optional)
Header text for plot

textLabels (optional)

- 0** Use values for x-axis (default)
- 1** Use text labels for x-axis

splitCurves (optional)

- 0** Interpolate missing values (default)
- 1** Split curve at missing values

trueDots (optional)

- 0** Draw only line (default)
- 1** Draw true dots additionally

niceTicks (optional)

- 0** No nice ticks (default)
- 1** Nice ticks on x-axis
- 2** Nice ticks on y-axis
- 3** Nice ticks on both axes

legendPos (optional)

- 0** No legend
- 1** Display legend below plot (default)
- 2** Display legend on right side of plot

AddRow

Adds a row to the table.

```
void ComTableReport.AddRow(string tableId,
                           string rowId,
                           [string caption]
                           )
```

ARGUMENTS

tableId Identifier of the table

rowId Identifier for the row to be inserted

caption (optional)
Text to be displayed in the row header (only one line, default: empty)

EXAMPLE

```
object report;

report = this.GetParent();
if (!report) {
    exit();
}

report.AddTable('table');
report.AddColumn('table', 'scenario', 'Scenario', 100, 1, 1, 0);
report.AddRow('table', 'peak', 'Peak Load');
```

AddTable

Adds a table to the report. A report can contain only one table or one plot.

```
void ComTablereport.AddTable(string id)
```

ARGUMENTS

id Identifier

EXAMPLE

```
object report;

report = this.GetParent();
if (!report) {
    exit();
}

report.AddTable('table');
```

AddTabularFilter

Adds a tabular filter to the report.

```
int ComTablereport.AddTabularFilter(string id,
                                     string label,
                                     string columnLabels,
                                     string rowLabels,
                                     [string values]
                                     )
```

ARGUMENTS

id Identifier

label Label text for tabular filter

columnLabels

Label texts for table columns (';-separated; several lines for one column separated by '\n')

rowLabels

Label texts for table rows (';-separated; only one line per table row)

values (optional)
Values for text fields (';' -separated)

RETURNS

- 0** On error: filter with given id already exists.
- 1** Filter was added to the report.

AddTextFilter

Adds a text filter to the report.

```
int ComTablereport.AddTextFilter(string id,
                                 string label,
                                 [string unit,]
                                 [string text]
                               )
```

ARGUMENTS

id Identifier
label Label text
unit (optional) Unit text (default: empty)
text (optional) Text for text field (default: empty)

RETURNS

- 0** On error: filter with given id already exists.
- 1** Filter was added to the report.

AddXLabel

Sets a label for the next x-value.

```
void ComTablereport.AddXLabel(string plotId,
                             string label
                           )
```

ARGUMENTS

plotId Identifier of the plot
label Label text

ClearAll

Clears all report data.

```
void ComTablereport.ClearAll()
```

DisableAutomaticRowNumbering

Disables automatic row numbering. Row labels are filled with user-defined row labels. Automatic row numbering is enabled per default.

```
void ComTablereport.DisableAutomaticRowNumbering(string tableView)
```

ARGUMENTS

tableView Identifier of the table

EXAMPLE

```
object report;

report = this.GetParent();
if (!report) {
    exit();
}

report.AddTable('table');
report.DisableAutomaticRowNumbering('table');
```

DisableSingleFilterRefresh

Configures filter section to not refresh report immediately after a filter value was changed. A refresh button for the whole filter section is displayed instead.

```
void ComTablereport.DisableSingleFilterRefresh()
```

DisplayButtonsCollapsible

Makes button section collapsible and expandable.

```
void ComTablereport.DisplayButtonsCollapsible([int startExpanded])
```

ARGUMENTS

startExpanded (optional)

- 0 Button section is initially collapsed
- 1 Button section is initially expanded (default)

DisplayFiltersCollapsible

Makes filter section collapsible and expandable.

```
void ComTablereport.DisplayFiltersCollapsible([int startExpanded])
```

ARGUMENTS

startExpanded (optional)

- 0 Filter section is initially collapsed.
- 1 Filter section is initially expanded (default).

DisplayFiltersIn2Columns

Displays filters in the filter section of the report in two columns.

```
int ComTablereport.DisplayFiltersIn2Columns (string id)
```

ARGUMENTS

id Identifier of filter to be displayed as first filter in the second column

RETURNS

- 0** On error: filter with given id does not exist.
- 1** Filter was set.

DisplayHeadersCollapsible

Makes header section collapsible and expandable.

```
void ComTablereport.DisplayHeadersCollapsible ([int startExpanded])
```

ARGUMENTS

startExpanded (optional)

- 0** Header section is initially collapsed
- 1** Header section is initially expanded (default)

EnableAutomaticRowNumbering

Enables automatic row numbering. Row labels are filled automatically with ascending numbers (starting at 1), user-defined row labels are ignored. Automatic row numbering is enabled per default.

```
void ComTablereport.EnableAutomaticRowNumbering (string tableId)
```

ARGUMENTS

tableId Identifier of the table

ExportToExcel

Exports the report with its current filter settings and sorting to Excel.

```
void ComTablereport.ExportToExcel ([string file,]
                                    [int show]
                                    )
```

ARGUMENTS

file (optional)

Path and name of file to be written (default: empty = temp. file is written)

show (optional)

- 0** Write only
- 1** Open written file in default application (default)

ExportToHTML

Exports the report with its current filter settings and sorting to HTML.

```
void ComTableReport.ExportToHTML([string file,
                                 [int show]
                                )
```

ARGUMENTS

file (optional)

Path and name of file to be written (default: empty = temp. file is written)

show (optional)

0 Write only

1 Open written file in default application (default)

FindNextCell

Searches for the next table cell with the given value and sets the selection to that cell. Searches row-wise from the current position on until the table end and then from the top until the current cell is reached again considering filters and sorting.

```
int ComTableReport.FindNextCell(string tableId,
                               string cellValue
                              )
```

ARGUMENTS

tableId Identifier of the table

cellValue Display text of cell to be searched

RETURNS

0 On error: no matching cell found.

1 On success.

FindNextRow

Searches for the next row with the given values and sets the selection to that row. Searches from the current position on until the table end and then from the top until the current row is reached again considering filters and sorting.

```
int ComTableReport.FindNextRow(string tableId,
                             string columnId1,
                             string cellValue1,
                             [string columnId2,]
                             [string cellValue2,]
                             [string columnId3,]
                             [string cellValue3]
                            )
```

ARGUMENTS

tableId Identifier of the table

columnId1

Column identifier of cell for first criterion to be searched

cellValue1

Display text of cell for first criterion to be searched

columnId2

Column identifier of cell for second criterion to be searched

cellValue2

Display text of cell for second criterion to be searched

columnId3

Column identifier of cell for third criterion to be searched

cellValue3

Display text of cell for third criterion to be searched

RETURNS

- 0** On error: no matching row found.
- 1** On success.

FindPreviousCell

Searches for previous table cell with the given value and sets the selection to that cell. Searches row-wise from the current position on until table begin and then from the end until the current cell is reached again considering filters and sorting.

```
int ComTableReport.FindPreviousCell(string tableId,
                                    string cellValue
                                   )
```

ARGUMENTS

- tableId* Identifier of the table
- cellValue* Display text of cell to be searched

RETURNS

- 0** On error: no matching cell found.
- 1** On success.

FindPreviousRow

Searches for the previous row with the given values and sets the selection to that row. Searches from the current position on until the table begin and then from the end until the current row is reached again considering filters and sorting.

```
int ComTableReport.FindPreviousRow(string tableId,
                                    string columnId1,
                                    string cellValue1,
                                    [string columnId2,]
                                    [string cellValue2,]
                                    [string columnId3,]
                                    [string cellValue3]
                                   )
```

ARGUMENTS

tableId Identifier of the table
columnId1 Column identifier of cell for first criterion to be searched
cellValue1 Display text of cell for first criterion to be searched
columnId2 Column identifier of cell for second criterion to be searched
cellValue2 Display text of cell for second criterion to be searched
columnId3 Column identifier of cell for third criterion to be searched
cellValue3 Display text of cell for third criterion to be searched

RETURNS

- 0** On error: no matching row found.
- 1** On success.

GetCellAccessObject

Gets the object accessible from a table cell.

```
int ComTableReport.GetCellAccessObject(string tableId,
                                      string columnId,
                                      string rowId,
                                      object& obj)
```

ARGUMENTS

tableId Identifier of the table
columnId Identifier of the column
rowId Identifier of the row
obj (out) Object accessible from the cell if set, else NULL

RETURNS

- 0** On error: cell with given ids does not exist.
- 1** On success.

GetCellAlignment

Gets the alignment of a cell.

```
int ComTableReport.GetCellAlignment(string tableId,
                                    string columnId,
                                    string rowId,
                                    int& value
                                    )
```

ARGUMENTS

tableId Identifier of the table
columnId Identifier of the column
rowId Identifier of the row
value (out)
Alignment of the cell

RETURNS

0 On error: cell with given ids does not exist.
1 On success.

GetCellBackgroundColor

Gets the value type of a cell.

```
int ComTableReport.GetCellBackgroundColor(string tableId,
                                         string columnId,
                                         string rowId,
                                         int& value
                                         )
```

ARGUMENTS

tableId Identifier of the table
columnId Identifier of the column
rowId Identifier of the row
value (out)
Background color of the cell

RETURNS

0 On error: cell with given ids does not exist.
1 On success.

GetCellColor

Gets the value type of a cell.

```
int ComTableReport.GetCellColor(string tableId,
                               string columnId,
                               string rowId,
                               int& value
                               )
```

ARGUMENTS

tableId Identifier of the table
columnId Identifier of the column
rowId Identifier of the row
value (out)
Color of the cell

RETURNS

- 0** On error: cell with given ids does not exist.
- 1** On success.

GetCellEditObjects

Gets the objects linked to a table cell for editing the cell.

```
int ComTablereport.GetCellEditObjects(string tableId,
                                      string columnId,
                                      string rowId,
                                      set<obj> & objs)
```

ARGUMENTS

- tableId* Identifier of the table
- columnId* Identifier of the column
- rowId* Identifier of the row
- objs (out)* Edit objects of the cell if any, else empty

RETURNS

- 0** On error: cell with given ids does not exist.
- 1** On success.

GetCellFontStyle

Gets the font style of a cell.

```
int ComTablereport.GetCellFontStyle(string tableId,
                                    string columnId,
                                    string rowId,
                                    int & value
                                    )
```

ARGUMENTS

- tableId* Identifier of the table
- columnId* Identifier of the column
- rowId* Identifier of the row
- value (out)* Font style of the cell:

- 0** Normal (default)
- 1** Bold
- 2** Italic
- 3** Bold and italic
- 4** Underlined
- 5** Underlined and bold
- 6** Underlined and italic
- 7** Underlined, bold and italic

RETURNS

- 0** On error: cell with given ids does not exist.
- 1** On success.

GetCellFormat

Gets the format of a cell.

```
int ComTableReport.GetCellFormat(string tableId,  
                                string columnId,  
                                string rowId,  
                                string& value  
)
```

ARGUMENTS

- tableId* Identifier of the table
- columnId* Identifier of the column
- rowId* Identifier of the row
- value (out)* Format of the cell

RETURNS

- 0** On error: cell with given ids does not exist.
- 1** On success.

GetCellHelpText

Gets the help text of a cell.

```
int ComTableReport.GetCellHelpText(string tableId,  
                                 string columnId,  
                                 string rowId,  
                                 string& value  
)
```

ARGUMENTS

- tableId* Identifier of the table
- columnId* Identifier of the column
- rowId* Identifier of the row
- value (out)* Help text of the cell

RETURNS

- 0** On error: cell with given ids does not exist.
- 1** On success.

GetCellValueType

Gets the value type of a cell.

```
int ComTableReport.GetCellValueType(string tableId,
                                    string columnId,
                                    string rowId
                                   )
```

ARGUMENTS

tableId Identifier of the table

columnId Identifier of the column

rowId Identifier of the row

RETURNS

- | | |
|----------|---|
| 0 | Error: cell with given ids does not exist |
| 1 | String |
| 2 | Integer |
| 3 | Double |
| 4 | Check box |
| 5 | Bar |
| 6 | Date |
| 7 | Object |

GetCheckboxCellValue

Gets the value of a check box cell.

```
int ComTableReport.GetCheckboxCellValue(string tableId,
                                       string columnId,
                                       string rowId,
                                       int& value)
```

ARGUMENTS

tableId Identifier of the table

columnId Identifier of the column

rowId Identifier of the row

value (out)

Value of the cell

RETURNS

- | | |
|----------|---|
| 0 | On error: cell with given ids does not exist or has the wrong type. |
| 1 | On success. |

GetColumnFilter

Get filter text of given column in table.

```
int ComTableReport.GetColumnFilter(string tableId,
                                    string columnId,
                                    string& filter
                                   )
```

ARGUMENTS

tableId Identifier of the table
columnId Identifier of the column to be set
filter (out) Text that represents the filter of the column

RETURNS

0 Column was found.
1 Column not found.

GetColumnHeader

Gets caption text of existing column of the table.

```
int ComTableReport.GetColumnHeader(string tableId,
                                   string columnId,
                                   string& caption
                                 )
```

ARGUMENTS

tableId Identifier of the table
columnId Identifier of the column
caption (out) Text that is displayed in the column header (lines separated by '\n')

RETURNS

0 On success.
1 On error: column with given ids does not exist.

EXAMPLE

```
object report;
string header;

report = this.GetParent();
if (!report) {
  exit();
}

report.AddTable('table');
report.AddColumn('table', 'scenario', '', 100, 1, 1, 0);
report.SetColumnHeader('table', 'scenario', 'Scenario');
report.GetColumnHeader('table', 'scenario', header);
```

GetColumnId

Get the unique identifier of the n-th column.

```
int ComTableReport.GetColumnId(string tableId, int columnIndex, string& columnId)
```

ARGUMENTS

tableId Identifier of the table
columnIndex Index of the column - 0 based
columnId (out) Column identifier at the given index

RETURNS

0 On success.
1 On error: No column with the given index was found.

GetDateCellValue

Gets the value of a date cell.

```
int ComTableReport.GetDateCellValue(string tableId,
                                    string columnId,
                                    string rowId,
                                    int& value)
```

ARGUMENTS

tableId Identifier of the table
columnId Identifier of the column
rowId Identifier of the row
value (out) Value of the cell

RETURNS

0 On error: cell with given ids does not exist or has the wrong type.
1 On success.

GetDoubleCellValue

Gets the value of a double cell.

```
int ComTableReport.GetDoubleCellValue(string tableId,
                                     string columnId,
                                     string rowId,
                                     double& value)
```

ARGUMENTS

tableId Identifier of the table
columnId Identifier of the column
rowId Identifier of the row
value (out) Value of the cell

RETURNS

- 0** On error: cell with given ids does not exist or has the wrong type.
- 1** On success.

GetIntCellValue

Gets the value of an integer cell.

```
int ComTablereport.GetIntCellValue(string tableId,
                                    string columnId,
                                    string rowId,
                                    int& value
                                   )
```

ARGUMENTS

- tableId* Identifier of the table
- columnId* Identifier of the column
- rowId* Identifier of the row
- value (out)* Value of the cell

RETURNS

- 0** Error: cell with given ids does not exist or has the wrong type
- 1** Ok

GetNumberOfColumns

Returns the number of columns that were added to the table report.

```
int ComTablereport.GetNumberOfColumns(string tableId)
```

ARGUMENTS

- tableId* Identifier of the table

RETURNS

- >=0** On success: the number of columns in the table.
- 1** On error: no table with the given ID was defined.

GetNumberOfRows

Returns the number of rows that were added to the table report.

```
int ComTablereport.GetNumberOfRows(string tableId)
```

ARGUMENTS

- tableId* Identifier of the table

RETURNS

- >=0** On success: the number of rows in the table.
- 1** On error: no table with the given ID was defined.

GetNumberOfSelectedCells

Gets the number of (visible) cells selected in the table.

```
int ComTableReport.GetNumberOfSelectedCells(string tableId,
                                         [string columnId,]
                                         [string rowId])
                                         )
```

ARGUMENTS

- tableId* Identifier of the table
- columnId (optional)*
 - set** Count only selected cells with this column id
 - empty** Count selected cells in all columns (default)
- rowId (optional)*
 - set** Count only selected cells with this row id
 - empty** Count selected cells in all rows (default)

RETURNS

Number of selected cells

GetObjectCellValue

Gets the value of an object cell.

```
int ComTableReport.GetObjectCellValue(string tableId,
                                     string columnId,
                                     string rowId,
                                     object & value)
```

ARGUMENTS

- tableId* Identifier of the table
- columnId* Identifier of the column
- rowId* Identifier of the row
- value (out)*
 - Value of the cell

RETURNS

- 0** On error: cell with given ids does not exist or has the wrong type.
- 1** On success.

GetRowCaption

Gets caption text of existing row of the table.

```
int ComTablereport.GetRowCaption(string tableView,
                                  string rowId,
                                  string& caption
)
```

ARGUMENTS

tableView Identifier of the table

rowId Identifier of the row

caption (out)
Text that is displayed in the row caption

RETURNS

0 On success.
1 On error: row with given ids does not exist.

GetRowIndex

Get the unique identifier of the n-th row.

```
int ComTablereport.GetRowIndex(string tableView, int rowIndex, string& rowId)
```

ARGUMENTS

tableView Identifier of the table

rowIndex Index of the row - 0 based

rowId (out)
Row identifier at the given index

RETURNS

0 On success.
1 On error: No row with the given index was found.

GetSelectedCell

Gets the identifiers for a selected cell.

```
int ComTablereport.GetSelectedCell(string tableView,
                                   int index,
                                   string& columnId,
                                   string& rowId
)
```

ARGUMENTS

tableView Identifier of the table

index Index of selected cell (1-based)

columnId (in, out)

in Consider only selected cells with this column id
out Column id of selected cell

rowId (*in, out*)

in Consider only selected cells with this row id
out Row id of selected cell

RETURNS

- 0** On error: cell with given ids does not exist.
- 1** On success.

GetSelectedElements

Gets objects selected for the report filtered according to filter in the report.

```
set ComTableReport.GetSelectedElements()
```

RETURNS

Filtered selected objects

GetSelection

Gets SetSelect with selected objects for the report.

```
object ComTableReport.GetSelection()
```

RETURNS

SetSelect or NULL.

GetStringCellValue

Gets the value of a string cell.

```
int ComTableReport.GetStringCellValue(string tableId,  
                                     string columnId,  
                                     string rowId,  
                                     string& value  
)
```

ARGUMENTS

tableId Identifier of the table
columnId Identifier of the column
rowId Identifier of the row
value (out) Value of the cell

RETURNS

- 0** On error: cell with given ids does not exist or has the wrong type.
- 1** On success.

GoToCell

Sets the selection to a table cell.

```
int ComTablereport.GoToCell(string tableView,
                            string columnId,
                            string rowId
                           )
```

ARGUMENTS

tableView Identifier of the table

columnId Column identifier of the cell to be selected

rowId Row identifier of the cell to be selected

RETURNS

0 On error: cell with given ids does not exist.

1 On success.

HasCell

Checks if a table cell exists.

```
int ComTablereport.HasCell(string tableView,
                           string columnId,
                           string rowId
                          )
```

ARGUMENTS

tableView Identifier of the table

columnId Identifier of the column

rowId Identifier of the row

RETURNS

0 Cell with given ids does not exist.

1 Cell was found.

HasColumnAutoFilter

Checks if the column has an enabled auto filter.

```
int ComTablereport.HasColumnAutoFilter(string tableView, string columnId)
```

ARGUMENTS

tableView Identifier of the table

columnId Identifier of the column

RETURNS

- 1** The column is not defined.
- 0** The column has no auto filter.
- 1** The column has an auto filter.

IsColumnScrollable

Checks if the column is marked as scrollable.

```
int ComTableReport.IsColumnScrollable(string tableId, string columnId)
```

ARGUMENTS

- tableId* Identifier of the table
- columnId* Identifier of the column

RETURNS

- 1** The column is not defined.
- 0** The column is not scrollable.
- 1** The column is scrollable.

IsColumnSortable

Checks if the column is marked as sortable.

```
int ComTableReport.IsColumnSortable(string tableId, string columnId)
```

ARGUMENTS

- tableId* Identifier of the table
- columnId* Identifier of the column

RETURNS

- 1** The column is not defined.
- 0** The column is not sortable.
- 1** The column is sortable.

IsRowVisible

Checks if the row with the given ID is visible for the user. The result depends on the filter settings of the report.

```
int ComTableReport.IsRowVisible(string tableId, string rowId)
```

ARGUMENTS

- tableId* Identifier of the table
- rowId* Identifier of the row

RETURNS

- 1** The row is not defined.
- 0** The row is invisible.
- 1** The row is visible.

RemoveRow

Removes a row from the table.

```
int ComTableReport.RemoveRow(string tableId,
                            string rowId
                           )
```

ARGUMENTS

- tableId* Identifier of the table
- rowId* Identifier of the row to be removed

RETURNS

- 0** Row with given id does not exist.
- 1** Row was removed.

EXAMPLE

```
object report;

report = this.GetParent();
if (!report) {
    exit();
}

report.AddTable('table');
report.AddColumn('table', 'scenario', 'Scenario', 100, 1, 1, 0);
report.AddRow('table', 'peak', 'Peak Load');
report.RemoveRow('table', 'peak');
```

SetBarLimits

Sets bar limits for all bar cells in an existing table column.

```
int ComTableReport.SetBarLimits(string tableId,
                               string columnId,
                               int min,
                               int max
                              )
```

ARGUMENTS

- tableId* Identifier of the table
- columnId* Identifier of the column
- min* Minimum value for all bars in the column
- max* Maximum value for all bars in the column

RETURNS

- 0** On error: column with given ids does not exist.
- 1** Limits were set.

SetCellAccess

Makes a cell accessible. Accessible means that an object dialogue is displayed on double click and the right mouse button menu additionally provides the entries 'Edit', 'Edit and Browse' and 'Mark in Graphic'.

```
void ComTablereport.SetCellAccess(string tableId,
                                  string columnId,
                                  string rowId,
                                  object accessObject,
                                  [string parameterName]
                                )
```

ARGUMENTS

tableId Identifier of the table

columnId Identifier of the column

rowId Identifier of the row

accessObject
Object to be accessible

parameterName (optional)

Set Dialog is shown at the page on that this variable is displayed

Empty Dialog is shown at stored dialog page (default)

SetCellEdit

Makes a table cell editable. This means that the cell value can be modified in the report window.

```
void ComTablereport.SetCellEdit(string tableId,
                               string columnId,
                               string rowId,
                               set editObjects,
                               [string selectableValues])
```

ARGUMENTS

tableId Identifier of the table

columnId Identifier of the column

rowId Identifier of the row

editObjects
Objects to handle the modification. These objects are passed as input to the "Edit" script.

selectableValues
\n'-separated list of selectable values:

Set A list dialogue with the given values is displayed for selection if the cell is edited

Empty Any text can be entered into the cell (default)

SetCellValueToBar

Fills a table cell with a bar.

```
void ComTableReport.SetCellValueToBar(string tableId,
                                      string columnId,
                                      string rowId,
                                      string barDesc,
                                      [string helpText,]
                                      [int border,]
                                      [int align,]
                                      [int backgroundColor]
                                      )
```

ARGUMENTS

tableId Identifier of the table

columnId Identifier of the column

rowId Identifier of the row

barDesc Bar description containing limits and segment widths and colours.
If limits are given the left cell side is considered to be the lower limit value and the right cell side is considered to be the upper limit value.
If no limits are given the limits are implicitly set to 0 and 100.
The available formats are listed below.

helpText (optional)

Text shown in balloon help of the cell (multiple lines separated by '\n'; default: empty)

border (optional)

Display a thin black border around the bar:

0 No border

1 Border (default)

align (optional)

Alignment for text:

0 Automatically according to data type (default)

1 Left

2 Center

3 Right

backgroundColor (optional)

Color for background (default: 0=white)

BAR DESCRIPTION FORMATS

<width>,<colour>;<width>,<colour>;<...>
 Relative widths in % (old style; e.g. "20,1;14,2;36,3;30,4")

<lower limit>;<upper limit>#<width>,<colour>;<width>,<colour>;<...>
 Relative widths in %, limits are ignored (old style; e.g. "90;140#10,1;7,2;18,3;15,4")

0#<width>,<colour>;<width>,<colour>;<...>
 Relative widths in % (e.g. "0#20,1;14,2;36,3;30,4")

0#<lower limit>;<upper limit>#<width>,<colour>;<width>,<colour>;<...>
 Relative widths within limits (e.g. "0#90;140#10,1;7,2;18,3;15,4")

1#<value>,<colour>;<value>,<colour>;<...>
 Absolute values in % (e.g. "1#20,1;34,2;70,3;100,4")

1#<lower limit>;<upper limit>#<value>,<colour>;<value>,<colour>;<...>
 Absolute values within limits(e.g. "1#90;140#100,1;107,2;125,3;140,4")

SetCellValueToCheckbox

Fills a table cell with a check box.

```
void ComTableReport.SetCellValueToCheckbox(string tableId,
                                         string columnId,
                                         string rowId,
                                         int value,
                                         [string helpText,]
                                         [int align,]
                                         [int backgroundColor]
                                         )
```

ARGUMENTS

tableId Identifier of the table

columnId Identifier of the column

rowId Identifier of the row

value Integer value to be set in the checkbox:

0 Check box is unchecked

1 Check box is checked

helpText (optional)

Text shown in balloon help of the cell (multiple lines separated by '\n'; default: empty)

align (optional)

Alignment for text:

0 Automatically according to data type (default)

1 Left

2 Center

3 Right

backgroundColor (optional)

Color for background (default: 0=white)

SetCellValueToDate

Fills a table cell with a date.

```
void ComTableReport.SetCellValueToDate(string tableId,
                                      string columnId,
                                      string rowId,
                                      int timeStamp,
                                      [string format,]
                                      [string helpText,]
                                      [int color,]
                                      [int fontStyle,]
                                      [int align,]
                                      [int backgroundColor]
                                      )
```

ARGUMENTS

tableId Identifier of the table

columnId Identifier of the column

rowId Identifier of the row

timeStamp
Time stamp value for date and time

format (optional)
Format for displaying date (currently ignored; default: "")

helpText (optional)
Text shown in balloon help of the cell (multiple lines separated by '\n'; default: empty)

color (optional)
Colour for text (default: 1=black)

fontStyle (optional)

0	Normal (default)
1	Bold
2	Italic
3	Bold and italic
4	Underlined
5	Underlined and bold
6	Underlined and italic
7	Underlined, bold and italic

align (optional)
Alignment for text:

0	Automatically according to data type (default)
1	Left
2	Center
3	Right

backgroundColor (optional)
Color for background (default: 0=white)

SetCellValueToDouble

Fills a table cell with a double value.

```
void ComTableReport.SetCellValueToDouble(string tableId,
                                         string columnId,
                                         string rowId,
                                         double value,
                                         [string format,]
                                         [string helpText,]
                                         [int color,]
                                         [int fontStyle,]
                                         [int align,]
                                         [int backgroundColor]
                                         )
```

ARGUMENTS

tableId Identifier of the table

columnId Identifier of the column

rowId Identifier of the row

value Double value to be set

format (optional)

Printf-like format for displaying the value (default: '%f')

helpText (optional)

Text shown in balloon help of the cell (multiple lines separated by '\n'; default: empty)

color (optional)

Color for text (default: 1=black)

fontStyle (optional)

0 Normal (default)

1 Bold

2 Italic

3 Bold and italic

4 Underlined

5 Underlined and bold

6 Underlined and italic

7 Underlined, bold and italic

align (optional)

Alignment for text:

0 Automatically according to data type (default)

1 Left

2 Center

3 Right

backgroundColor (optional)

Color for background (default: 0=white)

SetCellValueToInt

Fills a table cell with an integer value.

```
void ComTableReport.SetCellValueToInt(string tableId,
                                      string columnId,
                                      string rowId,
                                      int value,
                                      [string format,]
                                      [string helpText,]
                                      [int color,]
                                      [int fontStyle,]
                                      [int align,]
                                      [int backgroundColor]
                                      )
```

ARGUMENTS

tableId Identifier of the table

columnId Identifier of the column

rowId Identifier of the row

value Integer value to be set

format (optional)

Printf-like format for displaying the value (default: '%d')

helpText (optional)

Text shown in balloon help of the cell (multiple lines separated by '\n'; default: empty)

color (optional)

Color for text (default: 1=black)

fontStyle (optional)

0 Normal (default)

1 Bold

2 Italic

3 Bold and italic

4 Underlined

5 Underlined and bold

6 Underlined and italic

7 Underlined, bold and italic

align (optional)

Alignment for text:

0 Automatically according to data type (default)

1 Left

2 Center

3 Right

backgroundColor (optional)

Color for background (default: 0=white)

SetCellValueToObject

Fills a table cell with an object.

```
void ComTableReport.SetCellValueToObject(string tableId,
                                         string columnId,
                                         string rowId,
                                         object obj,
                                         [string format,]
                                         [string helpText,]
                                         [int color,]
                                         [int fontStyle,]
                                         [int align,]
                                         [int backgroundColor]
                                         )
```

ARGUMENTS

tableId Identifier of the table

columnId Identifier of the column

rowId Identifier of the row

obj Object to be set

format (optional)

Format for displaying object (currently ignored; default: "")

helpText (optional)

Text shown in balloon help of the cell (multiple lines separated by '\n'; default: empty)

color (optional)

Colour for text (default: 1=black)

fontStyle(optional)

0 Normal (default)

1 Bold

2 Italic

3 Bold and italic

4 Underlined

5 Underlined and bold

6 Underlined and italic

7 Underlined, bold and italic

align (optional)

Alignment for text:

0 Automatically according to data type (default)

1 Left

2 Center

3 Right

backgroundColor (optional)

Color for background (default: 0=white)

SetCellValueToString

Fills a table cell with a string value.

```
void ComTableReport.SetCellValueToString(string tableId,
                                         string columnId,
                                         string rowId,
                                         string value,
                                         [string format,]
                                         [string helpText,]
                                         [int color,]
                                         [int fontStyle,]
                                         [int align,]
                                         [int backgroundColor]
                                         )
```

ARGUMENTS

tableId Identifier of the table

columnId Identifier of the column

rowId Identifier of the row

value String value to be set

format (optional)

Printf-like format for displaying the value (default: '%s')

helpText (optional)

Text shown in balloon help of the cell (multiple lines separated by '\n'; default: empty)

color (optional)

Color for text (default: 1=black)

fontStyle (optional)

0 Normal (default)

1 Bold

2 Italic

3 Bold and italic

4 Underlined

5 Underlined and bold

6 Underlined and italic

7 Underlined, bold and italic

align (optional)

Alignment for text:

0 Automatically according to data type (default)

1 Left

2 Center

3 Right

backgroundColor (optional)

Color for background (default: 0=white)

EXAMPLE

The following example writes the name of the active scenario into a table cell:

```
object report, scenario;
string name;

report = this.GetParent();
if (!report) {
    exit();
}

report.AddTable('table');
report.AddColumn('table', 'scenario', 'Scenario', 100, 1, 1, 0);
report.AddRow('table', 'active', 'Active');

scenario = GetActiveScenario();
name = scenario:loc_name;
report.SetCellValueToString('table', 'scenario', 'active', name);
```

SetColumnHeader

Sets caption text in existing column of the table.

```
void ComTablereport.SetColumnHeader(string tableId,
                                    string columnId,
                                    string caption
                                    )
```

ARGUMENTS

tableId Identifier of the table

columnId Identifier of the column to be set

captionId Text to be displayed in the column header (lines separated by '\n')

EXAMPLE

```
object report;

report = this.GetParent();
if (!report) {
    exit();
}

report.AddTable('table');
report.AddColumn('table', 'scenario', '', 100, 1, 1, 0);
report.SetColumnHeader('table', 'scenario', 'Scenario');
```

SetCurveValue

Adds a y-value at a certain x-value to a curve.

```
int ComTablereport.SetCurveValue(string plotId,
                                 int curveId,
                                 double xValue,
                                 double yValue
                                 )
```

ARGUMENTS

plotId Identifier of the plot
curveId Identifier of the curve (0: no curve / new x-value)
xValue x-value for given y-value
yValue y-value

RETURNS

0 On error: curve not found.
1 On success.

SetDialogSize

Sets the width and height for the report dialog.

```
void ComTablereport.SetDialogSize(int width,
                                  int height
                                )
```

ARGUMENTS

width Dialogue width in pixel
height Dialogue height in pixel

EXAMPLE

```
object report;

report = this.GetParent();
if (!report) {
  exit();
}

report.SetDialogSize(700, 860);
```

SetExportHtml

Shows or hides html icon

```
void ComTablereport.SetExportHtml(int show)
```

ARGUMENTS

show
0 Hide html icon
1 Show html icon

SetExportXls

Shows or hides xlsx icon

```
void ComTablereport.SetExportXls(int show)
```

ARGUMENTS

show

- 0** Hide xlsx icon
- 1** Show xlsx icon

SetListFilterSelection

Sets selected entries in existing list filter.

```
int ComTablereport.SetListFilterSelection(string id,
                                         string selEntries
                                         )
```

ARGUMENTS

id Identifier*selEntries (optional)*

Selected list entries ('\n'-separated; default: empty)

RETURNS

- 0** On error: filter with given id does not exist.
- 1** Entries were set as selected filter entries.

SetNumberFormatForPlot

Sets the number format for the tick descriptions for an axis of a plot.

```
int ComTablereport.SetNumberFormatForPlot(string plotId,
                                         string axis,
                                         int characters,
                                         int precision
                                         )
```

ARGUMENTS

plotId Identifier of the plot*axis*

- X** Set format for x-axis
- Y** Set format for y-axis

characters

Number of characters

precision Number of digits after the point

RETURNS

- 0** On error: plot not found.
- 1** On success.

SetRefresh

Shows or hides refresh icon

```
void ComTablereport.SetRefresh(int show)
```

ARGUMENTS

show

- 0** Hide refresh icon
- 1** Show refresh icon

SetSorting

Sets initial sorting of table.

```
int ComTablereport.SetSorting(string tableViewId,
                               string columnId,
                               [int descending]
                               )
```

ARGUMENTS

tableViewId Identifier of the table

columnId Identifier of the column according to that is to be sorted

descending (optional)

- 0** Ascending order (default)
- 1** Descending order

RETURNS

- 0** On error.
- 1** Sorting set.

SetStatusText

Sets a text in the user-definable field of the report window's status bar.

```
void ComTablereport.SetStatusText(string tableViewId,
                                   string text
                                   )
```

ARGUMENTS

tableViewId Identifier of the table

text Text to be set

SetTextAxisDistForPlot

Sets distance between axis and tick description for an axis of a plot.

```
int ComTablereport.SetTextAxisDistForPlot(string plotId,
                                         string axis,
                                         int distance
                                         )
```

ARGUMENTS

plotId Identifier of the plot
axis
 X Set distance for x-axis
 Y Set distance for y-axis
distance Distance between text and axis

RETURNS

0 On error: plot not found.
1 On success.

SetTicksForPlot

Sets the number of ticks for an axis of a plot.

```
int ComTablereport.SetTicksForPlot(string plotId,
                                    string axis,
                                    int main,
                                    int number
                                   )
```

ARGUMENTS

plotId Identifier of the plot
axis
 X Set ticks on x-axis
 Y Set ticks on y-axis
main
 0 Set help ticks
 1 Set main ticks
number Number of ticks

RETURNS

0 On error: plot not found.
1 On success.

SetTitle

Sets the text in the title bar of the report window.

```
void ComTablereportSetTitle(string title)
```

ARGUMENTS

title Title text

EXAMPLE

```
object report;

report = this.GetParent();
if (!report) {
    exit();
}

reportSetTitle('Input Data Summary');
```

3.4.48 ComTasks

Overview

AppendCommand
AppendStudyCase
GetCommandsForStudyCase
GetNumberOfCommandsForStudyCase
GetNumberOfStudyCases
GetStudyCases
IsAdditionalResultsFlagSetForCommand
IsCommandIgnored
IsStudyCaseIgnored
RemoveCmdsForStudyCaseRow
RemoveCommand
RemoveStudyCase
RemoveStudyCases
SetAdditionalResultsFlagForCommand
SetIgnoreFlagForCommand
SetIgnoreFlagForStudyCase
SetResultsFolder

AppendCommand

Appends a command for calculation.

```
int ComTasks.AppendCommand(object 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.

EXAMPLE

In this example, we add the load flow command of a study case to the list of tasks to calculate.

```
int isAdded;
object comLdf,
      comTasks,
      casesFolder;
set commands;

casesFolder = GetProjectFolder('study');
commands = casesFolder.GetContents('*.*ComTasks');
if (commands.Count()=1) {
    comTasks = commands.First();
    comLdf = GetFromStudyCase('ComLdf');
    if ({comTasks<>nullptr}.and.{comLdf<>nullptr}) {
        isAdded = comTasks.AppendCommand(comLdf);
        if ( isAdded<>0 ) {
            printf('Command %o has been added for calculation', comLdf);
        }
        else {
            printf('Command %o could not be added for calculation', comLdf);
        }
    }
}
```

AppendStudyCase

Appends a study case to the list of study cases for calculation.

```
int ComTasks.AppendStudyCase(object 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.

EXAMPLE

In this example, we add the currently active study case to the list of study cases for calculation.

```
int isAdded;
object newCase,
      comTasks,
      casesFolder;
set commands;

casesFolder = GetProjectFolder('study');
commands = casesFolder.GetContents('*.*ComTasks');
if (commands.Count()=1) {
    comTasks = commands.First();
    newCase = GetActiveStudyCase();
    if ({comTasks<>nullptr}.and.{newCase<>nullptr}) {
```

```

isAdded = comTasks.AppendStudyCase(newCase);
if ( isAdded<>0 ) {
    printf('Study Case %o has been added for calculation', newCase);
}
else {
    printf('Study case %o could not be added for calculation.', newCase);
}
}
}

```

GetCommandsForStudyCase

Get all selected commands to be processed for a study case from the Task Automation command.

```
set ComTasks.GetCommandsForStudyCase(object studyCase,
                                    [int studyCaseRow])
```

RETURNS

Returns ordered set of all selected commands to be processed for a study case.

ARGUMENTS

studyCase

Study case to get all selected commands to be processed from.

studyCaseRow

- ≤ 0 Commands for the first matching study case found will be returned.
- > 0 Optionally, the row in the study case table containing the study case for which commands shall be returned can be passed. This is helpful, e.g., if a study case has been added multiple times for calculation with different command lists.

EXAMPLE

In this example, we get all selected commands for study case of row 1 from the Task Automation command.

```

object comTasks,
        casesFolder;
set commands,
        allSelectedCommands;

casesFolder = GetProjectFolder('study');
commands = casesFolder.GetContents('*.*.ComTasks');
if (commands.Count()=1) {
    comTasks = commands.First();
    if (comTasks<>NULL) {
        allSelectedCommands = comTasks.GetCommandsForStudyCase(NULL, 1);
    }
}

```

GetNumberOfCommandsForStudyCase

Get number of all selected commands to be processed for a study case from the Task Automation command.

```
int ComTasks.GetNumberOfCommandsForStudyCase (object studyCase,
                                             [int studyCaseRow])
```

RETURNS

Returns number of all selected commands to be processed for a study case.

ARGUMENTS

studyCase

Study case to get number of selected commands to be processed from.

studyCaseRow

≤ 0 Number of commands for the first matching study case found will be returned.

> 0 Optionally, the row in the study case table containing the study case for which number of commands shall be returned can be passed. This is helpful, e.g., if a study case has been added multiple times for calculation with different command lists.

GetNumberOfStudyCases

Get number of selected study cases from the Task Automation command.

```
int ComTasks.GetNumberOfStudyCases ()
```

RETURNS

Returns number of selected study cases from the Task Automation command.

GetStudyCases

Get all selected study cases from the Task Automation command.

```
set ComTasks.GetStudyCases ()
```

RETURNS

Returns ordered set of all selected study cases from the Task Automation command, set is empty on failure.

EXAMPLE

In this example, we get all selected study cases from the Task Automation command.

```
object comTasks,
       casesFolder;
set commands,
     allSelectedCases;

casesFolder = GetProjectFolder('study');
commands = casesFolder.GetContents('*.*.ComTasks');
if (commands.Count()=1) {
  comTasks = commands.First();
  if (comTasks<>NULL) {
```

```

        allSelectedCases = comTasks.GetStudyCases();
    }
}

```

IsAdditionalResultsFlagSetForCommand

Returns whether additional results flag of a command is set or not (using study case row and task row / using a command directly).

```

int ComTasks.IsAdditionalResultsFlagSetForCommand(int studyCaseRow,
                                                int taskRow)
int ComTasks.IsAdditionalResultsFlagSetForCommand(object command)

```

ARGUMENTS

studyCaseRow ≤ 0

The row in the study cases table of the command's study case for which the additional results flag shall be derived.

taskRow > 0

The row in the commands table for which the additional results flag shall be derived.

command

The command to get the additional results flag for.

RETURNS

- 0** Additional results flag of command is not set.
- 1** Additional results flag of command is set.
- 1** Additional results flag was not found for provided study case row and task row / provided command.

IsCommandIgnored

Returns whether the processing of a command will be ignored or not (using study case row and task row / using provided command).

```

int ComTasks.IsCommandIgnored(int studyCaseRow,
                             int taskRow)
int ComTasks.IsCommandIgnored(object command)

```

ARGUMENTS

studyCaseRow ≤ 0

The row in the study cases table of the command's study case for which the ignore flag shall be derived.

taskRow > 0

The row in the commands table for which the ignore flag shall be derived.

command

The command to get the ignore-flag for.

RETURNS

- 0** Processing of command will be done.
- 1** Processing of command will be ignored.
- 1** Command was not found (for provided study case row and task row / at all).

IsStudyCaseIgnored

Returns whether the command processing for a study case will be ignored or not.

```
int ComTasks.IsStudyCaseIgnored(object studyCase,
                               [int studyCaseRow])
```

RETURNS

- 0** Command processing for study case will be done.
- 1** Command processing for study case will be ignored.
- 1** study case or row was not found.

ARGUMENTS

studyCase

Study case to get ignore-flag from.

studyCaseRow

- ≤ 0 Ignore flag for the first matching study case found will be returned.
- > 0 Optionally, the row in the study case table containing the study case for which ignore-flag shall be returned can be passed. This is helpful, e.g., if a study case has been added multiple times for calculation with different command lists.

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 > 0

The row in the study case table containing the study case for which all commands shall be removed.

EXAMPLE

In this example, we remove all commands of the study case in row 1 from the calculation.

```
int isRemoved;
object comTasks,
       casesFolder;
set commands;
```

```

casesFolder = GetProjectFolder('study');
commands = casesFolder.GetContents('*.*ComTasks');
if (commands.Count() == 1) {
    comTasks = commands.First();
    if (comTasks != nullptr) {
        isRemoved = comTasks.RemoveCmdsForStudyCaseRow(1);
        if (isRemoved > 0) {
            printf('All commands are successfully removed for row %d', 1);
        }
        else {
            printf('Commands could not be removed');
        }
    }
}

```

RemoveCommand

Remove a command from the calculation (for study case row and task row / for all appearances).

```

int ComTasks.RemoveCommand(int studyCaseRow,
                           int taskRow
                           )
int ComTasks.RemoveCommand(object command)

```

ARGUMENTS

studyCaseRow> 0

The row in the study cases table of the command's study case for which the command shall be removed.

taskRow> 0

The row in the commands table for which the command shall be removed.

command

Command to remove.

RETURNS

0 Command could not be removed.

1 Command has been successfully removed.

RemoveStudyCase

Removes a study case from the study cases table.

```

int ComTasks.RemoveStudyCase(object studyCase,
                            [int studyCaseRow]
                            )

```

RETURNS

0 Study case could not be removed.

1 Study case has been successfully removed.

ARGUMENTS

studyCase

Study case which shall be removed.

studyCaseRow

- ≤ 0 Study case is removed for all entries in the study cases table matching the provided study case.
- > 0 Optionally, the row in the study case table containing the study case which shall be removed can be passed. This is helpful, e.g., if a study case has been added multiple times for calculation with different command lists.

RemoveStudyCases

Removes all study cases from calculation.

```
void ComTasks.RemoveStudyCases()
```

EXAMPLE

In this example we remove all study cases from the list of study cases for calculation.

```
object comTasks,
       casesFolder;
set commands;

casesFolder = GetProjectFolder('study');
commands = casesFolder.GetContents('*.*ComTasks');
if (commands.Count()=1) {
    comTasks = commands.First();
    if (comTasks<>nullptr) {
        comTasks.RemoveStudyCases();
    }
}
```

SetAdditionalResultsFlagForCommand

Set the flag whether to record an additional result file for a command of the calculation (using study case row and task row / using command).

```
int ComTasks.SetAdditionalResultsFlagForCommand(int studyCaseRow,
                                                int taskRow,
                                                int addResVal
                                               )
int ComTasks.SetAdditionalResultsFlagForCommand(object command,
                                                int addResVal
                                               )
```

ARGUMENTS

studyCaseRow > 0

The row in the study cases table of the command's study case for which the flag to record additional results shall be set.

taskRow > 0

The row in the commands table for which the flag to record additional results shall be set.

command

The command for which the flag to record additional results shall be set.

addResVal

- 1 Will set the command to record additional results for the calculation.
- 0 Will set the command to avoid recording of results for calculation.

RETURNS

- 0** Flag to record (or not record) additional results could not be set.
- 1** Flag to record (or not record) additional results has been successfully set.

EXAMPLE

In this example, we set the command of study case row 1 and command row 1 to record additional results for calculation.

```
int setIsSet;
object comTasks,
      casesFolder;
set commands;

casesFolder = GetProjectFolder('study');
commands = casesFolder.GetContents('*.*ComTasks');
if (commands.Count()=1) {
    comTasks = commands.First();
    if (comTasks<>nullptr) {
        setIsSet = SetAdditionalResultsFlagForCommand(1, 1, 1);
        if ( setIsSet<>0 ) {
            printf('Command in study case row 1 and command row 1 records additional results');
        }
        else {
            printf('Flag to record additional results could not be set');
        }
    }
}
```

In this example, we add the load flow command to the calculation and set the flag to record additional results afterwards.

```
int isAdded,
    isRecordSet;
object comLdf,
      comTasks,
      casesFolder;
set commands;

casesFolder = GetProjectFolder('study');
commands = casesFolder.GetContents('*.*ComTasks');
if (commands.Count()=1) {
    comTasks = commands.First();
    comLdf = GetFromStudyCase('ComLdf');
    if ({comTasks<>nullptr}.and.{comLdf<>nullptr}) {
        isAdded = comTasks.AppendCommand(comLdf);
        if ( isAdded<>0 ) {
            printf('Command %o has been added for calculation', comLdf);
            isRecordSet = SetAdditionalResultsFlagForCommand(comLdf, 1);
            if ( isRecordSet<>0 ) {
                printf('Flag to record additional results set for cmd %o', comLdf);
            }
        }
    }
}
```

```

        else {
            printf('Flag to record add. res. could not be set for %o', comLdf);
        }
    }
    else {
        printf('Command %o could not be added for calculation', comLdf);
    }
}
}

```

SetIgnoreFlagForCommand

Set the flag whether to ignore a command for the calculation (using study case row and task row / using command).

```

int ComTasks.SetIgnoreFlagForCommand(int studyCaseRow,
                                     int taskRow,
                                     int ignoreVal
                                     )
int ComTasks.SetIgnoreFlagForCommand(object command,
                                     int ignoreVal
                                     )

```

ARGUMENTS

studyCaseRow> 0

The row in the study cases table of the command's study case for which the ignore flag shall be set.

taskRow> 0

The row in the commands table for which the ignore flag shall be set.

command

Command for which the ignore flag shall be set.

ignoreVal

1 Will set the command to be ignored for calculation.

0 Will set the command to be processed for calculation.

RETURNS

0 Ignore flag could not be set.

1 Ignore flag has been successfully set.

EXAMPLE

In this example, we set the command of study case row 1 and command row 1 to ignored for calculation.

```

int isIgnored;
object comTasks,
      casesFolder;
set commands;

casesFolder = GetProjectFolder('study');
commands = casesFolder.GetContents('*.*.ComTasks');
if (commands.Count()=1) {
    comTasks = commands.First();
}

```

```

if (comTasks<>nullptr) {
    isIgnored = SetIgnoreFlagForCommand(1, 1, 1);
    if (isIgnored<>0) {
        printf('Cmd in study case row 1, cmd row 1 is ignored');
    }
    else {
        printf('Ignore status could not be set');
    }
}
}

```

In this example, we add the load flow command to the calculation and set it to ignored afterwards.

```

int isAdded,
    isIgnored;
object comLdf,
    comTasks,
    casesFolder;
set commands;

casesFolder = GetProjectFolder('study');
commands = casesFolder.GetContents('*.*ComTasks');
if (commands.Count()=1) {
    comTasks = commands.First();
    comLdf = GetFromStudyCase('ComLdf');
    if ({comTasks<>nullptr}.and.{comLdf<>nullptr}) {
        isAdded = comTasks.AppendCommand(comLdf);
        if (isAdded<>0) {
            printf('Command %o has been added for calculation', comLdf);
            isIgnored = SetIgnoreFlagForCommand(comLdf, 1);
            if (isIgnored<>0) {
                printf('Command %o will be ignored for calculation', comLdf);
            }
            else {
                printf('Ignore status for command %o could not be set', comLdf);
            }
        }
        else {
            printf('Command %o could not be added for calculation', comLdf);
        }
    }
}

```

SetIgnoreFlagForStudyCase

Set the flag whether to ignore the processing of commands of a study case for the calculation.

```

int ComTasks.SetIgnoreFlagForStudyCase(object studyCase,
                                         int ignoreVal,
                                         [int studyCaseRow]
                                         )

```

RETURNS

- 0** Ignore flag could not be set.
- 1** Ignore flag has been successfully set.

ARGUMENTS

studyCase

Study case for which the ignore flag shall be set.

ignoreVal

- 1 Will set the flag to ignore the processing of commands of a study case.
- 0 Will set the flag to process the commands of a study case.

studyCaseRow

- ≤ 0 Ignore flag is set for all entries in the study cases table matching the provided study case.
- > 0 Optionally, the row in the study case table containing the study case for which the ignore-flag shall be set can be passed. This is helpful, e.g., if a study case has been added multiple times for calculation with different command lists.

SetResultsFolder

Set a folder to store results for a given row in the study case table.

```
int ComTasks.SetResultsFolder(object 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 results 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.

EXAMPLE

In this example, we set the active study case as new results folder for the study case in row 1 from the calculation.

```
int isSet;
object comTasks,
      casesFolder,
      activeCase;
set commands;

casesFolder = GetProjectFolder('study');
commands = casesFolder.GetContents('*.*ComTasks');
if (commands.Count()=1) {
    comTasks = commands.First();
    activeCase = GetActiveStudyCase();
    if ({comTasks->nullptr}.and{activeCase->nullptr}) {
        isSet = comTasks.SetResultsFolder(activeCase, 1);
        if ( isSet<>0 ) {
            printf('Results folder successfully changed.');
        }
    } else {
```

```
    printf('Results folder could not be changed');
}
}
}
```

3.4.49 ComTececo

Overview

UpdateTablesByCalcPeriod

UpdateTablesByCalcPeriod

Update all calculation points with respect to a new start- and end year

```
int ComTececo.UpdateTablesByCalcPeriod(double start,  
                                      double 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.

EXAMPLE

The following example configures a Techno-economical calculation with a study period 1.1.2015-31.12.2020 and executes it.

```
object tececoCmd;  
  
tececoCmd = GetFromStudyCase('ComTececo');  
if ( tececoCmd<>nullptr ) {  
    tececoCmd:e:CalcPoints = 0;                      ! yearly calculation points  
    tececoCmd.UpdateTablesByCalcPeriod(2015, 2020);   ! Set the study period to 1.1.2015-31.12.  
    tececoCmd.Execute();  
}
```

3.4.50 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.

```
void ComTransfer.GetTransferCalcData(double& transferCapacity, int& totalIterations)
```

ARGUMENTS

transferCapacity (out)
Transfer capacity value at the last feasible iteration.

totalIterations (out)
Total iteration number.

EXAMPLE

```
object cmdTransfer;
double dTotCapacity;
int iret, maxIteration;
cmdTransfer = GetFromStudyCase('ComTransfer');
if (cmdTransfer) {
    iret = cmdTransfer.Execute(); ! Performs transfer capacity calculation
    ! If the calculation was success then the data can be obtained
    if (iret == 0) {
        cmdTransfer.GetTransferCalcData(dTotCapacity, maxIteration);
        printf('Transfer capacity: %f MW, Total iteration: %d', dTotCapacity, maxIteration);
    }
}
```

IsLastIterationFeasible

The function verifies if the last transfer calculation iteration resulted in a feasible solution or not.

```
int ComTransfer.IsLastIterationFeasible()
```

RETURNS

- 1** Last transfer calculation iteration resulted in a feasible solution.
- 0** Last transfer calculation iteration did not result in a feasible solution.

EXAMPLE

```
object cmdTransfer;
int iret;
cmdTransfer = GetFromStudyCase('ComTransfer');
if (cmdTransfer) {
    iret = cmdTransfer.Execute(); ! Performs transfer capacity calculation
    ! If the calculation was success then the data can be obtained
    if (iret == 0) {
        iret = cmdTransfer.IsLastIterationFeasible();
        if (iret) {
            printf('Transfer capacity: the last iteration was feasible!');
        } else {
            printf('Transfer capacity: the last iteration was NOT feasible!');
        }
    }
}
```

3.4.51 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.

```
void 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. |

3.4.52 ComUcteexp

Overview

[BuildNodeNames](#)
[DeleteCompleteQuickAccess](#)
[ExportAndInitQuickAccess](#)
[GetConnectedBranches](#)
[GetFromToNodeNames](#)
[GetOrderCode](#)
[GetUcteNodeName](#)
[InitQuickAccess](#)
[QuickAccessAvailable](#)
[ResetQuickAccess](#)
[SetGridSelection](#)

BuildNodeNames

Builds the node names as used in UCTE export and makes them accessible via :UcteNodeName 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** On error (e.g. load flow calculation failed).

DEPRECATED NAMES**BuildExportStructure****EXAMPLE**

```

object command, term;
set terms;
int err,
    available,
    cacheIndex;

cacheIndex = 1;

!get ComUcteexp from active study case
command = GetFromStudyCase('ComUcteexp');

err = command.BuildNodeNames(cacheIndex);

if (err > 0) {
    Error('Error in determination of UCTE node names');
    exit();
}

!output node names
terms = GetCalcRelevantObjects('* ElmTerm');
for(term = terms.First(); term; term = terms.Next()) {
    printf('Terminal:%o UCTE Name: %s', term, term:UcteNodeName);
}

available = command.QuickAccessAvailable(cacheIndex);

printf('Quick access for cache index %d available: %d', cacheIndex, available);

```

DeleteCompleteQuickAccess

Deletes all quick access caches.

```
void ComUcteexp.DeleteCompleteQuickAccess()
```

EXAMPLE

```

object command;
int err, cacheIndex, available;

cacheIndex = 1;

!get ComUcteexp from active study case
command = GetFromStudyCase('ComUcteexp');

err = command.BuildNodeNames(cacheIndex);

if (err > 0) {
    Error('Error during filling quick access cache for cache index %d.', cacheIndex);
    exit();
}

```

```

}

available = command.QuickAccessAvailable(cacheIndex);
printf('Quick access for cache index %d available: %d', cacheIndex, available);

command.DeleteCompleteQuickAccess();

available = command.QuickAccessAvailable(cacheIndex);
printf('Quick access for cache index %d available: %d', cacheIndex, available);

```

ExportAndInitQuickAccess

Performs an UCTE export and fills the quick access cache given by the cache index.

```
void ComUcteexp.ExportAndInitQuickAccess(int cacheIndex)
```

ARGUMENTS

cacheIndex

Index of the quick access cache (must be greater than or equals to 0)

EXAMPLE

```

object command;
int err, cacheIndex, available;

cacheIndex = 1;

!get ComUcteexp from active study case
command = GetFromStudyCase('ComUcteexp');

err = command.ExportAndInitQuickAccess(cacheIndex);

if (err > 0) {
    Error('Error during filling quick access cache for cache index %d.', cacheIndex);
    exit();
}

available = command.QuickAccessAvailable(cacheIndex);

printf('Quick access for cache index %d available: %d', cacheIndex, available);

```

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.

```
void ComUcteexp.GetConnectedBranches(object terminal,
                                     set& connectedBranches,
                                     [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)

EXAMPLE

```
object command, term, branch;
set terms, branches;
int err, cacheIndex;

cacheIndex = 1;

!get ComUcteexp from active study case
command = GetFromStudyCase('ComUcteexp');

err = command.BuildNodeNames(cacheIndex);

if (err > 0) {
    Error('Error during filling quick access cache with index %d.', cacheIndex);
    exit();
}

!output node names
terms = GetCalcRelevantObjects('* ElmTerm');
for(term = terms.First(); term; term = terms.Next()) {
    command.GetConnectedBranches(term, branches, cacheIndex);
    printf('Terminal %o has the following connected branches:', term);

    for(branch = branches.First(); branch; branch = branches.Next()) {
        printf('%o', branch);
    }
}
```

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.

```
void ComUcteexp.GetFromToNodeNames(object branch,
                                    string& nodeNameFrom,
                                    string& nodeNameTo,
                                    [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)

EXAMPLE

```
object command, line;
set lines;
```

```

int err, cacheIndex;
string nodeFrom, nodeTo;

cacheIndex = 1;

!get ComUcteexp from active study case
command = GetFromStudyCase('ComUcteexp');

err = command.BuildNodeNames(cacheIndex);

if (err > 0) {
    Error('Error during filling quick access cache for cache index %d.', cacheIndex);
    exit();
}

!output node names
lines = GetCalcRelevantObjects('* ElmLne');
for(line = lines.First(); line; line = lines.Next()) {
    command.GetFromToNodeNames(line, nodeFrom, nodeTo, cacheIndex);
    printf('Line %o has start node %s and end node %s', line, nodeFrom, nodeTo);
}

```

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.

```

void ComUcteexp.GetOrderCode(object branch,
                            string& orderCode,
                            [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)

EXAMPLE

```

object command, line;
set lines;
int err, cacheIndex;
string orderCode;

cacheIndex = 1;

!get ComUcteexp from active study case
command = GetFromStudyCase('ComUcteexp');

err = command.BuildNodeNames(cacheIndex);

if (err > 0) {
    Error('Error during filling quick access cache for cache index %d.', cacheIndex);
    exit();
}

```

```

}

!output node names
lines = GetCalcRelevantObjects('*.ElmLne');
for(line = lines.First(); line; line = lines.Next()) {
    command.GetOrderCode(line, orderCode, cacheIndex);
    printf('Line %o has UCTE order code %s.', line, orderCode);
}

```

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.

```
void ComUcteexp.GetUcteNodeName(object terminal,
                                string& uctenodeName,
                                [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)

EXAMPLE

```

object command, term;
set terms;
int err, cacheIndex;
string nodeName;

cacheIndex = 1;

!get ComUcteexp from active study case
command = GetFromStudyCase('ComUcteexp');

err = command.BuildNodeNames(cacheIndex);

if (err > 0) {
    Error('Error during filling quick access cache for cache index %d.', cacheIndex);
    exit();
}

!output node names
terms = GetCalcRelevantObjects('*.ElmTerm');
for(term = terms.First(); term; term = terms.Next()) {
    command.GetUcteNodeName(term, nodeName, cacheIndex);
    printf('Terminal %o has UCTE node name %s.', term, nodeName);
}

```

InitQuickAccess

Initializes the quick access cache given by the optional cache index. The quick access 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.

```
void ComUcteexp.InitQuickAccess([int cacheIndex])
```

ARGUMENTS

cacheIndex (optional)

Index of the quick access cache (must be greater than or equals to 0)

EXAMPLE

```
object command;
int err, cacheIndex, available;

cacheIndex = 1;

!get ComUcteexp from active study case
command = GetFromStudyCase('ComUcteexp');

err = command.Execute();

if (err > 0) {
    Error('Error during filling quick access cache for cache index %d.', cacheIndex);
    exit();
}

command.InitQuickAccess(cacheIndex);
available = command.QuickAccessAvailable(cacheIndex);

printf('Quick access for cache index %d available: %d', cacheIndex, available);
```

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.

```
int ComUcteexp.QuickAccessAvailable([int cacheIndex])
```

ARGUMENTS

cacheIndex (optional)

Index of the quick access cache (must be greater than or equals to 0)

RETURNS

0 on success and 1 on error.

EXAMPLE

```
object command;
int err, available, cacheIndex;

cacheIndex = 1;

!get ComUcteexp from active study case
```

```

command = GetFromStudyCase('ComUcteexp');

err = command.BuildNodeNames(cacheIndex);

if (err > 0) {
    Error('Error during filling quick access cache for cache index %d.', cacheIndex);
    exit();
}

available = command.QuickAccessAvailable(cacheIndex);
printf('Quick access cache for cache index %d available: %d', cacheIndex, available);

```

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.

```
void 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.

```
void ComUcteexp.SetGridSelection(set gridsToExport)
```

ARGUMENTS

gridsToExport

Grids (instances of class ElmNet) to be selected for export. All not contained grids will be de-selected.

3.4.53 ComWktimp

Overview

[GetCreatedObjects](#)
[GetModifiedObjects](#)

GetCreatedObjects

Returns the newly created objects after execution of a WKT import.

```
set ComWktimp.GetCreatedObjects()
```

RETURNS

Collection of objects that have been created during WKT import.

EXAMPLE

The following example returns the created objects after execution of a WKT import:

```
set created;
object obj;

ImportCmd.Execute(); !execute wkt import

printf('Created objects:');
created = ImportCmd.GetCreatedObjects();
for(obj = created.First(); obj; obj = created.Next()) {
    printf('%o', obj);
}
```

GetModifiedObjects

Returns the modified objects after execution of a WKT import.

```
set ComWktimp.GetModifiedObjects()
```

RETURNS

Collection of objects that have been modified during WKT import.

EXAMPLE

The following example returns the modified objects after execution of a WKT import:

```
set modified;
object obj;

ImportCmd.Execute(); !execute wkt import

printf('\nModified objects:');
modified = ImportCmd.GetModifiedObjects();
for(obj = modified.First(); obj; obj = modified.Next()) {
    printf('%o', obj);
}
```

3.5 Settings

3.5.1 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.

EXAMPLE

See example of [SetCluster.GetNumberOfClusters](#).

GetNumberOfClusters

Get the number of clusters.

```
int SetCluster.GetNumberOfClusters()
```

RETURNS

The number of clusters.

EXAMPLE

The following example gets the SetCluster object from the reliability and prints the active power sum of loads for every load state to the output window.

```
object rel3,
       clusterObj,
       ldf,
       sumGrd;
set temp;
int nrOfClusters,
     clusterIndex,
     errorCode;
rel3 = GetFromStudyCase('ComRel3');
ldf = GetFromStudyCase('ComLdf');
temp = rel3.GetContents('*.' + SetCluster', 0);
clusterObj = temp.First();
if (clusterObj = nullptr) {
    exit();
}
ldf.Execute(); ! ldf result required by CalcCluster
sumGrd = GetSummaryGrid();

nrOfClusters = clusterObj.GetNumberOfClusters();
```

```
Info('Calculating %d clusters found in %o', nrOfClusters,clusterObj);
for (clusterIndex=0; clusterIndex<nrOfClusters; clusterIndex+=1) {
    errorCode = clusterObj.CalcCluster(clusterIndex,1);
    if (errorCode>0) {
        break;
    }
    printf('%f',sumGrd:m:Pload);
}
```

3.5.2 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 3.5.3

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 Contingency 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(string page,
                               int energizing,
                               [int alarm,]
                               [int normal]
                               )
```

ARGUMENTS

page

- empty** set for currently 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 3.5.4

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 Contingency 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
Hosting Capacity Analysis	hostcap
User defined calculation	usercalc

Table 3.5.5

Energizing State Name	"energizing" value
De-energized	33
Out of Calculation	37
De-energised, Planned Outage	61

Table 3.5.6

Alarm Name	"alarm" value
Voltage Violations / Overloading	29
Outages	31
Overloading of Thermal / Peak Short Circuit Current	32
Feeder Radiality Check	38

Table 3.5.7

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(object 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\(\)](#)

3.5.3 SetDataext

Overview

[AddConfiguration](#)
[GetConfiguration](#)
[GetConfigurations](#)
[RemoveAllConfigurations](#)
[RemoveConfiguration](#)

AddConfiguration

Adds a new IntAddonvars configuration object for the given classFilter with the descriptiveName as object name. For the classFilter expressions like Elm* or ElmTr? are possible. If there already is an object matching classFilter and descriptiveName exactly, this object is returned instead.

```
object SetDataext.AddConfiguration(string classFilter, string descriptiveName)
```

ARGUMENTS*classFilter*

The class filter of the IntAddonvars object

descriptiveName

The object name of the IntAddonvars object

RETURNS

The IntAddonvars object which exactly matches the classFilter and descriptive name or a newly created one.

SEE ALSO

[IntAddonvars.AddDouble\(\)](#), [IntAddonvars.AddDoubleMatrix\(\)](#), [IntAddonvars.AddDoubleVector\(\)](#),
[IntAddonvars.AddInteger\(\)](#), [IntAddonvars.AddIntegerVector\(\)](#), [IntAddonvars.AddObject\(\)](#), [IntAddonvars.AddObjectVector\(\)](#), [IntAddonvars.AddString\(\)](#), [IntAddonvars.RemoveParameter\(\)](#)

GetConfiguration

Returns the IntAddonvars object which exactly matches the classFilter and descriptiveName, if the latter is specified. If there are multiple matches the first object will be returned. Otherwise nothing is returned.

```
object SetDataext.GetConfiguration(string classFilter, [string descriptiveName])
```

ARGUMENTS*classFilter*

The class filter of the IntAddonvars object

descriptiveName (optional)

The object name of the IntAddonvars object

RETURNS

The IntAddonvars object which exactly matches the classFilter and optionally the descriptiveName or nothing.

SEE ALSO

[IntAddonvars.AddDouble\(\)](#), [IntAddonvars.AddDoubleMatrix\(\)](#), [IntAddonvars.AddDoubleVector\(\)](#),
[IntAddonvars.AddInteger\(\)](#), [IntAddonvars.AddIntegerVector\(\)](#), [IntAddonvars.AddObject\(\)](#), [IntAddonvars.AddObjectVector\(\)](#), [IntAddonvars.AddString\(\)](#), [IntAddonvars.RemoveParameter\(\)](#)

GetConfigurations

Returns all IntAddonvars objects by a given classFilter.

```
set SetDataext.GetConfigurations(string classFilter)
set SetDataext.GetConfigurations()
```

ARGUMENTS*classFilter*

The class filter for the IntAddonvars object

RETURNS

A list of IntAddonvars objects matching the classFilter exactly. If no filter is specified all IntAddonvars are returned.

SEE ALSO

[IntAddonvars.AddDouble\(\)](#), [IntAddonvars.AddDoubleMatrix\(\)](#), [IntAddonvars.AddDoubleVector\(\)](#),
[IntAddonvars.AddInteger\(\)](#), [IntAddonvars.AddIntegerVector\(\)](#), [IntAddonvars.AddObject\(\)](#), [In-](#)
[tAddonvars.AddObjectVector\(\)](#), [IntAddonvars.AddString\(\)](#), [IntAddonvars.RemoveParameter\(\)](#)

RemoveAllConfigurations

Removes all IntAddonvars objects effectively removing all Data Extensions.

```
void SetDataext.RemoveAllConfigurations()
```

RemoveConfiguration

Removes the IntAddonvars object exactly matching classFilter and descriptive name.

```
void SetDataext.RemoveConfiguration(string classFilter, [string descriptiveName])
```

ARGUMENTS

classFilter

The class filter of the IntAddonvars object

descriptiveName (optional)

The object name of the IntAddonvars object

3.5.4 SetDeskpage

Overview

[Close](#)

[Show](#)

Close

Closes the graphic page, if currently shown in the graphics board.

```
int SetDeskpage.Close()
```

RETURNS

0 On success, no error occurred.

1 Otherwise

Show

Displays the diagram page in the graphics board.

```
int SetDeskpage.Show()
```

RETURNS

0 On success, no error occurred.

1 Otherwise

3.5.5 SetDesktop

Overview

AddPage
 Close
 DoAutoSizeX
 Freeze
 GetActivePage
 GetCanvasSize
 GetPage
 IsFrozen
 IsOpened
 RemovePage
 SetAdaptX
 SetAutoSizeX
 SetResults
 SetScaleX
 SetXVar
 Show
 Unfreeze
 WriteWMF
 ZoomAll

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.

```
object SetDesktop.AddPage(object page2add)
```

ARGUMENTS

page2add

The page to add to the desktop.

- Page is a plot page (GrpPage or SetVipage): 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 NULL if the desktop was not changed.

EXAMPLE

```
object desktop; ! graphic board
object pagecopied; ! page created by AddPage
desktop = GetGraphBoard();
if (desktop) {
    ! add PageTemplate to desktop (it is assumed that PageTemplate is a
    ! virtual instrument panel stored in the "Contents" of the DPL script.
    pagecopied = desktop.AddPage(PageTemplate);
    pagecopied.ShowFullName();
}
```

Close

Closes the graphics board, if it is currently shown.

```
int SetDesktop.Close()
```

RETURNS

- 0 on success
- 1 on error

DoAutoScaleX

Scales the x-axes of all plots in the graphics board which use the x-axis scale defined in the graphics board.

```
void SetDesktop.DoAutoScaleX()
```

EXAMPLE

The following example performs an automatic scaling of x.

```
object graphBoard;  
! Look for opened graphics board.  
graphBoard=GetGraphBoard();  
if (graphBoard) {  
    graphBoard.DoAutoScaleX();  
}
```

Freeze

Enables the graphical freeze mode, preventing changes to open diagrams.

```
int SetDesktop.Freeze()
```

RETURNS

- 0 on success
- 1 on error

GetActivePage

Returns the page object of the currently shown page.

```
object SetDesktop.GetActivePage()
```

RETURNS

Page object of the active page, or 0 if no page is active.

GetCanvasSize

Returns the pixel dimensions of the currently active (top-most) graphic page.

```
int SetDesktop.GetCanvasSize(  
                            int& width,  
                            int& height)
```

ARGUMENTS

width (out)

Pixel width of the canvas. -1 if no graphics page is open.

height (out)

Pixel height of the canvas. -1 if no graphics page is open.

RETURNS

- 0** on success: canvas size could be determined
- 1** on error: no graphics page is open

GetPage

Searches, activates and returns a graphics page in the currently open graphics board. If “create” is true, then a new page will be created and added to the graphics board if no page with name was found.

```
object SetDesktop.GetPage(string name,
                           [int create,]
                           [string class]
                           )
```

ARGUMENTS

name Name of the page.

create (optional)

Possible values:

- 0** do not create new plot page
- 1** create plot page if it does not exist already

class (optional)

Classname of the plot page object to create: either 'GrpPage' or 'SetVipage'. If not specified, a 'GrpPage' will be created if the new plot framework is enabled in the project settings, otherwise a 'SetVipage' will be created.

RETURNS

Plot page (GrpPage or SetVipage), or network graphic page (SetDeskpage)

EXAMPLE

The following example looks for the plot pages named 'Voltage', 'Current' and 'Power' in the graphics board currently open. The plot pages are created if they do not exist.

```
object graphBoard;
object pageVoltage;
object pageCurrent;
object pagePower;
! Look for opened graphics board.
graphBoard=GetGraphBoard();
if (graphBoard) {
    ! Search or create plot pages
    pageVoltage=graphBoard.GetPage('Voltage',1);
    pageCurrent=graphBoard.GetPage('Current',1);
    pagePower=graphBoard.GetPage('Power',1);
}
```

IsFrozen

Returns whether the graphical freeze mode is currently enabled.

```
int SetDesktop.IsFrozen()
```

RETURNS

- 0** freeze mode is disabled, or the graphics board is not open
- 1** graphics board is open and freeze mode is enabled

IsOpened

Returns whether the graphics board is currently shown.

```
int SetDesktop.IsOpened()
```

RETURNS

- 1** if graphics board is shown
- 0** otherwise

RemovePage

Removes a graphic page. The page to be removed can be identified either by its name or by its page object.

```
int SetDesktop.RemovePage(string pageName)
int SetDesktop.RemovePage(object pageObject)
```

ARGUMENTS

pageName

Name of graphics page.

pageObject

A graphics page object.

RETURNS

- 0** on success
- 1** on error

SetAdaptX

Sets the Adapt Scale option of the x-scale.

```
void SetDesktop.SetAdaptX(int mode,
                           [double trigger]
                           )
```

ARGUMENTS

mode Possible values:

- 0** off
- 1** on

trigger (optional)

Trigger value, unused if mode is off or empty

EXAMPLE

The following example looks for an opened graphics board and sets its adapt scale option.

```
object graphBoard;
graphBoard=GetGraphBoard(); ! Look for open graphics board.
if (graphBoard) {
    ! Turn on adapt scale, use a trigger value of 3
    graphBoard.SetAdaptX(1,3);
    ! Turn off adapt scale
    graphBoard.SetAdaptX(0,3);
    ! Turn on adapt scale again, do not change the trigger value
    graphBoard.SetAdaptX(1);
}
```

SetAutoScaleX

Sets automatic scaling mode of the x-scale. A warning is issued if an invalid mode is passed to the function.

```
void SetDesktop.SetAutoScaleX(int mode)
```

ARGUMENTS

mode Possible values:

- 0** never
- 1** after simulation
- 2** during simulation

EXAMPLE

The following example looks for an open graphics boards and sets its Auto Scale setting to Off.

```
! Set Auto Scale to Off
object graphBoard;
graphBoard=GetGraphBoard(); ! Look for open graphics board.
if (graphBoard) {
    ! Turn off automatic scaling
    graphBoard.SetAutoScaleX(0);
}
```

SetResults

Sets default results object of graphics board.

```
void SetDesktop.SetResults(object res)
```

ARGUMENTS

res Result object to set or NULL to reset. Valid result object is any of class ElmRes, IntComtrade and IntComtradeset.

EXAMPLE

The following example looks for an open graphics board and sets its default results to the results object named 'Results'.

```
object graphBoard;
graphBoard=GetGraphBoard(); ! Look for open graphics board.
if (graphBoard) {
    graphBoard.SetResults(Results);
}
```

SetScaleX

Sets x-axis scale. A function call without arguments sets the Auto Scale setting to On without changing the scale itself.

```
void SetDesktop.SetScaleX()
void SetDesktop.SetScaleX(double min,
                        double max,
                        [int log]
)
```

ARGUMENTS

min (optional)

Minimum of x-scale.

max (optional)

Maximum of x-scale.

log (optional)

Possible values:

0	linear
1	logarithmic

EXAMPLE

The following examples look for an open graphics board and set its x-axis scale. There are three different examples. 1. Example: Set 'Auto Scale' of x axis to 'On'. 2. Example: Set minimum to 0 and maximum to 20. 3. Example: Set minimum to 1 and maximum to 1000. Changes to a logarithmic scale.

```
object graphBoard;
graphBoard=GetGraphBoard(); ! Look for open graphics board.
if (graphBoard) {
    graphBoard.SetScaleX(); ! Automatic Scaling to On
}
! Set minimum and maximum without changing linear/log
object graphBoard;
graphBoard=GetGraphBoard(); ! Look for open graphics board.
if (graphBoard) {
    ! Set minimum and maximum
    graphBoard.SetScaleX(0,20);
}
! Set minimum and maximum, change to logarithmic scale
object graphBoard;
graphBoard=GetGraphBoard(); ! Look for open graphics board.
if (graphBoard) {
```

```

    ! Set minimum and maximum
    graphBoard.setScaleX(1,1000,1);
}

```

SetXVar

Sets x-axis variable. If The default x-axis variable (time) is set if no argument is passed.

```

void SetDesktop.SetXVar()
void SetDesktop.SetXVar(object obj,]
                      string varname
                      )

```

ARGUMENTS

obj (*optional*)
x-axis object

varname (*optional*)
variable of *obj*

EXAMPLE

The following examples look for an open graphics board and sets its x-axis variable. The first example sets an user defined x-axis variable of the graphics board. The second one sets the default x-axis (in simulation: time).

```

! set x-axis variable 'm:U1:bus1' of object line
object graphBoard;
graphBoard=GetGraphBoard(); ! Look for open desktop.
if (graphBoard) {
    graphBoard.SetXVar(line, 'm:U1:bus1');
}

! set default x-axis variable
object graphBoard;
graphBoard=GetGraphBoard(); ! Look for open desktop.
if (graphBoard) {
    ! Set default x-axis variable (time)
    graphBoard.SetXVar();
}

```

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(string pageName)
int SetDesktop.Show(object pageObject)

```

ARGUMENTS

pageName (*optional*)
Name of graphics page.

pageObject (optional)

A graphics page object.

RETURNS

0 on success

1 on error

EXAMPLE

The following example activates all pages in the graphics board one by one and exports them as WMF figures.

```
object graphicsBoard,page;
set pages;
int n;
string FileName;
graphicsBoard = GetGraphBoard();
if (graphicsBoard) {
    pages = graphicsBoard.GetContents();
    page = pages.First();
    while (page) {
        graphicsBoard.Show(page);
        FileName = sprintf('c:\\temp\\%s%d', page:loc_name, n);
        graphicsBoard.WriteWMF(FileName);
        page = pages.Next();
    }
}
```

Unfreeze

Disables the graphical freeze mode, allowing changes to open diagrams.

```
int SetDesktop.Unfreeze()
```

RETURNS

0 on success

1 on error

WriteWMF

Writes the currently open graphic to a windows metafile file (*.wmf).

Please use the Save File command (ComWr) for writing to other formats like *.pdf, *.png, *.svg, *.emf or *.bmp.

```
int SetDesktop.WriteWMF(string filename)
```

ARGUMENTS

filename Filename without extension.

RETURNS

0 On error.

1 On success.

EXAMPLE

The following example exports the open graphic to a windows metafile.

```
object graphicsBoard;
graphicsBoard = GetGraphBoard();
if (graphicsBoard) {
    graphicsBoard.WriteWMF('c:\temp\DPLEExample');
```

ZoomAll

Adjusts the zoom level of the currently active (top-most) graphic page such that the entire diagram is shown.

```
int SetDesktop.ZoomAll()
```

RETURNS

- 0** on success
- 1** on error

3.5.6 SetDistrstate**Overview**

[CalcCluster](#)

CalcCluster

Calculates a load flow with a given load distribution state applied.

```
int SetDistrstate.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 if ok. -1 if load flow of cluster did not converge.

3.5.7 SetFilt

Overview

[Get*](#)

Get*

Returns a container with the filtered objects.

```
set SetFilt.Get()
```

RETURNS

The set of filtered objects.

3.5.8 SetLevelvis

Overview

[AdaptWidth](#)

[Align](#)

[ChangeFont](#)

[ChangeFrameAndWidth](#)

[ChangeLayer](#)

[ChangeRefPoints](#)

[ChangeWidthVisibilityAndColour](#)

[Mark](#)

[Reset](#)

AdaptWidth

This function resizes the in the object specified group of text boxes regarding their text contents.

```
void SetLevelvis.AdaptWidth()
```

Align

This function aligns the text within a text box.

```
void 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.

```
void 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.

```
void SetLevelvis.ChangeFrameAndWidth([int iFrame,]
                                      [int iWidth,]
                                      [int iVisibility,]
                                      [int iColour])
                                         )
```

ChangeLayer

This function sets the specified group of text boxes to a given layer.

```
void SetLevelvis.ChangeLayer(string 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

```
void 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.

```
void 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.

```
void SetLevelvis.Mark()
```

Reset

This function resets the individually modified text box settings.

```
void 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')

3.5.9 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.

3.5.10 SetPath

Overview

[AllBreakers*](#)
[AllClosedBreakers*](#)
[AllOpenBreakers*](#)
[AllProtectionDevices*](#)
[Create](#)
[GetAll*](#)
[GetBranches*](#)
[GetBuses*](#)
[GetPathFolder](#)

AllBreakers*

Returns all breakers in the path definition.

```
set SetPath.AllBreakers()
```

RETURNS

The set of breakers.

AllClosedBreakers*

Returns all closed breakers in the path definition.

```
set SetPath.AllClosedBreakers()
```

RETURNS

The set of closed breakers.

AllOpenBreakers*

Returns all open breakers in the path definition.

```
set SetPath.AllOpenBreakers()
```

RETURNS

The set of open breakers.

AllProtectionDevices*

Returns all protection devices in the path definition for a given direction.

```
set SetPath.AllProtectionDevices(int reverse)
```

ARGUMENTS

reverse

- 0** Return devices in forward direction.
- 1** Return devices in reverse direction.

RETURNS

The set of protection devices.

Create

Creates or extends the path with the elements provided.

```
object SetPath.Create(set elements)
```

ARGUMENTS

elements Elements the path shall be created or extended with.

RETURNS

object Modification was successful.

NULL Modification failed. (e.g. elements form an incomplete path)

EXAMPLE

The following emulates *Path* → *New* from the context menu, by creating a new path definition with the contents of the general selection selection of the script.

```
object folder, path, return;

folder = GetDataFolder('IntPath');
path = folder.CreateObject('SetPath', 'Path');
return = path.Create(SEL.All());
if (return) {
    printf('Path %o was modified', path);
}
else {
    printf('An error occurred while modifying path %o', path);
}
```

GetAll*

Returns all objects in the path definition.

```
set SetPath.GetAll()
```

RETURNS

The set of objects.

EXAMPLE

The following example writes all objects in the path definition to the output window, assuming *aPath* is an external *SetPath* object.

```
set list;
object obj;
list = aPath.GetAll();
obj = list.First();
while (obj) {
    obj.ShowFullName();
    obj = list.Next();
}
```

GetBranches*

Returns all branches in the path definition.

```
set SetPath.GetBranches([int reverse])
```

ARGUMENTS

reverse (*optional*)

- 0** Sort the branches in forward direction.
- 1** Sort the branches in reverse direction.

RETURNS

The set of branches.

GetBuses*

Returns all busbars and terminals in the path definition.

```
set SetPath.GetBuses()
```

RETURNS

The set of busbars and terminals.

GetPathFolder

Returns the default folder for storing path objects.

```
object SetPath.GetPathFolder([int create])
```

ARGUMENTS

create (optional)

- 0 Return only if the folder exists.
- 1 Create the folder if it does not exist.

RETURNS

object Default folder for storing path.

NULL Default folder does not exist or could not be created.

3.5.11 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.

```
void SetSelect.AddRef(object O)
void SetSelect.AddRef(set S)
```

ARGUMENTS

O An object.

S A set of objects.

EXAMPLE

The following example adds all loads and lines from the general DPL selection to the selection “MySelection”.

```
set lines;
lines = SEL.AllLines();
MySelection.AddRef(lines);
lines = SEL.AllLoads();
MySelection.AddRef(lines);
```

All*

Returns all objects in the selection.

```
set SetSelect.All()
```

RETURNS

The set of objects

EXAMPLE

The following example writes objects in the general DPL selection to the output window.

```
set list;
object obj;
list = SEL.All();
obj = list.First();
while (obj) {
    obj.ShowFullName();
    obj = list.Next();
}
```

AllAsm*

Returns all asynchronous machines in the selection.

```
set SetSelect.AllAsm()
```

RETURNS

The set of objects

EXAMPLE

The following example writes all asynchronous machines in the general DPL selection to the output window.

```
set list;
object asynMachines;
list = SEL.AllAsm();
asynMachines = list.First();
while (asynMachines) {
    asynMachines.ShowFullName();
    asynMachines = list.Next();
}
```

AllBars*

Returns all busbars and terminals in the selection.

```
set SetSelect.AllBars()
```

RETURNS

The set of objects

EXAMPLE

The following example writes all busbars in the general DPL selection to the output window.

```
set list;
object bar;
list = SEL.AllBars();
bar = list.First();
while (bar) {
    bar.ShowFullName();
    bar = list.Next();
}
```

AllBreakers*

Returns all breakers in the selection.

```
set SetSelect.AllBreakers()
```

RETURNS

The set of objects

EXAMPLE

The following example writes all breakers in the general DPL selection to the output window.

```
set list;
object breaker;
list = SEL.AllBreakers();
breaker = list.First();
while (breaker) {
    breaker.ShowFullName();
    breaker = list.Next();
}
```

AllClosedBreakers*

Returns all closed breakers in the selection.

```
set SetSelect.AllClosedBreakers()
```

RETURNS

The set of objects

EXAMPLE

The following example writes all closed breakers in the general DPL selection to the output window.

```
set list;
object closedbreaker;
list = SEL.AllClosedBreakers();
closedbreaker = list.First();
while (closedbreaker) {
    closedbreaker.ShowFullName();
    closedbreaker = list.Next();
}
```

AllElm*

Returns all elements (Elm*) in the selection.

```
set SetSelect.AllElm()
```

RETURNS

The set of containing objects

EXAMPLE

The following example writes all objects in the general DPL selection to the output window.

```
set list;
object obj;
list = SEL.AllElm();
obj = list.First();
while (obj) {
    obj.ShowFullName();
    obj = list.Next();
}
```

AllLines*

Returns all lines and line routes in the selection.

```
set SetSelect.AllLines()
```

RETURNS

The set of objects

EXAMPLE

The following example writes all lines and line routes in the general DPL selection to the output window.

```
set list;
object line;
list = SEL.AllLines();
line = list.First();
while (line) {
    line.ShowFullName();
    line = list.Next();
}
```

AllLoads*

Returns all loads in the selection.

```
set SetSelect.AllLoads()
```

RETURNS

The set of objects

EXAMPLE

The following example writes all loads in the general DPL selection to the output window.

```
set list;
object load;
list = SEL.AllLoads();
load = list.First();
while (load) {
    load.ShowFullName();
    load = list.Next();
}
```

AllOpenBreakers*

Returns all open breakers in the selection.

```
set SetSelect.AllOpenBreakers()
```

RETURNS

The set of objects

EXAMPLE

The following example writes all open breakers in the general DPL selection to the output window.

```
set list;
object openbreaker;
list = SEL.AllOpenBreakers();
openbreaker = list.First();
while (openbreaker) {
    openbreaker.ShowFullName();
    openbreaker = list.Next();
}
```

AllSym*

Returns all synchronous machines in the selection.

```
set SetSelect.AllSym()
```

RETURNS

The set of objects

EXAMPLE

The following example writes all synchronous machines in the general DPL selection to the output window.

```
set list;
object synMachines;
list = SEL.AllSym();
synMachines = list.First();
while (synMachines) {
    synMachines.ShowFullName();
    synMachines = list.Next();
}
```

AllTypLne*

Returns all line types in the selection.

```
set SetSelect.AllTypLne()
```

RETURNS

The set of objects

EXAMPLE

The following example writes objects in the general DPL selection to the output window.

```
set list;
object obj;
list = SEL.All();
obj = list.First();
while (obj) {
    obj.ShowFullName();
    obj = list.Next();
}
```

Clear

Clears (deletes) the selection.

```
void SetSelect.Clear()
```

EXAMPLE

The following example clears the set from the previous content, then adds all loads in the general DPL selection.

```
set loads;
loads = SEL.AllLoads();
MySelection.Clear();
MySelection.AddRef(loads);
```

GetAll*

Returns all objects in the selection which are of the class 'ClassName'.

```
set SetSelect.GetAll(string ClassName)
```

ARGUMENTS

ClassName

The object class name.

RETURNS

The set of objects

EXAMPLE

The following example writes all three winding transformers in the general DPL selection to the output window.

```
set list;
object transformer;
list = SEL.GetAll('ElmTr3');
transformer = list.First();
while (transformer) {
    transformer.ShowFullName();
    transformer = list.Next();
}
```

3.5.12 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.

```
string 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.

```
string 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(string 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.

3.5.13 SetTime

Overview

[Date](#)
[SetTime](#)
[SetTimeUTC](#)
[Time](#)

Date

Sets date component to current system date.

```
void SetTime.Date()
```

EXAMPLE

The following example sets the study time of active project to current date:

```
object studytime;
studytime = GetFromStudyCase('SetTime');
studytime.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.

```
void SetTime.SetTime(double H,
                     [double M,]
                     [double S]
                     )
```

ARGUMENTS

H The hours

M (optional)
The minutes

S (optional)
The seconds

EXAMPLE

The following sets the time to 1134.45 hours, 91.2 minutes and 675.3 seconds, which results in the time 08:09:27 on the 48th day of the year.

```
object studytime;
studytime = GetFromStudyCase('SetTime');
studytime.SetTime(1134.45, 91.2, 675.3);
```

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.

```
void SetTime.SetTimeUTC(int time)
```

ARGUMENTS

time UTC time in seconds since 01.01.1970 00:00 GMT

EXAMPLE

Example demonstrates how to change date and time of active study case:

```
object studytime;
studytime = GetFromStudyCase('SetTime');
studytime.SetTimeUTC(1200478788); !Wed, 16 Jan 2008 10:19:48 GMT
```

SEE ALSO

[SetTime.Date\(\)](#), [SetTime.Time\(\)](#)

Time

Sets time component to current system time.

```
void SetTime.Time()
```

EXAMPLE

The following example sets the study time of active project to current time:

```
object studytime;
studytime = GetFromStudyCase('SetTime');
studytime.Time();
```

SEE ALSO

[SetTime.Date\(\)](#), [SetTime.SetTimeUTC\(\)](#)

3.5.14 SetUser**Overview**

[GetNumProcesses*](#)

GetNumProcesses*

This function returns the actual number of processes for parallel computation.

```
int SetUser.GetNumProcesses()
```

RETURNS

The actual number of processes for parallel computation.

EXAMPLE

The following example obtains the number of processes used by Parallel Contingency Analysis.

```
object oComSimoutage;
object oUserSetting;
int numProcesses;
oComSimoutage = GetCaseObject('ComSimOutage');
oUserSetting = oComSimoutage:e:parallelSetting;

if (oUserSetting = NULL) {
```

```

        Error('No user setting object found.');
        exit();
    }
numProcesses = oUserSetting.GetNumProcesses();
Info('%d processes will be used in parallel computation.', numProcesses);

```

3.5.15 SetVipage

Overview

[Close](#)
[DoAutoSizeX](#)
[DoAutoSizeY](#)
[GetOrInsertPlot](#)
[InsertPlot](#)
[MigratePage](#)
[SetAdaptX](#)
[SetAutoSizeX](#)
[SetResults](#)
[SetScaleX](#)
[SetStyle](#)
[SetTile](#)
[SetXVar](#)
[Show](#)

Close

Closes the graphic page, if currently shown, and deletes it from the database.

```
int SetVipage.Close()
```

RETURNS

- 0** On success, no error occurred.
- 1** Otherwise

DoAutoSizeX

Scales the x-axes of all plots on the virtual instrument panel automatically.

```
void SetVipage.DoAutoSizeX()
```

EXAMPLE

The following example looks for a page named voltage and performs an automatic scaling of the x-axes.

```

object page;
object graphBoard;
! Look for opened graphics board.
graphBoard=GetGraphBoard();
if (graphBoard) {
    page=graphBoard.GetPage('Voltage',1);
    if (page) {
        page.DoAutoSizeX();
    }
}

```

```

    }
}
```

DoAutoScaleY

Scales the y-axes of all plots on the virtual instrument panel automatically.

```
void SetVipage.DoAutoScaleY()
```

EXAMPLE

The following example looks for a page named voltage and performs an automatic scaling of the y-axes.

```

object page;
object graphBoard;
graphBoard=GetGraphBoard();
if (graphBoard) {
    page=graphBoard.GetPage('Voltage',1);
    if (page) {
        page.DoAutoScaleY();
    }
}
```

GetOrInsertPlot

Get or create a virtual instrument of the virtual instrument panel.

```
object SetVipage.GetOrInsertPlot (string name,
                                [string class,]
                                [int create]
                                )
```

DEPRECATED NAMES

GetVI

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

EXAMPLE

The following example looks for a subplot (VisPlot) named RST on a virtual instrument panel. The plot is created if it was not found.

```
object graphBoard;
object page;
object plot;
graphBoard=GetGraphBoard(); ! Look for open desktop
if (graphBoard) {
    ! Get Virtual Instrument Panel named 'Voltage'
    page=graphBoard.GetPage('Voltage',1);
    if (page) {
        ! Get plot named RST. Create the plot if not exists
        plot=page.GetOrInsertPlot('RST','VisPlot',1);
    }
}
```

InsertPlot

Creates a copy of the virtual instrument passed and displays the copy on this panel.

```
object SetVipage.InsertPlot(object vi)
```

DEPRECATED NAMES

CreateVI

ARGUMENTS

vi The virtual instrument which will be copied. Only virtual instruments are allowed (classname 'Vis*').

RETURNS

Returns the created virtual instrument.

EXAMPLE

The following example creates a copy of the plot stored inside the script on page 'Page 1'.

```
object desktop; ! graphics board
object page;      ! page
object newPlot;   ! plot created by InsertPlot

desktop = GetGraphBoard();
page = desktop.GetPage('Page 1',1); ! get panel, create if not there
newPlot = page.InsertPlot(plot);     ! copy of the plot stored inside ComDpl
newPlot.ShowFullName();            ! report full name of the created plot
```

MigratePage

Converts this SetVipage to the new plot framework introduced in PowerFactory 2021, creating a GrpPage:

- The original SetVipage will remain unchanged
- The created GrpPage will initially be hidden. Use GrpPage.Show() to make it visible.

Please note that currently (PowerFactory 2021) only plots of type VisPlot, VisPlot2, and VisXy-plot can be migrated. All other plot types will be missing in the migrated GrpPage.

```
void SetVipage.MigratePage()
```

RETURNS

The migrated plot page (GrpPage)

EXAMPLE

The following example looks for a SetVipage named 'Plot Generators', migrates it, and makes the migrated GrpPage visible.

```
object graphicsBoard;
object setVipage, grpPage;
graphicsBoard = GetGraphicsBoard();
if (graphicsBoard) {
    setVipage = graphicsBoard.GetPage('Plot Generators', 0);
    if (setVipage) {
        grpPage = setVipage.MigratePage();
        grpPage.Show();
    }
}
```

SetAdaptX

Sets the Adapt Scale option of the x-scale.

```
void SetVipage.SetAdaptX(int mode,
                           [double trigger]
                           )
```

ARGUMENTS

mode Possible values:

0	off
1	on

trigger (optional)

Trigger value, unused if mode is off or empty

EXAMPLE

The following examples look for a virtual instrument panel named 'Voltage' and sets its adapt scale option.

```
object page;
object graphBoard;
graphBoard=GetGraphBoard(); ! Look for open graphics board.
if (graphBoard) {
    page=graphBoard.GetPage('Voltage',1);! Get panel named 'Voltage'
    if (page) {
        ! Turn on adapt scale, use a trigger value of 3
        page.SetAdaptX(1,3);
        ! Turn off adapt scale
        page.SetAdaptX(0,3);
        ! Turn on adapt scale again, do not change the trigger value
        page.SetAdaptX(1);
    }
}
```

SetAutoScaleX

Sets automatic scaling mode of the x-scale. A warning is issued if an invalid mode is passed to the function.

```
void SetVipage.SetAutoScaleX(int mode)
```

ARGUMENTS

mode Possible values:

- 0** never
- 1** after simulation
- 2** during simulation

EXAMPLE

The following examples look for a virtual instrument panel named 'Voltage' and change its Auto Scale setting to Off.

```
! Set Auto Scale to Off
object page;
object graphBoard;
graphBoard=GetGraphBoard(); ! Look for open graphics board.
if (graphBoard) {
    page=graphBoard.GetPage('Voltage',1); ! Get panel named 'Voltage'
    if (page) {
        ! Set new limits to change x-scale of page to used scale
        page.setScaleX(0,10);
        page.SetAutoScaleX(0); ! Turn off automatic scaling
    }
}
```

SetResults

Sets default results object of virtual instrument panel.

```
void SetVipage.SetResults(object res)
```

ARGUMENTS

res Result object to set or NULL to reset. Valid result object is any of class ElmRes, IntComtrade and IntComtradeset.

EXAMPLE

The following example looks for a virtual instrument panel named 'Voltage' and resets its default results.

```
object graphBoard;
object page;
! Look for open graphics board.
graphBoard=GetGraphBoard();
if (graphBoard) {
    ! Get Virtual Instrument Panel named Voltage
    page=graphBoard.GetPage('Voltage',1);
```

```

if (page) {
    page.SetResults(nullptr); ! reset default results
}
}

```

SetScaleX

Sets x-axis scale. A function call without arguments sets the Auto Scale setting to On without changing the scale itself.

```

void SetVipage.SetScaleX()
void SetVipage.SetScaleX(double min,
                        double max,
                        [int log]
)

```

ARGUMENTS

min (optional)

Minimum of x-scale.

max (optional)

Maximum of x-scale.

log (optional)

Possible values:

0 linear

1 logarithmic

EXAMPLE

The following examples look for a virtual instrument panel named 'Voltage' and sets the x-axis variable. There are three different examples. 1. Example: Set 'Auto Scale' of x axis to 'On'. 2. Example: Set minimum to 0 and maximum to 20. 3. Example: Set minimum to 1 and maximum to 1000. Changes to a logarithmic scale.

```

object graphBoard;
object page;
graphBoard=GetGraphBoard(); ! Look for open graphics board.
if (graphBoard) {
    ! Get Virtual Instrument Panel named Voltage
    page=graphBoard.GetPage('Voltage',1);
    if (page) {
        page.SetScaleX(); ! Automatic Scaling to On
    }
}

! Set minimum and maximum without changing linear/log
object graphBoard;
object page;
graphBoard=GetGraphBoard(); ! Look for open graphics board.
if (graphBoard) {
    ! Get Virtual Instrument Panel named Voltage
    page=graphBoard.GetPage('Voltage',1);
    if (page) {
        ! Set minimum and maximum
        page.SetScaleX(0,20);
    }
}

```

```

}

! Set minimum and maximum, change to logarithmic scale
object graphBoard;
object page;
graphBoard=GetGraphBoard(); ! Look for open graphics board.
if (graphBoard) {
    ! Get Virtual Instrument Panel named Voltage
    page=graphBoard.GetPage('Voltage',1);
    if (page) {
        ! Set minimum and maximum, change to log. scale
        page.SetScaleX(1,1000,1);
    }
}

```

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.

```
void SetVipage.setStyle(string name)
```

ARGUMENTS

name Style Name

EXAMPLE

The following example looks for a virtual instrument panel named 'Voltage' and sets its style to 'Paper'.

```

object desktop;
object page;
desktop=GetGraphBoard(); ! get SetDesktop currently being open
if (desktop) {
    page=desktop.GetPage('Voltage',1); ! get panel named 'Voltage'.
    if (page) {
        page.setStyle('Paper'); ! Set style of panel to the one named 'Paper'
    }
}

```

SetTitle

Rearranges the virtual instrument on the panel.

```
void SetVipage.setTitle([int tile])
```

ARGUMENTS

tile=1 (optional) **tile =0** arrange virtual instruments automatically (like tiles)
tile=1 arrange them horizontally

EXAMPLE

The following example looks for a virtual instrument panel named 'Voltage' and rearranges the Virtual Instrument.

```

object graphBoard;
object page;
graphBoard=GetGraphBoard(); ! Look for open desktop.
if (graphBoard) {
    ! Get Virtual Instrument Panel named 'Voltage'
    page=graphBoard.GetPage('Voltage',1);
    if (page) {
        ! Arrange vis horizontally (default input parameter is 1)
        page.SetTile();
    }
}

```

SetXVar

Sets x-axis variable. If The default x-axis variable (time) is set if no argument is passed.

```

void SetVipage.SetXVar()
void SetVipage.SetXVar(object obj,
                      string varname
                     )

```

ARGUMENTS

obj (optional)
x-axis object

varname (optional)
variable of obj

EXAMPLE

The following examples look for a virtual instrument panel named 'Voltage' and set the x-axis variable. The first example sets an user defined x-axis variable of the panel. The second one sets the default x-axis (in simulation: time).

```

! set x-axis variable 'm:U1:bus1' of object line
object graphBoard;
object page;
graphBoard=GetGraphBoard(); ! Look for open desktop.
if (graphBoard) {
    page=graphBoard.GetPage('Voltage',1); ! Get panel named 'voltage'
    if (page) {
        ! Set x-scale variable
        page.SetXVar(line,'m:U1:bus1');
    }
}

! set default x-axis variable
object graphBoard;
object page;
graphBoard=GetGraphBoard(); ! Look for open desktop.
if (graphBoard) {
    page=graphBoard.GetPage('Voltage',1); ! Get panel named 'voltage'
    if (page) {
        ! Set default x-scale variable (time)
        page.SetXVar();
    }
}

```

Show

Displays the plot page in the graphics board.

```
int SetVipage.Show()
```

RETURNS

- 0** On success, no error occurred.
- 1** Otherwise

3.6 Others

3.6.1 BlkDef

Overview

[Compile](#)
[Encrypt](#)
[GetCheckSum](#)
[Pack](#)
[PackAsMacro](#)
[ResetThirdPartyModule](#)
[SetThirdPartyModule](#)

Compile

Compiles the model to a DLL. Can be called on an already compiled model. A study case of a project has to be active.

```
void 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 removeObjectHistory])
```

ARGUMENTS

removeObjectHistory (optional)

H andling of unencrypted object history in database, e.g. used by project versions or by undo:

- 0** Do not remove.
- 1** Do remove (default).
- 2** Show dialog and ask.

RETURNS

- 0** On success.
- 1** On error.

SEE ALSO

[BlkDef.PackAsMacro\(\)](#)

GetCheckSum

```
string 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\(\)](#)

ResetThirdPartyModule

Resets the third party licence. Only possible for non-encrypted, non-compiled blocks. Requires masterkey licence for third party module currently set.

```
int BlkDef.ResetThirdPartyModule()
```

RETURNS

- 0** On success.
- 1** On error.

SetThirdPartyModule

Sets the third party licence to a specific value. Only possible for non-encrypted, non-compiled blocks with no third party licence set so far. Requires masterkey licence for third party module to be set.

```
int BlkDef.SetThirdPartyModule(string companyCode,  
                               string moduleCode  
                             )
```

ARGUMENTS*companyCode*

D isplay name or numeric value of company code.

moduleCode

D isplay name or numeric value of third party module.

RETURNS

- 0** On success.
- 1** On error.

3.6.2 BlkSig**Overview**

[GetFromSigName](#)
[GetToSigName](#)

GetFromSigName

```
string BlkSig.GetFromSigName()
```

RETURNS

The name of the output from which the signal is connected. In cases of no connection, an empty string.

GetToSigName

```
string BlkSig.GetToSigName()
```

RETURNS

The name of the input to which the signal is connected. In cases of no connection, an empty string.

3.6.3 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.

3.6.4 CimArchive

Overview

[ConvertToBusBranch](#)

ConvertToBusBranch

Performs the conversion of CimModels in the selected CimArchive to Bus-Branch model representation.

```
void CimArchive.ConvertToBusBranch()
```

3.6.5 CimModel

Overview

[DeleteParameterAtIndex](#)
[GetAttributeEnumerationType](#)
[GetModelsReferencingThis](#)
[GetParameterCount](#)
[GetParameterNamespace](#)
[GetParameterValue](#)
[HasParameter](#)
[RemoveParameter](#)
[SetAssociationValue](#)
[SetAssociationValue](#)
[SetAttributeEnumeration](#)
[SetAttributeEnumeration](#)

[SetAttributeValue](#)
[SetAttributeValue](#)

DeleteParameterAtIndex

Removes the parameter (attribute, or association) value at the given index.

```
void CimModel.DeleteParameterAtIndex(string 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.

```
string CimModel.GetAttributeEnumerationType(string attribute)
```

ARGUMENTS

attribute

Full-name specifier of the attribute (e.g. "GeneratingUnit.genControlSource")

GetModelsReferencingThis

Returns all CIM models (CimModel) that reference the calling model.

```
set CimModel.GetModelsReferencingThis()
```

RETURNS

CIM models that reference the calling model. The order of the set is undefined.

GetParameterCount

Returns the number of parameters (attribute, or association) of given type.

```
int CimModel.GetParameterCount(string 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).

```
string CimModel.GetParameterNamespace(string 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.

```
string CimModel.GetParameterValue(string 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(string 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 occurrences of the parameter (attribute, or association).

```
void CimModel.RemoveParameter(string 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.

```
void CimModel.SetAssociationValue(string association,
                                  string 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.

```
void CimModel.SetAssociationValue(string association,
                                  string value,
                                  string 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.

```
void CimModel.SetAttributeEnumeration(string attribute,
                                      string enumerationType,
                                      string 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.

```
void CimModel.SetAttributeEnumeration(string attribute,
                                      string enumerationType,
                                      string value,
                                      string 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.

```
void CimModel.SetValue(string attribute,
                      string 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.

```
void CimModel.SetValue(string attribute,
                      string value,
                      string 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")

3.6.6 CimObject

Overview

```
DeleteParameterAtIndex
GetAttributeEnumerationType
GetObjectsReferencingThis
GetObjectsWithSameId
GetParameterCount
GetParameterNamespace
GetParameterValue
GetPfObjects
HasParameter
RemoveParameter
SetAssociationValue
SetAssociationValue
SetAttributeEnumeration
SetAttributeEnumeration
SetAttributeValue
SetAttributeValue
```

DeleteParameterAtIndex

Removes the parameter (attribute, or association) value at the given index.

```
void CimObject.DeleteParameterAtIndex(string parameter, int index)
```

ARGUMENTS

<i>parameter</i>	Full-name specifier of the attribute, or association (e.g. "Model.profile")
<i>index</i>	Index of the parameter

GetAttributeEnumerationType

Returns the enumeration type of the attribute.

```
string CimObject.GetAttributeEnumerationType(string attribute)
```

ARGUMENTS

<i>attribute</i>	Full-name specifier of the attribute (e.g. "GeneratingUnit.genControlSource")
------------------	---

GetObjectsReferencingThis

Returns all CIM objects (CimObject) that reference the calling object. The set of objects returned is also determined by the DependentOn and Supersedes references set in parent CIM model objects. In order for a CIM object to reference another CIM object, the parent CIM model of the former object has to hold a reference to the parent CIM model of the later object.

```
set CimObject.GetObjectsReferencingThis()
```

g

RETURNS

CIM objects that reference the calling object. The order of the set is undefined.

GetObjectsWithSameId

Returns all CIM objects (CimObject) that have the same Resource ID as this object.

```
set CimObject.GetObjectsWithSameId()
```

RETURNS

CIM objects that have the same Resource ID as this object.

GetParameterCount

Returns the number of parameters (attribute, or association) of given type.

```
int CimObject.GetParameterCount(string 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).

```
string CimObject.GetParameterNamespace(string 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.

```
string CimObject.GetParameterValue(string 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

GetPfObjects

Returns all PF objects that have the same Resource ID as this CIM object.

```
set CimObject.GetPfObjects()
```

RETURNS

PF objects that have the same Resource ID as this CIM object.

HasParameter

Checks whether the CimObject has the parameter (attribute, or association) specified.

```
int CimObject.HasParameter(string 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 occurrences of the parameter (attribute, or association).

```
void CimObject.RemoveParameter(string 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.

```
void CimObject.SetAssociationValue(string association,
                                  string 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.

```
void CimObject.SetAssociationValue(string association,
                                   string value,
                                   string 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.

```
void CimObject.SetAttributeEnumeration(string attribute,
                                       string enumerationType,
                                       string 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.

```
void CimObject.SetAttributeEnumeration(string attribute,
                                       string enumerationType,
                                       string value,
                                       string 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.

```
void CimObject.SetAttributeValue(string attribute,
                                string 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.

```
void CimObject.SetAttributeValue(string attribute,
                                string value,
                                string 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")

3.6.7 GrpPage

Overview

[DoAutoScale](#)
[DoAutoScaleX](#)
[DoAutoScaleY](#)
[GetOrInsertCurvePlot](#)
[GetOrInsertDiscreteBarPlot](#)
[GetOrInsertXYPlot](#)
[GetPlot](#)
[RemovePage](#)
[SetAutoScaleModeX](#)
[SetAutoScaleModeY](#)
[SetLayoutMode](#)
[SetResults](#)
[SetScaleTypeX](#)
[SetScaleTypeY](#)
[SetScaleX](#)
[SetScaleY](#)
[Show](#)

DoAutoScale

Adapts axis ranges of all plots on the page such that they show the entire data range.

```
void GrpPage.DoAutoScale([int axisDimension])
```

ARGUMENTS

axisDimension (optional)

Limits auto-scaling to one dimension. Possible values:

- 0** scale only x-axes
- 1** scale only y-axes

EXAMPLE

The following example looks for a page named 'Voltage' and performs an automatic scaling of both the x-axes and y-axes.

```
object page;
object graphicsBoard;
! Look for opened graphics board.
graphicsBoard=GetGraphicsBoard();
if (graphicsBoard) {
    page=graphicsBoard.GetPage('Voltage', 0);
    if (page) {
        page.DoAutoScale();
    }
}
```

DoAutoScaleX

Adapts x-axis ranges of all plots on the page such that they show the entire data range.

```
void GrpPage.DoAutoScaleX()
```

DoAutoScaleY

Adapts y-axis ranges of all plots on the page such that they show the entire data range.

```
void GrpPage.DoAutoScaleY()
```

GetOrInsertCurvePlot

Finds a curve plot by name, or creates it if not found.

```
object GrpPage.GetOrInsertCurvePlot(string name,
                                    [int create]
                                    )
```

ARGUMENTS

name Name of plot

create=1 (optional)

Possible values:

- 0** do not create new plot
- 1** create plot if it does not exist already

RETURNS

PltLinebarplot object

GetOrInsertDiscreteBarPlot

Finds a discrete bar plot by name, or creates it if not found. A discrete bar plot is a PltLinebarplot whose x-axis mode is set to 'Discrete', i.e., it shows net elements on the x-axis.

```
object GrpPage.GetOrInsertDiscreteBarPlot(string name,
                                         [int create]
                                         )
```

ARGUMENTS

name Name of plot

create=1 (optional)

Possible values:

0 do not create new plot

1 create plot if it does not exist already

RETURNS

PltLinebarplot object

GetOrInsertXYPlot

Finds a XY plot by name, or creates it if not found. A XY plot is a PltLinebarplot whose x-axis mode is set to 'XY'.

```
object GrpPage.GetOrInsertXYPlot(string name,
                                 [int create]
                                 )
```

ARGUMENTS

name Name of plot

create=1 (optional)

Possible values:

0 do not create new plot

1 create plot if it does not exist already

RETURNS

PltLinebarplot object

GetPlot

Returns the plot on this page with the given name.

```
object GrpPage.GetPlot(string name)
```

ARGUMENTS

name Name of the plot to look for

RETURNS

Plot object if found, or NULL otherwise

EXAMPLE

The following example looks for a plot named 'RST' on the plot page named 'Voltage'.

```
object graphicsBoard;
object page;
object plot;
graphicsBoard=GetGraphicsBoard(); ! Look for open desktop
if (graphicsBoard) {
    page=graphicsBoard.GetPage('Voltage',1);
    if (page) {
        plot=page.GetPlot('RST');
    }
}
```

RemovePage

Closes the graphic page, if currently shown, and deletes it from the database.

```
int GrpPage.RemovePage()
```

SetAutoScaleModeX

Defines whether the x-axes on the page should automatically adapt their range on data changes.

```
void GrpPage.SetAutoScaleModeX(int mode)
```

ARGUMENTS

mode Possible values:

- 0** off (do not react to data changes)
- 1** adapt scale after calculation has finished
- 2** adapt scale during live plotting

SetAutoScaleModeY

Defines whether the y-axes on the page should automatically adapt their range on data changes.

```
void GrpPage.SetAutoScaleModeY(int mode)
```

ARGUMENTS

mode Possible values:

- 0** off (do not react to data changes)
- 1** adapt scale after calculation has finished
- 2** adapt scale during live plotting

SetLayoutMode

Defines the automatic arrangement of plots on the page.

```
void GrpPage.SetLayoutMode (int mode)
```

ARGUMENTS

mode Possible values:

- 0** off (do not arrange plots automatically)
- 1** arrange plots vertically
- 2** arrange plots on grid

SetResults

Sets the default results object of page.

```
void GrpPage.SetResults (object res)
```

ARGUMENTS

res Result object to set or NULL to reset. Valid result object is any of class ElmRes, IntComtrade and IntComtradeset.

SetScaleTypeX

Sets the scale type (linear, logarithmic) of all x-axes on the page.

```
void GrpPage.setScaleTypeX (int scaleType)
```

ARGUMENTS

scaleType

Possible values:

- 0** linear
- 1** logarithmic

SetScaleTypeY

Set the scale type (linear, logarithmic, dB) of all y-axes on the page.

```
void GrpPage.setScaleTypeY (int scaleType)
```

ARGUMENTS

scaleType

Possible values:

- 0** linear
- 1** logarithmic
- 2** dB

SetScaleX

Sets the scale of all x-axes on the page.

```
void GrpPage.SetScaleX(double min,
                      double max
                     )
```

ARGUMENTS

min Minimum of x-scale.

max Maximum of x-scale.

EXAMPLE

The following example looks for a page named 'Voltage' and sets its x-axis range to [1.0;2.5].

```
object page;
object graphicsBoard;
! Look for opened graphics board.
graphicsBoard=GetGraphicsBoard();
if (graphicsBoard) {
    page=graphicsBoard.GetPage('Voltage', 0);
    if (page) {
        page.setScaleX(1.0, 2.5);
    }
}
```

SetScaleY

Sets the scale of all y-axes on the page.

```
void GrpPage.SetScaleY(double min,
                      double max
                     )
```

ARGUMENTS

min Minimum of y-scale.

max Maximum of y-scale.

Show

Displays the diagram page in the graphics board.

```
int GrpPage.Show()
```

RETURNS

0 On success, no error occurred.

1 Otherwise

3.6.8 IntAddonvars

Overview

[AddDouble](#)
[AddDoubleMatrix](#)
[AddDoubleVector](#)
[AddInteger](#)
[AddIntegerVector](#)
[AddObject](#)
[AddObjectVector](#)
[AddString](#)
[RemoveParameter](#)

AddDouble

Adds a new double parameter to the Date Extension configuration object.

```
int IntAddonvars.AddDouble(string parameterName,
                           string desc,
                           string unitText,
                           double initialValue
                           )
```

ARGUMENTS

parameterName
 The name of the new parameter
desc The description of the new parameter
unitText The unit of the new parameter
initialValue
 The initial value of the new parameter

RETURNS

Zero if attribute was successfully added. One based row number if there already is a parameter with the same name.

AddDoubleMatrix

Adds a new double vector parameter to the Date Extension configuration object.

```
int IntAddonvars.AddDoubleMatrix(string parameterName,
                                 string desc,
                                 int initialRows,
                                 int initialColumns,
                                 string unitText,
                                 double initialValue
                                 )
```

ARGUMENTS

parameterName
 The name of the new parameter
desc The description of the new parameter

initialRows

The initial number of rows for the matrix

initialColumns

The initial number of columns for the matrix

unitText The unit of the new parameter

initialValue

The initial value for the elements

RETURNS

Zero if attribute was successfully added. One based row number if there already is a parameter with the same name.

AddDoubleVector

Adds a new double vector parameter to the Date Extension configuration object.

```
int IntAddonvars.AddDoubleVector(string parameterName,
                                string desc,
                                int initialSize,
                                string unitText,
                                double initialValue
                               )
```

ARGUMENTS

parameterName

The name of the new parameter

desc The description of the new parameter

initialSize The initial size of the vector

unitText The unit of the new parameter

initialValue

The initial value for the elements

RETURNS

Zero if attribute was successfully added. One based row number if there already is a parameter with the same name.

AddInteger

Adds a new integer parameter to the Date Extension configuration object.

```
int IntAddonvars.AddInteger(string parameterName,
                            string desc,
                            string unitText,
                            int initialValue
                           )
```

ARGUMENTS

parameterName
 The name of the new parameter
desc The description of the new parameter
unitText The unit of the new parameter
initialValue
 The initial value of the new parameter

RETURNS

Zero if attribute was successfully added. One based row number if there already is a parameter with the same name.

AddIntegerVector

Adds a new integer vector parameter to the Date Extension configuration object.

```
int IntAddonvars.AddIntegerVector(string parameterName,
                                  string desc,
                                  int initialSize,
                                  string unitText,
                                  int initialValue)
```

ARGUMENTS

parameterName
 The name of the new parameter
desc The description of the new parameter
initialSize The initial size of the vector
unitText The unit of the new parameter
initialValue
 The initial value for the elements

RETURNS

Zero if attribute was successfully added. One based row number if there already is a parameter with the same name.

AddObject

Adds a new string parameter to the Date Extension configuration object.

```
int IntAddonvars.AddObject(string parameterName,
                           string desc,
                           string classFilter
                           )
```

ARGUMENTS

parameterName
 The name of the new parameter
desc The description of the new parameter
classFilter
 The filter for the objects which are allowed for selection

RETURNS

Zero if attribute was successfully added. One based row number if there already is a parameter with the same name.

AddObjectVector

Adds a new object vector parameter to the Date Extension configuration object.

```
int IntAddonvars.AddObjectVector(string parameterName,  
                                string desc,  
                                int initialSize,  
                                string classFilter  
)
```

ARGUMENTS*parameterName*

The name of the new parameter

desc

The description of the new parameter

initialSize

The initial size of the vector

classFilter

The filter for the objects which are allowed for selection

RETURNS

Zero if attribute was successfully added. One based row number if there already is a parameter with the same name.

AddString

Adds a new string parameter to the Date Extension configuration object.

```
int IntAddonvars.AddString(string parameterName,  
                           string desc,  
                           string unitText,  
                           string initialValue  
)
```

ARGUMENTS*parameterName*

The name of the new parameter

desc

The description of the new parameter

unitText

The unit of the new parameter

initialValue

The initial value of the new parameter

RETURNS

Zero if attribute was successfully added. One based row number if there already is a parameter with the same name.

RemoveParameter

Removes the given parameter from the Data Extension configuration.

```
void IntAddonvars.RemoveParameter(string parameterName)
```

ARGUMENTS

parameterName

The name of the parameter to be removed

3.6.9 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(object other)
```

ARGUMENTS

other 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(object other)
```

ARGUMENTS

other 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 occurred

SEE ALSO

[IntScheme.Consolidate\(\)](#)

Deactivate

De-activates the study case.

```
int IntCase.Deactivate()
```

RETURNS

- 0** on success
- 1** on error

SetStudyTime

Sets the current Study Case time to seconds since 01.01.1970 00:00:00. Use IntCase:iStudyTime for getting current Study Case time.

```
void IntCase.SetStudyTime(int dateTime)
```

ARGUMENTS

dateTime Seconds since 01.01.1970 00:00:00.

3.6.10 IntComtrade

Overview

ConvertToASCIIFormat*
ConvertToBinaryFormat*
FindColumn*
FindMaxInColumn*
FindMinInColumn*
GetAnalogueDescriptions
GetDescription*
GetDigitalDescriptions
GetNumberOfAnalogueSignalDescriptions
GetNumberOfColumns*
GetNumberOfDigitalSignalDescriptions
GetNumberOfRows*
GetObjectValue*
GetSignalHeader
GetUnit*
GetValue*
GetVariable*
Load*
Release*
SortAccordingToColumn*

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 occurred, 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 occurred, e.g. file is already in binary format.

FindColumn*

Returns the first column matching the variable name.

```
int IntComtrade.FindColumn(string 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.

EXAMPLE

The following example searches a variable in the first comtrade object found in the study case.

```
object intComtrade;
string variable;
int index;
intComtrade = GetFromStudyCase('IntComtrade');
if (nullptr=intComtrade) {
    exit();
}
variable = 'U RMS Feeder 1 (c)';
intComtrade.Load();
index = intComtrade.FindColumn(variable);
printf('Column index of %s is %d',variable,index);
! this will not find the variable again, because we are starting
! the search after the match
index = intComtrade.FindColumn(variable,index+1);
printf('Column index of %s is %d',variable,index);

intComtrade.Release();
```

FindMaxInColumn*

Find the maximum value of the variable in the given column.

```
int IntComtrade.FindMaxInColumn(int column,
                               [double& value])
```

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.

EXAMPLE

```

object case,
      elmRes;
set    resObjs;
int   row,
      column;
double value;

column = 1;
case = GetActiveStudyCase();
if (nullptr==case) {
    exit(1); ! there is no active study case
}
resObjs = case.GetContents('All calculations.ElmRes',0); ! get all ElmRes stored in case
elmRes = resObjs.First();
if (elmRes) {
    elmRes.Load();
    row = elmRes.FindMaxInColumn(column,value);
    if (row<0) {
        Info('The maximum of column %d was not found',column);
    }
    else {
        Info('The maximum of column %d is %f (row: %d)',column, value, row);
    }
    row = elmRes.FindMinInColumn(column,value);
    if (row<0) {
        Info('The minimum of column %d was not found',column);
    }
    else {
        Info('The minimum of column %d is %f (row: %d)',column, value, row);
    }
    elmRes.Release();
}

```

FindMinInColumn*

Find the minimum value of the variable in the given column.

```

int IntComtrade.FindMinInColumn(int column,
                                 [double& value])

```

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.

EXAMPLE

See example of [IntComtrade.FindMaxInColumn](#).

GetAnalogueDescriptions

Get the descriptions of the analogue channel Please note that the comtrade info file contains proprietary data from PFM. Info files not written by DlgSILENT PFM are not supported.

```
string IntComtrade.GetAnalogueDescriptions(int index)
```

ARGUMENTS

index Digital channel index

RETURNS

Descriptions for analogue channel in the comtrade info file.

GetDescription*

Get the description of a column.

```
string 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.

EXAMPLE

See example of [IntComtrade.GetVariable](#).

GetDigitalDescriptions

Get the descriptions of the digital channel Please note that the comtrade info file contains proprietary data from PFM. Info files not written by DlgSILENT PFM are not supported.

```
string IntComtrade.GetDigitalDescriptions(int index)
```

ARGUMENTS

index Digital channel index

RETURNS

Descriptions for digital channel in the comtrade info file.

GetNumberOfAnalogueSignalDescriptions

Gets the number of descriptions for analogue channels in the comtrade info file. Please note that the comtrade info file contains proprietary data from PFM. Info files not written by DlgSILENT PFM are not supported.

```
int IntComtrade.GetNumberOfAnalogueSignalDescriptions()
```

RETURNS

Number of descriptions for analogue channels in the comtrade info file.

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.

EXAMPLE

See example of [IntComtrade.InitialiseWriting](#).

GetNumberOfDigitalSignalDescriptions

Get the number of descriptions for digital channels in the comtrade info file. Please note that the comtrade info file contains proprietary data from PFM. Info files not written by DlgSILENT PFM are not supported.

```
int IntComtrade.GetNumberOfDigitalSignalDescriptions()
```

RETURNS

Number of descriptions for digital channels in the comtrade info 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.

EXAMPLE

See example of [IntComtrade.InitialiseWriting](#).

GetObjectValue*

Returns a value from a result object for row iX of curve col.

```
int IntComtrade.GetObjectValue(object& o,
                               int iX,
                               [int col])
```

ARGUMENTS

o (out) The object 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 analysis, the value is invalid in case that it was not written, because it was below the recording limit. Result files created using DPL/Python are always full and will not return invalid values.

EXAMPLE

See example of [IntComtrade.SortAccordingToColumn](#).

GetSignalHeader

Get the headline of the channel section in the comtrade info file. Please note that the comtrade info file contains proprietary data from PFM. Info files not written by DIgSILENT PFM are not supported.

```
string IntComtrade.GetSignalHeader()
```

RETURNS

Headline of signal descriptions

GetUnit*

Get the unit of a column.

```
string 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.

EXAMPLE

See example of [IntComtrade.GetVariable](#).

GetValue*

Returns a value from a result object for row iX of curve col.

```
int IntComtrade.GetValue(double& d,
                         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 analysis, the value is invalid in case that it was not written, because it was below the recording limit. Result files created using DPL/Python are always full and will not return invalid values.

EXAMPLE

See example of [IntComtrade.SortAccordingToColumn](#).

GetVariable*

Get variable name of column

```
string 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.

EXAMPLE

```
object case,
       elmRes;
string name,
      unit,
```

```

    short,
    long;

set      resObjs;
int      variables,
        column;
case = GetActiveStudyCase();
if (nullptr=case) {
    exit(1); ! there is no active study case
}
resObjs = case.GetContents('*.ElmRes',0); ! get all ElmRes stored in case
elmRes = resObjs.First();
if (nullptr=elmRes) {
    exit(1);
}
elmRes.Load();
variables = elmRes.GetNumberOfColumns();
if (variables>0) {
    printf('Name; unit; Short Description; Long Description');
    for (column=-1; column<variables; column+=1) {
        ! -1 returns default column
        name = elmRes.GetVariable(column);
        unit = elmRes.GetUnit(column);
        short= elmRes.GetDescription(column,1);
        long = elmRes.GetDescription(column);
        printf('%s; %s; %s; %s', name, unit, short, long);
    }
}
elmRes.Release();

```

Load*

Loads the data of a result object (**IntComtrade**) in memory for reading.

```
void IntComtrade.Load()
```

EXAMPLE

```

object case,
       elmRes;
set      resObjs;
int      variables,rows;

case = GetActiveStudyCase();
if (nullptr=case) {
    exit(1); ! there is no active study case
}
resObjs = case.GetContents('*.ElmRes',0); ! get all ElmRes stored in case
elmRes = resObjs.First();
if (nullptr=elmRes) {
    exit(1);
}
elmRes.Load();
variables = elmRes.GetNumberOfColumns();
rows      = elmRes.GetNumberOfRows();
Info('The result file %s contains %d columns and %d rows',elmRes, variables, rows);
elmRes.Release();

```

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.

```
void IntComtrade.Release()
```

EXAMPLE

See example of [IntComtrade.Load](#).

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.

EXAMPLE

The following example outputs column 1 and 2 for the first 50 rows. The output is repeated after sorting the data according to column 1.

```
object case,
        elmRes;
set    resObjs;
int    row,
       rows,
       column,
       failed;
double value1, value2;

case = GetActiveStudyCase();
if (nullptr=case) {
    exit(1); ! there is no active study case
}
resObjs = case.GetContents('* ElmRes', 0); ! get all ElmRes stored in case
elmRes = resObjs.First();
if (nullptr=elmRes) {
    exit(1);
}
elmRes.Load();
column = 1;
rows = 50;
Info('Orginal data in first %d rows of column %d and %d', rows, column, column+1);
failed = elmRes.GetValue(value1, row, column);
failed = elmRes.GetValue(value2, row, column+1);
row = 0;
while (.not.failed) {
    row += 1;
```

```

printf('%04d %f %f', row, value1, value2); ! report row starting with 1
failed = elmRes.GetValue(value1, row, column);
failed = elmRes.GetValue(value2, row, column+1);
if (row > rows) {
    break;
}
failed = elmRes.SortAccordingToColumn(column);
Info('Sorted data in first %d rows in column %d and %d', rows, column, column+1);
failed = elmRes.GetValue(value1, row, column);
failed = elmRes.GetValue(value2, row, column+1);
row = 0;
while (.not.failed) {
    row += 1;
    printf('%04d %f %f', row, value1, value2); ! report row starting with 1
    failed = elmRes.GetValue(value1, row, column);
    failed = elmRes.GetValue(value2, row, column+1);
    if (row > rows) {
        break;
    }
}
elmRes.Release();

```

3.6.11 IntComtradeset

Overview

[FindColumn*](#)
[FindMaxInColumn*](#)
[FindMinInColumn*](#)
[GetAnalogueDescriptions](#)
[GetDescription*](#)
[GetDigitalDescriptions](#)
[GetNumberOfAnalogueSignalDescriptions](#)
[GetNumberOfColumns*](#)
[GetNumberOfDigitalSignalDescriptions](#)
[GetNumberOfRows*](#)
[GetObjectValue*](#)
[GetSignalHeader](#)
[GetUnit*](#)
[GetValue*](#)
[GetVariable*](#)
[Load*](#)
[Release*](#)
[SortAccordingToColumn*](#)

FindColumn*

Returns the first column matching the variable name.

```
int IntComtradeset.FindColumn(string 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.

EXAMPLE

The following example searches a variable in the first comtrade object found in the study case.

```
object intComtradeset;
string variable;
int index;
intComtradeset = GetFromStudyCase('IntComtradeset');
if (nullptr=intComtradeset) {
    exit();
}
variable = 'U Step-Up T1 (c)';
intComtradeset.Load();
index = intComtradeset.FindColumn(variable);
printf('Column index of %s is %d',variable,index);
! this will not find the variable again, because we are starting
! the search after the match
index = intComtradeset.FindColumn(variable,index+1);
printf('Column index of %s is %d',variable,index);
intComtradeset.Release();
```

FindMaxInColumn*

Find the maximum value of the variable in the given column.

```
int IntComtradeset.FindMaxInColumn(int column,
                                    [double& value])
```

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.

EXAMPLE

```

object case,
      elmRes;
set   resObjs;
int   row,
      column;
double value;

column = 1;
case = GetActiveStudyCase();
if (nullptr==case) {
    exit(1); ! there is no active study case
}
resObjs = case.GetContents('All calculations.ElmRes',0); ! get all ElmRes stored in case
elmRes = resObjs.First();
if (elmRes) {
    elmRes.Load();
    row = elmRes.FindMaxInColumn(column,value);
    if (row<0) {
        Info('The maximum of column %d was not found',column);
    }
    else {
        Info('The maximum of column %d is %f (row: %d)',column, value, row);
    }
    row = elmRes.FindMinInColumn(column,value);
    if (row<0) {
        Info('The minimum of column %d was not found',column);
    }
    else {
        Info('The minimum of column %d is %f (row: %d)',column, value, row);
    }
    elmRes.Release();
}

```

FindMinInColumn*

Find the minimum value of the variable in the given column.

```
int IntComtradeset.FindMinInColumn(int column,
                                    [double& value])
```

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.

EXAMPLE

See example of [IntComtradeset.FindMaxInColumn](#).

GetAnalogueDescriptions

Get the descriptions of the analogue channel Please note that the comtrade info file contains proprietary data from PFM. Info files not written by DlgSILENT PFM are not supported.

```
string IntComtradeset.GetAnalogueDescriptions(int index)
```

ARGUMENTS

index Digital channel index

RETURNS

Descriptions for analogue channel in the comtrade info file.

GetDescription*

Get the description of a column.

```
string 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.

EXAMPLE

See example of [IntComtradeset.GetVariable](#).

GetDigitalDescriptions

Get the descriptions of the digital channel Please note that the comtrade info file contains proprietary data from PFM. Info files not written by DlgSILENT PFM are not supported.

```
string IntComtradeset.GetDigitalDescriptions(int index)
```

ARGUMENTS

index Digital channel index

RETURNS

Descriptions for digital channel in the comtrade info file.

GetNumberOfAnalogueSignalDescriptions

Gets the number of descriptions for analogue channels in the comtrade info file. Please note that the comtrade info file contains proprietary data from PFM. Info files not written by DlgSILENT PFM are not supported.

```
int IntComtradeset.GetNumberOfAnalogueSignalDescriptions()
```

RETURNS

Number of descriptions for analogue channels in the comtrade info file.

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.

EXAMPLE

See example of [IntComtradeset.InitialiseWriting](#).

GetNumberOfDigitalSignalDescriptions

Get the number of descriptions for digital channels in the comtrade info file. Please note that the comtrade info file contains proprietary data from PFM. Info files not written by DlgSILENT PFM are not supported.

```
int IntComtradeset.GetNumberOfDigitalSignalDescriptions()
```

RETURNS

Number of descriptions for digital channels in the comtrade info 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.

EXAMPLE

See example of [IntComtradeset.InitialiseWriting](#).

GetObjectValue*

Returns a value from a result object for row iX of curve col.

```
int IntComtradeset.GetObjectValue(object& o,
                                  int ix,
                                  [int col])
```

ARGUMENTS

o (out) The object 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 analysis, the value is invalid in case that it was not written, because it was below the recording limit. Result files created using DPL/Python are always full and will not return invalid values.

EXAMPLE

See example of [IntComtradeset.SortAccordingToColumn](#).

GetSignalHeader

Get the headline of the channel section in the comtrade info file. Please note that the comtrade info file contains proprietary data from PFM. Info files not written by DIgSILENT PFM are not supported.

```
string IntComtradeset.GetSignalHeader()
```

RETURNS

Headline of signal descriptions

GetUnit*

Get the unit of a column.

```
string 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.

EXAMPLE

See example of [IntComtradeset.GetVariable](#).

GetValue*

Returns a value from a result object for row iX of curve col.

```
int IntComtradeset.GetValue(double& d,
                             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 analysis, the value is invalid in case that it was not written, because it was below the recording limit. Result files created using DPL/Python are always full and will not return invalid values.

EXAMPLE

See example of [IntComtradeset.SortAccordingToColumn](#).

GetVariable*

Get variable name of column

```
string 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.

EXAMPLE

```
object case,
      elmRes;
string name,
      unit,
      short,
      long;

set    resObjs;
int   variables,
      column;
```

```

case = GetActiveStudyCase();
if (nullptr==case) {
    exit(1); ! there is no active study case
}
resObjs = case.GetContents('*.ElmRes',0); ! get all ElmRes stored in case
elmRes = resObjs.First();
if (nullptr==elmRes) {
    exit(1);
}
elmRes.Load();
variables = elmRes.GetNumberOfColumns();
if (variables>0) {
    printf('Name; unit; Short Description; Long Description');
    for (column=-1; column<variables; column+=1) {
        ! -1 returns default column
        name = elmRes.GetVariable(column);
        unit = elmRes.GetUnit(column);
        short= elmRes.GetDescription(column,1);
        long = elmRes.GetDescription(column);
        printf('%s; %s; %s; %s', name, unit, short, long);
    }
}
elmRes.Release();

```

Load*

Loads the data of a result object (**IntComtradeset**) in memory for reading.

```
void IntComtradeset.Load()
```

EXAMPLE

```

object case,
       elmRes;
set    resObjs;
int    variables,rows;

case = GetActiveStudyCase();
if (nullptr==case) {
    exit(1); ! there is no active study case
}
resObjs = case.GetContents('*.ElmRes',0); ! get all ElmRes stored in case
elmRes = resObjs.First();
if (nullptr==elmRes) {
    exit(1);
}
elmRes.Load();
variables = elmRes.GetNumberOfColumns();
rows      = elmRes.GetNumberOfRows();
Info('The result file %s contains %d columns and %d rows',elmRes, variables, rows);
elmRes.Release();

```

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.

```
void IntComtradeset.Release()
```

EXAMPLE

See example of [IntComtradeset.Load](#).

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.

EXAMPLE

The following example outputs column 1 and 2 for the first 50 rows. The output is repeated after sorting the data according to column 1.

```
object case,
        elmRes;
set    resObjs;
int    row,
       rows,
       column,
       failed;
double value1, value2;

case = GetActiveStudyCase();
if (nullptr=case) {
    exit(1); ! there is no active study case
}
resObjs = case.GetContents('*ElmRes',0); ! get all ElmRes stored in case
elmRes = resObjs.First();
if (nullptr=elmRes) {
    exit(1);
}
elmRes.Load();
column = 1;
rows = 50;
Info('Orginal data in first %d rows of column %d and %d',rows, column, column+1);
failed = elmRes.GetValue(value1,row,column);
failed = elmRes.GetValue(value2,row,column+1);
row = 0;
while (.not.failed) {
    row += 1;
```

```

printf('%04d %f %f', row, value1, value2); ! report row starting with 1
failed = elmRes.GetValue(value1, row, column);
failed = elmRes.GetValue(value2, row, column+1);
if (row > rows) {
    break;
}
failed = elmRes.SortAccordingToColumn(column);
Info('Sorted data in first %d rows in column %d and %d', rows, column, column+1);
failed = elmRes.GetValue(value1, row, column);
failed = elmRes.GetValue(value2, row, column+1);
row = 0;
while (.not.failed) {
    row += 1;
    printf('%04d %f %f', row, value1, value2); ! report row starting with 1
    failed = elmRes.GetValue(value1, row, column);
    failed = elmRes.GetValue(value2, row, column+1);
    if (row > rows) {
        break;
    }
}
elmRes.Release();

```

3.6.12 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.

```
void IntDataset.AddRef(object obj)
void IntDataset.AddRef(set 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.

```
set IntDataset.All()
```

RETURNS

All objects contained in dataset object.

Clear

Deletes all children of the dataset object.

```
void IntDataset.Clear()
```

GetAll

Returns all children of the dataset filtered according to given class name.

```
set IntDataset.GetAll(string className)
```

ARGUMENTS

className

class name filter, e.g. ElmTerm

RETURNS

All objects of given class stored in dataset object.

3.6.13 IntDocument

Overview

[Export](#)

[Import](#)

[Reset](#)

[View](#)

Export

Exports the embedded data as new file on disk. The embedded data remains unmodified. If desired it can be removed by calling the Reset() function afterwards

```
int IntDocument.Export(string filename)
```

ARGUMENTS

filename Name of export file on disk

RETURNS

0 On success.

1 On error.

SEE ALSO

[IntDocument.Import\(\)](#)

Import

Imports the content of selected file into the PowerFactory object. The data is afterwards embedded in the PowerFactory database.

```
int IntDocument.Import()
```

RETURNS

- 0** On success.
- 1** On error.

SEE ALSO

[IntDocument.Export\(\)](#)

Reset

Resets embedded data and reference to an external file.

```
int IntDocument.Reset()
```

View

Views the file in external application. If the file is embedded, it's extracted into a temporary file that is opened afterwards. Please note, the action is only executed if access to given file (type) is enabled in the 'External Access' configuration of PowerFactory (IntExtaccess).

```
int IntDocument.View()
```

RETURNS

- 0** Success, file was opened
- 1** Error, file not opened (because of invalid address or security reasons)

SEE ALSO

[IntUrl.View\(\)](#)

3.6.14 IntDplmap

Overview

[Clear*](#)
[Contains*](#)
[First*](#)
[GetValue*](#)
[Insert*](#)
[Next*](#)
[Remove*](#)
[Size*](#)
[Update*](#)

Clear*

Removes all key/value pairs from the container and resets type information.

```
void IntDplmap.Clear()
```

EXAMPLE

See example of [IntDplmap.GetValue\(\)](#).

Contains*

Checks if a key/value pair with given key is contained in the container.

```
int IntDplmap.Contains(int | double | string | object | set key)
```

ARGUMENTS

key Key of the associated pair in the container

RETURNS

1 if an entry of same key is contained, otherwise 0.

EXAMPLE

See example of [IntDplmap.GetValue\(\)](#).

First*

Outputs the first key/value pair stored in the container.

Note:

- The sequence of the returned pairs is determined by internal criteria and cannot be changed.
- It is not allowed to modify a container while iterating over it. If doing so, the next call of the Next command will return a value of 1.

Exception: Update() does not invalidate current position.

```
int IntDplmap.First (int& | double& | string& | object& | set& key,
                     int& | double& | string& | object& | set& value)
```

ARGUMENTS

key (out) Key of the associated pair in the container

value (out)

Value of the associated pair in the container

RETURNS

1 if no next entry is available in the container (e.g. end is reached), otherwise 0.

EXAMPLE

Iteration over a map:

```
'map' refers to an object of class IntDplmap that is stored inside the
!script
int t, key, m;
string value;

!insertion of some elements
map.Insert(1, 'one');
map.Insert(2, 'two');
map.Insert(3, 'three');
map.Insert(4, 'four');
map.Insert(5, 'five');

!iterate over the map and print its content
```

```

for (t = map.First(key, value); t = 0; t = map.Next(key, value)) {
    printf('%d -> %s', key, value);
}

!change value of all odd keys into upper-case while iterating over map
printf('Modifying map...');
for (t = map.First(key, value); t = 0; t = map.Next(key, value)) {
    m = modulo(key, 2);
    if (m = 1) {
        value = toupper(value);
        map.Update(key, value);
    }
}

!iterate over the map and print its content
for (t = map.First(key, value); t = 0; t = map.Next(key, value)) {
    printf('%d -> %s', key, value);
}

```

GetValue*

Returns the associated value for given key.

```
int|double|string|object|set IntDplmap.GetValue(int|double|string|object|set key,
                                                [int& error])
```

ARGUMENTS

key Key of the associated pair in the container to find.

error (optional, out)

- 1** Key was not found in container.
- 0** Key was found in the container.

RETURNS

The value which is associated to the given key or an undefined value if key is not associated with any value.

EXAMPLE

The following example shows how to use the different IntDplmap methods:

```
!'map'refers to an IntDplmap object stored inside the script
int count, i, tmp;
object o;
set aSet, bSet;
string s, s2;

!clear map
map.Clear();
count = map.Size();
printf('Map Size: %d', count);

!example for an int -> string map
!insert of some elements
map.Insert(1, 'one');
map.Insert(2, 'two');
```

```

!3 not inserted
map.Insert(4, 'four');
map.Insert(5, 'five');
count = map.Size();
printf('Map Size: %d', count);

!get values from map
for (i = 1; i < count + 1; i += 1) {
    tmp = map.Contains(i);
    if (tmp > 0){
        s = map.GetValue(i);
        printf('%d = %s', i, s);
    }
    else {
        printf('%d not contained', i);
    }
}

!replace existing elements
map.Insert(1, '1');
map.Insert(2, '2');
map.Insert(4, '4');
map.Insert(5, '5');
count = map.Size();
printf('Map Size: %d', count);

for (i = 1; i < count + 1; i += 1){
    s = map.GetValue(i, tmp);
    if (tmp = 0){
        printf('%d = %s', i, s);
    }
    else {
        printf('%d not contained', i);
    }
}

map.Clear();
count = map.Size();
printf('Map Size: %d', count);

!example for an string -> string map
map.Insert('A', 'a');
map.Insert('B', 'b');
map.Insert('C', 'c');
map.Insert('D', 'd');
s2 = 'D';
tmp = map.Contains(s2);
if (tmp > 0){
    s = map.GetValue(s2);
    printf('%s = %s', s, s2);
}

s2 = 'F'; !not contained!
tmp = map.Contains(s2);
if (tmp > 0) {
    s = map.GetValue(s2);
    printf('%s = %s', s, s2);
}
map.Clear();

```

```

!example for an int -> object map
!Terminal1 and Terminal2 object must be accessible
map.Insert(1, Terminal1);
map.Insert(2, Terminal2);
tmp = map.Contains(1);
if (tmp > 0){
    o = map.GetValue(1);
    o.ShowFullName();
}
tmp = map.Contains(2);
if (tmp > 0){
    o = map.GetValue(2);
    o.ShowFullName();
}
map.Clear();

!example for an object -> object map
map.Insert(Terminal1, Terminal2);
tmp = map.Contains(Terminal1);
if (tmp > 0){
    o = map.GetValue(Terminal1);
    o.ShowFullName();
}
map.Clear();
printf('\n');

!example for string -> set map
aSet.Add(Terminal1);
map.Insert('set1', aSet);
tmp = map.Contains('set1');
if (tmp > 0){
    bSet = map.GetValue('set1');
    o = bSet.First();
    while (o) {
        o.ShowFullName();
        o = bSet.Next();
    }
}

printf('\n');

! Terminal2 added to aSet, but not reflected by set stored in map
! (sets are always inserted by value, not by reference)
aSet.Add(Terminal2);
tmp = map.Contains('set1');
if (tmp > 0){
    bSet = map.GetValue('set1');
    o = bSet.First();
    while (o) {
        o.ShowFullName();
        o = bSet.Next();
    }
}

printf('\n');

! insert current aSet, so Terminal2 will be contained in this set
map.Insert('set1', aSet);

```

```

tmp = map.Contains('set1');
if (tmp > 0){
    bSet = map.GetValue('set1');
    o = bSet.First();
    while (o) {
        o.ShowFullName();
        o = bSet.Next();
    }
}

```

Insert*

Inserts given key and value as an associated pair into the container.

On the first insertion, the container is (automatically) typed by given data types of key and value. From now on, only keys and values of that types are accepted. (This type information is removed when [IntDplmap.Clear\(\)](#) is called.)

If given key already exists in the container, its associated value will be overwritten. (Each key can only be contained once in a map (no multi-map support).)

Note:

- Type of key and value can be different, of course.
- Sets are always inserted by value, not by reference!

```
void IntDplmap.Insert (int | double | string | object | set key,
                      int | double | string | object | set value)
```

ARGUMENTS

key Key of the associated pair in the container.

value Value of the associated pair in the container.

EXAMPLE

See example of [IntDplmap.GetValue\(\)](#).

Next*

Outputs the next key/value pair relative to the last key/value pair in the container.

Note:

- The sequence of the returned pairs is determined by internal criteria and cannot be changed.
- It is not allowed to modify a container while iterating over it. If doing so, the next call of the Next command will return a value of 1.

Exception: Update() does not invalidate current position.

```
int IntDplmap.Next (int& | double& | string& | object& | set& key,
                     int& | double& | string& | object& | set& value)
```

ARGUMENTS

key (out) Key of the associated pair in the container.

value (out)
Value of the associated pair in the container.

RETURNS

1 if no next entry is available in the container (e.g. end is reached), otherwise 0.

EXAMPLE

See example of [IntDplmap.First\(\)](#).

Remove*

Removes the key/value pair for given key from the container. No error will occur, if the key is not contained in the container.

```
void IntDplmap.Remove (int | double | string | object | set key)
```

ARGUMENTS

key Key of the associated pair in the container

EXAMPLE

See example of [IntDplmap.GetValue\(\)](#).

Size*

Returns the number of key/value pairs stored in the container.

```
int IntDplmap.Size()
```

RETURNS

Number of key-value pairs stored in the container.

EXAMPLE

See example of [IntDplmap.GetValue\(\)](#).

Update*

Is a special insert function that can be used for updating key/value pairs in the map. It can only be used if the key is already contained in the map.

```
void IntDplmap.Update(int | double | string | object | set key,
                      int | double | string | object | set value)
```

ARGUMENTS

key Key of the associated pair in the container

value Value of the associated pair in the container

EXAMPLE

See example of [IntDplmap.GetValue\(\)](#).

3.6.15 IntDplvec

Overview

[Clear*](#)
[Get*](#)
[IndexOf*](#)
[Insert*](#)
[Remove*](#)
[Size*](#)
[Sort*](#)

Clear*

Removes all elements from the container and resets the typ information.

```
void IntDplvec.Clear()
```

EXAMPLE

See example of [IntDplvec.Get\(\)](#).

Get*

Returns the element stored at given position in the container.

```
int|double|string|object|set IntDplvec.Get(int position)
```

ARGUMENTS

position Position in the container. It is zero-based and must always be lesser than the container's size.

RETURNS

Element stored at given position in the container.

EXAMPLE

The following example shows how to use the different IntDplvec methods:

```
! 'vec' always refers to an IntDplvec object that is stored inside the script
int count, i, tmp;

! clear vector at the beginning
vec.Clear();

! output vector's size
count = vec.Size();
printf('Vec Size: %d', count);

! insert some integers
for (i = 0; i < 10; i += 1) {
    vec.Insert(i);
}
```

```

! output all values stored in vector
count = vec.Size();
printf('Vec Size: %d', count);
for (i = 0; i < count; i += 1) {
    tmp = vec.Get(i);
    printf('Vector[%d] = %d', i, tmp);
}

! replace elements
for (i = 0; i < count; i += 1) {
    tmp = vec.Get(i);
    tmp = tmp * 10;
    vec.Insert(i, tmp);
}

! output all values stored in vector
count = vec.Size();
printf('Vec Size: %d', count);
for (i = 0; i < count; i += 1) {
    tmp = vec.Get(i);
    printf('Vector[%d] = %d', i, tmp);
}

! delete every 2nd element
for (i = count - 1; i >= 0; i -= 2){
    vec.Remove(i);
}

! output all values stored in vector
count = vec.Size();
printf('Vec Size: %d', count);
for (i = 0; i < count; i += 1) {
    tmp = vec.Get(i);
    printf('Vector[%d] = %d', i, tmp);
}

! insert some values and search their positions afterwards
vec.Insert(0, 33);
vec.Insert(2, 33);
vec.Insert(33);
tmp = vec.IndexOf(33);
while (tmp > -1) {
    printf('Value 33 found at: %d', tmp);
    tmp += 1;
    tmp = vec.IndexOf(33, tmp);
}

! empty the vector
vec.Clear();

```

IndexOf*

Returns the position where the given element is stored in the container.

```

int IntDplvec.IndexOf(int|double|string|object|set element,
                      [int startPosition]
)

```

ARGUMENTS

element Element for which the position will be searched.

startPosition

Start position from which the next occurrence greater or equal to this position is searched.

RETURNS

Position of the the given element in the container. The returned position is zero-based. If no occurrence was found, -1 is returned.

EXAMPLE

See example of [IntDplvec.Get\(\)](#).

Insert*

Inserts an element at given position into the container. If no position is given then the element is appended to the back. Inserting an element to an empty container fixes the type of elements which can be hold by itself. Clearing the container resets this type information.

```
void IntDplvec.Insert (int|double|string|object|set element)
void IntDplvec.Insert (int position,
                      int|double|string|object|set element,
                      )
```

ARGUMENTS

element Element to be inserted.

position Position (zero-based) to insert the element at. Any old entry at that position will be overwritten. Note: The size of the vector is automatically increased if given position is greater than current size.

EXAMPLE

See example of [IntDplvec.Get\(\)](#).

Remove*

Removes the element stored at given position from the container.

```
void IntDplvec.Remove (int position)
```

ARGUMENTS

position Given position (zero-based) at which the element is to be removed.

EXAMPLE

See example of [IntDplvec.Get\(\)](#).

Size*

Returns the number of elements stored in the container.

```
int IntDplvec.Size()
```

RETURNS

Number of elements stored in the container.

EXAMPLE

See example of [IntDplvec.Get\(\)](#).

Sort*

Sorts the elements of the vector depending on the type of elements stored inside the vector:

string lexically

double/int according to value

object according to full name (path + name) or given attribute

```
void IntDplvec.Sort ([int descending,]
                     [string attribute]
                     )
```

ARGUMENTS

descending (optional)

1 Descending sorting order

0 Ascending sorting order (default)

attribute (optional)

For objects only: Attribute according to which the sorting is done (default is full name)

EXAMPLE

See example of [IntDplvec.Get\(\)](#).

3.6.16 IntEvt**Overview**

[CreateCBEVENTS](#)

[RemoveSwitchEvents](#)

CreateCBEVENTS

Create boundary breaker events for all shc locations which occur simultaneously in this fault case.

```
void 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.

```
void 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. |

3.6.17 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(string url)
```

ARGUMENTS

url url to check

EXAMPLE

```
int i;
object extAccess;

!for demonstration purpose, create a local IntExtaccess object
!(normally stored in (System\)\Configuration\Security folder)
extAccess = this.CreateObject('IntExtaccess', 'Test');
extAccess:sPermissions:0 = '^http:+'; !grant access to http-resources

i = extAccess.CheckUrl('http://www.digsilent.de');

if (i=0) {
    printf('OK');
}
else {
    printf('No access');
}
```

RETURNS

- 0** access granted

1 access denied

3.6.18 IntGate

Overview

[AddTrigger](#)

AddTrigger

Adds either a condition or a gate to the gate.

```
void IntGate.AddTrigger(object newTrigger)
```

ARGUMENTS

newTrigger

The condition or gate that shall be added.

3.6.19 IntGrf

Overview

[MoveToLayer](#)

MoveToLayer

Moves an annotation element stored as (obsolete) *IntGrf* object to an annotation layer (*IntGrflayer*) or group (*IntGrfgroup*).

```
void IntGrf.MoveToLayer(object layer)
```

ARGUMENTS

layer Target *IntGrflayer* or *IntGrfgroup* object.

3.6.20 IntGrfgroup

Overview

[ClearData](#)
[Export](#)
[ExportToVec](#)
[Import](#)
[ImportFromVec](#)

ClearData

Removes all annotation elements from this group.

```
void IntGrfgroup.ClearData()
```

Export

Exports all objects of a group into svg-file.

```
void IntGrfgroup.Export(string 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

ExportToVec

Fills string description of annotation elements of this group into an *IntDplvec*. Clears *IntDplvec* before filling it.

```
void IntGrfgroup.ExportToVec(object intDplVec)
```

ARGUMENTS

intDplVec *IntDplvec* object to be filled.

EXAMPLE

This example exports the referred group object to a DPL-vector stored inside the DPL-script and prints all lines.

```
int iSize, i;
string sLine;

oGroup.ExportToVec(DPLVec);      ! group from External Objects, DPLVec stored in this
iSize = DPLVec.Size();

for (i = 0; i < iSize; i+=1) {
    sLine = DPLVec.Get(i);
    printf('sLine = %s',sLine);
}
```

Import

Imports svg-file into group object.

```
void IntGrfgroup.Import(string path)
```

ARGUMENTS

path Path of file to be imported.

ImportFromVec

Fills this group with the string description of annotation elements from an *IntDplvec*. Clears the group before filling it.

```
void IntGrfgroup.ImportFromVec(object intDplVec)
```

ARGUMENTS

intDp/Vec *IntDplvec* containg description of annotation elements.

EXAMPLE

This example imports the lines from an internally stored DPL-vector to a group object.

```
oGroup.ImportFromVec(DPLVec); ! group from External Objects, DPLVec stored in this
```

3.6.21 IntGrflayer

Overview

[ClearData](#)
[Export](#)
[ExportToVec](#)
[Import](#)
[ImportFromVec](#)

ClearData

Removes all annotation elements on this layer (keeps contained groups and annotation elements).

```
void IntGrflayer.ClearData()
```

Export

Exports all objects of a layer into svg-file, inclusive annotation objects of contained group objects.

```
void IntGrflayer.Export(string 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

ExportToVec

Fills string description of annotation elements of this layer into an *IntDplvec*. Clears *IntDplvec* before filling it.

```
void IntGrflayer.ExportToVec(object intDplVec)
```

ARGUMENTS

intDp/Vec *IntDplvec* object to be filled.

EXAMPLE

This example exports the referred layer object to a DPL-vector stored inside the DPL-script and prints all lines.

```
int iSize, i;
string sLine;

oLayer.ExportToVec(DPLVec);      ! layer from External Objects, DPLVec stored in this
iSize = DPLVec.Size();

for (i = 0; i < iSize; i+=1) {
    sLine = DPLVec.Get(i);
    printf('sLine = %s',sLine);
}
```

Import

Imports svg file into layer.

```
void IntGrflayer.Import(string path)
```

ARGUMENTS

path Path of file to be imported.

ImportFromVec

Fills this layer with the string description of annotation elements from an *IntDplvec*. Clears layer before filling it.

```
void IntGrflayer.ImportFromVec(object intDplVec)
```

ARGUMENTS

intDp/Vec *IntDplvec* containg description of annotation elements.

EXAMPLE

This example imports the lines from an internally stored DPL-vector to a layer object.

```
oLayer.ImportFromVec(DPLVec);      ! layer from External Objects, DPLVec stored in this
```

3.6.22 IntGrfnet

Overview

[Close](#)
[SetLayerVisibility](#)
[SetSymbolComponentVisibility](#)
[Show](#)

Close

Closes the graphic page that displays this diagram.

```
int IntGrfnet.Close()
```

RETURNS

- 0** On success, no error occurred.
- 1** Otherwise

SetLayerVisibility

Sets a layer visible or invisible.

```
void IntGrfnet.SetLayerVisibility(string sLayer,  
                                  int iVis)
```

ARGUMENTS

- sLayer* Layer to be modified.
- iVis* Visibility
 - 0** Make layer invisible.
 - 1** Make layer visible.

SetSymbolComponentVisibility

Determines which parts of net element symbols are shown in the diagram.

```
void IntGrfnet.SetSymbolComponentVisibility(int componentID,  
                                         int visible)
```

ARGUMENTS

- componentID* Component to be modified.
 - 5** Connection points
 - 7** Tap positions
 - 8** Vector groups
 - 9** Load flow arrows
 - 11** Phases
 - 13** Line sections and loads

- 14** Connection arrows
- 21** Connection numbers (block diagrams only)
- 22** Connection names (block diagrams only)
- 33** Remotely controlled substation markers
- 38** Tie open point markers
- 39** Open standby switch markers
- 40** Normally open switch markers

visible Visibility

- 0** Make component invisible.
- 1** Make component visible.

Show

Opens a diagram.

```
int IntGrfnet.Show()
```

RETURNS

- 0** On success, no error occurred.
- 1** Otherwise

EXAMPLE

```
int err;
object project, grfnet, desktop, diagramsfolder;
set objs;

project = GetActiveProject();
if (.not. project) {
  Error('No project found!');
  exit();
}
diagramsfolder = GetProjectFolder('dia');
objs = diagramsfolder.GetContents('*.*.IntGrfnet', 1);

grfnet = objs.First();
if (.not. grfnet) {
  Error('No diagram found!');
  exit();
}

err = grfnet.Show();
printf('%o.Show() returned %d', grfnet, err);
```

3.6.23 IntIcon

Overview

[Export](#)
[Import](#)

Export

Exports current icon as a bitmap file.

```
int IntIcon.Export(string filename)
```

ARGUMENTS

filename Name of export image on disk. Extension needs to be '.bmp'

RETURNS

- 0** On success.
- 1** On error.

SEE ALSO

[IntIcon.Import\(\)](#)

Import

Imports icon from a bitmap file.

```
int IntIcon.Import(string filename)
```

ARGUMENTS

filename Name of bitmap file on disk. Extension and format needs to be '.bmp'

RETURNS

- 0** On success.
- 1** On error.

SEE ALSO

[IntIcon.Export\(\)](#)

3.6.24 IntLibrary

Overview

[Activate](#)
[Deactivate](#)

Activate

Activates this library. If another library is already activated it will be deactivated first.

```
int IntLibrary.Activate()
```

RETURNS

- 1 if successful**
- 0 otherwise**

Deactivate

Deactivates this library.

```
int Library.Deactivate()
```

RETURNS

1 if successful

0 otherwise

3.6.25 IntMat

Overview

- ColLbl*
- Get*
- GetColumnLabel*
- GetColumnLabelIndex*
- GetNumberOfColumns*
- GetNumberOfRows*
- GetRowLabel*
- GetRowLabelIndex*
- 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.

```
string IntMat.Collbl(int column)
string IntMat.Collbl(string 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.

```
double 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\(\)](#)

EXAMPLE

The following example multiplies two matrices

```
int r,c,z,s,s1r,s2c;
double v1,v2,v;
s = M1.GetNumberOfColumns();
r = M2.GetNumberOfRows();
if (s<>r) {exit();}
s1r = M1.GetNumberOfRows();
s2c = M2.GetNumberOfColumns();
M3.Init(s1r,s2c);
r=1;
while (r<=s1r) {
    c=1;
    while (c<=s2c) {
        z=1; v=0.0;
        while (z<=s) {
            v1=M1.Get(r,z);
            v2=M2.Get(z,c);
            v+=v1*v2;
            z+=1;
        }
        M3.Set(r,c,v);
        c+=1;
    }
    r+=1;
}
```

GetColumnName*

Returns the label of a column.

`string IntMat.GetColumnName(int column)`

ARGUMENTS

column Column index (first column has index 1).

RETURNS

Column label of given column.

DEPRECATED NAMES

ColLbl

SEE ALSO

[IntMat.SetColumnName\(\)](#), [IntMat.GetColumnLabelIndex\(\)](#), [IntMat.GetRowLabel\(\)](#)

EXAMPLE

The following example assigns the label of the first column to the string *label*

```
string label;  
label = Mat.GetColumnLabel(1);
```

GetColumnLabelIndex*

Gets the index of a label in all column labels.

```
int IntMat.GetColumnLabelIndex(string label)
```

ARGUMENTS

label Label to search.

RETURNS

≥1 The index in the column labels, if label was found.
0 Otherwise

SEE ALSO

[IntMat.GetColumnLabel\(\)](#)

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\(\)](#)

EXAMPLE

See example of [IntMat.Get\(\)](#).

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\(\)](#)

EXAMPLE

See example of [IntMat.Get\(\)](#).

GetRowLabel*

Returns the label of a row.

```
string 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.GetRowLabelIndex\(\)](#), [IntMat.GetColumnLabel\(\)](#)

EXAMPLE

The following example assigns the label of the first row to the string *label*

```
string label;
label = Mat.GetRowLabel(1);
```

GetRowLabelIndex*

Gets the index of a label in all row labels.

```
int IntMat.GetRowLabelIndex(string label)
```

ARGUMENTS

label Label to search.

RETURNS

≥1 The index in the row labels, if it was found.
0 Otherwise

SEE ALSO

[IntMat.GetRowLabel\(\)](#)

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,
                [double 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 omitted.

RETURNS

Always 1 and can be ignored.

EXAMPLE

See example of [IntMat.Get\(\)](#).

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.

EXAMPLE

The following example inverts a matrix named `Matrix` stored inside the script.

```
int err;
err = Matrix.Invert();
if (err) {
    printf('Matrix %o is not invertible', Matrix);
}
else {
    printf('Matrix %o successfully inverted', Matrix);
}
```

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(object M1,
                     object 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,
                  [double 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 omitted.

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.

```
string IntMat.RowLbl(int row)
string IntMat.RowLbl(string label,
                     int row
                     )
```

Save*

Saves the current state of this matrix to database.

```
void 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,
              double value
              )
```

ARGUMENTS

- row* Row index, 1 based. The first row has index 1. Invalid index ($leq 0$) leads to scripting error.
- column* Column index, 1 based. The first column has index 1. Invalid index ($leq 0$) leads to scripting error.
- value* Value to assign.

RETURNS

Always 1 and can be ignored.

SEE ALSO

[IntMat.Get\(\)](#)

EXAMPLE

See example of [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.

```
void IntMat.SetColumnLabel(int column,
                           string label
                           )
```

ARGUMENTS

- column* Column index (first column has index 1).
label Label to set.

SEE ALSO

[IntMat.GetColumnLabel\(\)](#), [IntMat.GetColumnLabelIndex\(\)](#), [IntMat.SetRowLabel\(\)](#)

EXAMPLE

The following example labels some columns.

```
Mat.SetColumnLabel(1, 'transformers');
Mat.SetColumnLabel(2, 'lines');
Mat.SetColumnLabel(3, 'busbars');
```

SetRowLabel*

Sets the label of a row.

This operation is performed in memory only. Use [IntMat.Save\(\)](#) to save the modified matrix to database.

```
string IntMat.SetRowLabel(int row,
                         string label
                         )
```

ARGUMENTS

- row* Row index (first row has index 1).
label Label to set.

SEE ALSO

[IntMat.GetRowLabel\(\)](#), [IntMat.GetRowLabelIndex\(\)](#), [IntMat.SetColumnLabel\(\)](#)

EXAMPLE

The following example labels some rows.

```
Mat.SetRowLabel(1, 'transformers');
Mat.SetRowLabel(2, 'lines');
Mat.SetRowLabel(3, 'busbars');
```

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.

3.6.26 IntMon**Overview**

[AddVar](#)
[AddVars](#)
[ClearVars](#)
[GetVar*](#)
[NVars*](#)
[PrintAllVal*](#)
[PrintVal*](#)
[RemoveVar](#)

AddVar

Appends the variable “name” to the list of selected variable names.

```
void IntMon.AddVar(string name)
```

ARGUMENTS

name The variable name to add.

EXAMPLE

See example of [IntMon.GetVar](#).

AddVars

Appends the filtered variables to the list of selected variables.

```
void IntMon.AddVars(string varFilter)
```

ARGUMENTS

varFilter The filter for variables to add. For example: ‘e:’ to add all parameters of element to variable selection.

ClearVars

Clears the list of selected variable names.

```
void IntMon.ClearVars()
```

EXAMPLE

See example of [IntMon.GetVar](#).

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.

```
string IntMon.GetVar(int row)
```

ARGUMENTS

row Given row

RETURNS

The variable name in line *row*.

EXAMPLE

The following example script gets the variable set object stored inside the scripting object, removes all variables, add two new ones and outputs the variable names previously added in the input window.

```
object mon;
string name;
set tmp;
int count,
    index;

tmp = this.GetContents('monitor.IntMon', 0);
mon = tmp.First();
if (mon=nullptr) {
    mon = this.CreateObject('IntMon', 'monitor');
}
mon.ClearVars();
mon.AddVar('e:plini');
mon.AddVar('e:qlini');

Info('Variables stored');
count = mon.NVars();
for (index=0; index<count; index+=1) {
    name = mon.GetVar(index);
    printf('%d. %s', index+1, name);
}
Info('Variables stored after removing e:plini');
mon.RemoveVar('e:plini');
count = mon.NVars();
for (index=0; index<count; index+=1) {
    name = mon.GetVar(index);
    printf('%d. %s', index+1, name);
}
```

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.

EXAMPLE

See example of [IntMon.GetVar](#).

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.

```
void IntMon.PrintAllVal()
```

EXAMPLE

The following example script gets the variable set object stored inside the scripting object and reports all variables for the current calculation. Input data is not reported.

```
object mon,
       load,
       ldf;
set tmp;

tmp = this.GetContents('monitor.IntMon', 0);
mon = tmp.First();
if (mon=nullptr) {
    mon = this.CreateObject('IntMon', 'monitor');
}
ldf = GetFromStudyCase('ComLdf');
ldf.Execute();
tmp = GetCalcRelevantObjects('ElmLod');
for (load = tmp.First(); load; load = tmp.Next()) {
    mon:obj_id = load;
    mon.PrintAllVal();
}
```

PrintVal*

Prints the values of the selected variables to the output window.

```
void IntMon.PrintVal()
```

EXAMPLE

The following example script gets the variable set object stored inside the scripting object, adds two variables and reports the results for all loads found after calculating a load flow.

```
object mon,
```

```

ldf,
load;
set tmp;

tmp = this.GetContents('monitor.IntMon', 0);
mon = tmp.First();
if (mon=nullptr) {
    mon = this.CreateObject('IntMon', 'monitor');
}
mon.ClearVars();
mon.AddVar('e:plini');
mon.AddVar('e:qlini');

ldf = GetFromStudyCase('ComLdf');
ldf.Execute();
tmp = GetCalcRelevantObjects('ElmLod');
for (load = tmp.First(); load; load = tmp.Next()) {
    mon:obj_id = load;
    mon.PrintVal();
}

```

RemoveVar

Removes the variable “name” from the list of selected variable names.

```
int IntMon.RemoveVar(string name)
```

ARGUMENTS

name The variable name.

RETURNS

0 If variable with name was found and removed.

1 If the variable name was not found.

EXAMPLE

See example of [IntMon.GetVar](#).

3.6.27 IntOutage

Overview

[Apply](#)
[ApplyAll](#)
[Check*](#)
[CheckAll*](#)
[IsInStudyTime*](#)
[ResetAll](#)

Apply

```
void 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

```
void 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.

```
void IntOutage.CheckAll([int emitMsg,
                        [object gridfilter,
                         [set& notOutaged,
                          [set& wronglyOutaged]
                        )

```

ARGUMENTS

int emitMsg (optional)

whether to report inconsistencies 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

```
void 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

3.6.28 IntPlannedout

Overview

[SetRecurrence](#)

SetRecurrence

Copies the settings of a Recurrence Pattern object to the planned outage object and enables the flag Recurrent.

```
int IntPlannedout.SetRecurrence(object recurrencePattern)
```

ARGUMENTS

recurrencePattern

Recurrence pattern object, classname IntRecurrence

RETURNS

- 0** Ok, settings copied successful
- 1** Failed, passed object is NULL or not of class IntRecurrence

3.6.29 IntPlot

Overview

[SetAdaptY](#)
[SetAutoScaleY](#)
[SetScaleY](#)

SetAdaptY

Sets the Adapt Scale option of the x-scale.

```
void IntPlot.SetAdaptY(int mode,
                       [double offset]
                       )
```

ARGUMENTS

mode Possible values:

- 0** off
- 1** on

offset (optional)

Offset, unused if mode is off or empty

EXAMPLE

The following examples look for a subplot named 'RST', gets its plot type and changes the adapt scale option of the scale.

```

! Modify adapt scale option of Plot Type
object page;
object graphBoard;
object plot;
object scale;
graphBoard=GetGraphBoard(); ! get open graphics board
if (graphBoard) {
    page=graphBoard.GetPage('Voltage',1); ! get panel named Voltage
    if (page) {
        plot=page.GetVI('RST','VisPlot',1); ! Get subplot named 'RST'
        if (plot) {
            plot.SetDefScaleY(); ! reset option 'Use local y-Axis'
            scale=plot.GetScaleObjY(); ! get plot type
            if (scale) {
                scale.SetAdaptY(1,3);! Turn on adapt scale, offset is 3
            }
        }
    }
}

```

SetAutoScaleY

Sets automatic scaling mode of the y-scale. A warning is issued if an invalid mode is passed to the function.

```
void IntPlot.SetAutoScaleY(int mode)
```

ARGUMENTS

mode Possible values:

- 0** never
- 1** after simulation
- 2** during simulation

EXAMPLE

The following example sets the Auto Scale setting of the plot type to On.

```

object page;
object graphBoard;
object plot;
object scale;
graphBoard=GetGraphBoard(); ! Look for open graphics board.
if (graphBoard) {
    page=graphBoard.GetPage('Voltage',1); ! get panel named Voltage
    if (page) {
        plot=page.GetVI('RST','VisPlot',1); ! get subplot named 'RST'
        if (plot) {
            plot.SetDefScaleY(); ! reset option 'Use local y-Axis'
            scale=plot.GetScaleObjY(); ! get plot type
            if (scale) {
                scale.SetAutoScaleY(1);
            }
        }
    }
}

```

SetScaleY

Sets y-axis scale limits. A function call without arguments sets the Auto Scale setting to On without changing the scale itself.

```
void IntPlot.SetScaleY()
void IntPlot.SetScaleY(double min,
                      double max,
                      [int log]
                      )
```

ARGUMENTS

min (optional)

Minimum of y-scale.

max (optional)

Maximum of y-scale.

log (optional)

Possible values:

0 linear

1 logarithmic

EXAMPLE

The following example looks for a Subplot named 'RST' and set its y-axis scale to a logarithmic scale in the range s[1,1000]

```
object page;
object graphBoard;
object plot;
object scale;
graphBoard=GetGraphBoard(); ! get open graphics board
if (graphBoard) {
    page=graphBoard.GetPage('Voltage',1); ! get panel named 'Voltage'
    if (page) {
        plot=page.GetVI('RST','VisPlot',1);
        if (plot) {
            plot.SetDefScaleY();! Reset option 'Use local y-Axis'
            scale=plot.GetScaleObjY(); ! get plot type
            if (scale) {
                scale.setScaleY(1,1000,1);
            }
        }
    }
}
```

3.6.30 IntPrj

Overview

Activate
AddProjectToCombined
AddProjectToRemoteDatabase
Archive
BeginDataExtensionModification
CanAddProjectToRemoteDatabase
CanSubscribeProjectReadOnly
CanSubscribeProjectReadWrite
ClearInvalidReferences
CopyDataExtensionFrom
CreateVersion
Deactivate
EndDataExtensionModification
GetDerivedProjects
GetExternalReferences
GetGeoCoordinateSystem
GetLatestVersion
GetVersions
HasExternalReferences
LoadData
MergeToBaseProject
Migrate
NormaliseCombined
PackExternalReferences
Purge
RemoveProjectFromCombined
Restore
SetGeoCoordinateSystem
SubscribeProjectReadOnly
SubscribeProjectReadWrite
TransformGeoCoordinates
UnsubscribeProject
UpdateStatistics
UpdateToDefaultStructure
UpdateToMostRecentBaseVersion

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.

```
void 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

BeginDataExtensionModification

Signals the start of a Data Extension modification.

Must be terminated with EndDataExtensionModification, otherwise all changes will be discarded.
Data Extensions can only be modified when the project is active.

```
object IntPrj.BeginDataExtensionModification()
```

RETURNS

The SetDataext configuration object in the settings folder for further processing.

SEE ALSO

[IntPrj.EndDataExtensionModification\(\)](#), [SetDataext.AddConfiguration\(\)](#), [SetDataext.GetConfiguration\(\)](#),
[SetDataext.GetConfigurations\(\)](#), [SetDataext.RemoveAllConfigurations\(\)](#), [SetDataext.RemoveConfiguration\(\)](#)

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

ClearInvalidReferences

Removes all references from the project which can no longer be resolved due to target objects being deleted or projects no longer being available in the database, i.e. project stubs and non-migrated projects.

```
int IntPrj.ClearInvalidReferences(int type)
```

ARGUMENTS*type*

- 0** Remove references to deleted objects
- 1** Remove references to project stubs and non-migrated projects
- 2** Remove both

RETURNS

Number of references cleared.

CopyDataExtensionFrom

Copies the Data Extension configuration from another project into this project. The configuration is applied immediately. The target project must be active.

```
int IntPrj.CopyDataExtensionFrom(object other)
```

ARGUMENTS

other Project to copy the Data Extension configuration from.

RETURNS

- 0** Copied successfully
- 1** Target project is not active
- 2** Source object is not a project
- 3** Error during copying, see Output Window

CreateVersion

Creates a new version of project it was called on.

Optionally allows to pass the version name, the timestamp, a bool for user notification, a bool to enforce project approval and a description.

```
object IntPrj.CreateVersion([string name = '',]
                           [int timestamp = 0,]
                           [int notifyUsers = 0,]
                           [int approvalRequired = 0,]
                           [string description = ''])
object IntPrj.CreateVersion(int notifyUsersAndApprovalRequired,
                           [string name = '']
                           [int timestamp = 0,]
                           [string description = ''])
```

ARGUMENTS

name Version name. The default version name e.g. 'Project Version' is used on an empty string. (default: empty string)

timestamp Seconds since 01.01.1970 00:00:00. On zero the current date and time is used. (default: 0)

notifyUsers User notifications activated:

- 0** Create version and do not notify users (default).
- 1** Notify users.

approvalRequired

Project approval required:

- 0** Create version without approval (default).
- 1** Require approval.

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.

description

Version description. (default: empty string)

RETURNS

- object** Newly created *IntVersion* object.
- NULL** 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 |

EndDataExtensionModification

Terminates a Data Extension modification previously initiated with BeginDataExtensionModification.

This will deactivate the Study Case if the new Data Extension configuration can be applied. In case of errors the project will be deactivated and rolled back. Omitting the call to EndDataExtensionModification and exiting from the script will discard all changes to the Data Extension configuration.

```
void IntPrj.EndDataExtensionModification()
```

SEE ALSO

[IntPrj.BeginDataExtensionModification\(\)](#)

GetDerivedProjects

Return a set holding all versions created in the project.

```
set IntPrj.GetDerivedProjects()
```

RETURNS

Set holding all versions of a project.

GetExternalReferences

Fills the given map with objects from this project mapping to its external references.

```
int IntPrj.GetExternalReferences(object resultMap,  
                                [object externalReferencesSettings])
```

ARGUMENTS

resultMap

DPL map (*IntDplmap*) which will contain objects mapping to its external references. Objects without external references are not mapped.

externalReferencesSettings (optional)

External References settings object (*SetExtref*). Defines the properties for objects

being external references. The External References settings object from current user is used if omitted.

RETURNS

- 0** Getting external references succeeded.
- 1** Getting external references cancelled in dialogue or failed e.g project inactive or not migrated.

SEE ALSO

[IntPrj.HasExternalReferences\(\)](#), [IntPrj.PackExternalReferences\(\)](#)

GetGeoCoordinateSystem

Returns the EPSG code of the geographic coordinate system in which the geo coordinates of the project's net elements are stored.

```
int IntPrj.GetGeoCoordinateSystem()
```

RETURNS

EPSG code of the project's geographic coordinate system, or 0 if the coordinate system is not known.

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.

```
object 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.

```
set 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 considerGlobal = 1,]
                                  [int considerRemoteVariants = 0]
                                  )
int IntPrj.HasExternalReferences(object externalReferencesSettings)
```

ARGUMENTS

considerGlobal (optional)

- 0** References to global (non-system) objects are not considered as external references.
- 1** References to global (non-system) objects are considered as external references (default).

considerRemoteVariants (optional)

- 0** References to remote variants are not considered as external references (default).
- 1** References to remote variants are considered as external references.

externalReferencesSettings (optional)

External References settings object (SetExtref). Defines the properties for objects being external references.

RETURNS

- 0** No external reference was found.
- 1** An external reference was found.

SEE ALSO

[IntPrj.PackExternalReferences\(\)](#), [IntPrj.GetExternalReferences\(\)](#)

LoadData

Loads all objects of the project from the data base. Does nothing when called on an active project.

This function is useful to optimise searches which would traverse deep into an inactive project.

```
void IntPrj.LoadData()
```

MergeToBaseProject

Merges the modifications of a derived project to the base project.

```
int IntPrj.MergeToBaseProject(int conflictMode)
```

ARGUMENTS

conflictMode

Assignment in case of modification conflict:

- 0** Favour none. Diff browser is shown in case of conflicts.

- 1** Favour base version.
- 2** Favour derived project.
- 3** Favour base project.

RETURNS

- 0** Merge finished sucessfully.
- 1** Merge failed (no further information).
- 2** Merge failed due to failing assignment check.
- 3** Merge failed due to invalid projects (e.g. no derived project).
- 4** Merge was canceled by user.
- 5** Merge completed with errors (see output window).

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).

```
void 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)

NormaliseCombined

Normalises a combined project so it appears to be a regular project. This will remove all intermediate folders which were added when creating the combined project. This might lead to naming duplications which will be resolved by the normal logic of numbering duplicates.

```
int IntPrj.NormaliseCombined()
```

RETURNS

- 0** operation was successful
- 1** operation was cancelled because the project is active

PackExternalReferences

Packs external references of this project and prints a report to the Output Window.

```
int IntPrj.PackExternalReferences([object externalReferencesSettings])
```

ARGUMENTS

externalReferencesSettings (optional)

External References settings object (SetExtref). Defines the properties for objects being external references. The External References settings object from current user is used if omitted.

RETURNS

- 0** Packing external references succeeded.
- 1** Packing external references cancelled in dialogue or failed e.g project inactive or not migrated.

SEE ALSO

[IntPrj.HasExternalReferences\(\)](#), [IntPrj.GetExternalReferences\(\)](#)

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.

```
void 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(string 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

SetGeoCoordinateSystem

Defines the the geographic coordinate system in which the geo coordinates of the project's net elements are stored.

```
void IntPrj.SetGeoCoordinateSystem(int epsgCode)
```

ARGUMENTS

epsgCode

EPSG code of the geographic coordinate system to be used for this project. A value of 0 denotes an unknown coordinate system.

SubscribeProjectReadOnly

Subscribes a project read only if the permission is granted.

Can only be used if the database driver is set to Offline Mode.

```
void 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.

```
void IntPrj.SubscribeProjectReadWrite()
```

TransformGeoCoordinates

Transforms geographic coordinates from a source coordinate system to a destination coordinate system. Geodetic coordinates (longitude/latitude) are specified in decimal degrees. Projected coordinates are specified in meters.

```
int IntPrj.TransformGeoCoordinates(double xIn, double yIn,
                                   double& xOut, double& yOut,
                                   int sourceEpsg, int destEpsg
                                 )
```

ARGUMENTS

xIn horizontal component of the input location: x position or longitude, resp.

yIn vertical component of the input location: y position or latitude, resp.

xOut (out) horizontal component of the output location: x position or longitude, resp.

yOut (out) vertical component of the output location: y position or latitude, resp.

sourceEpsg EPSG code of the source coordinate system

destEpsg EPSG code of the destination coordinate system

RETURNS

0 operation was successful

1 an error occurred

UnsubscribeProject

Unsubscribes a project.

Can only be used if the database driver is set to Offline Mode.

```
void 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.

```
void IntPrj.UpdateStatistics()
```

UpdateToDefaultStructure

Updates folder structure of currently active project to that of the default project (used for creation of new projects). Existing folders will be moved to a new location within the project. Folders that have no correspondence in the default project will remain untouched.

NB: Folders might get moved or additional ones might be created, but no folder is deleted by this routine.

```
int IntPrj.UpdateToDefaultStructure(int createMissingFolders, int updateForeignKeys)
```

ARGUMENTS

createMissingFolders (*optional*)

- 0** Missing folders will not be created. Only existing ones will be moved to new locations, if required.
- 1** Missing folders will be created (default).

updateForeignKeys (*optional*)

- 0** Foreign-keys of folders will not be updated.
- 1** Foreign-keys of folders will be updated (default).

RETURNS

- 0** operation was successful
- 1** operation was not successful, e.g. project not active

UpdateToMostRecentBaseVersion

Updates a derived project to the most recent version of the base project. This is done by creating a new derived project from the new version and (optionally) merging the modifications from the existing derived project into it.

```
int IntPrj.UpdateToMostRecentBaseVersion(object& newDerivedProject,
                                         int versionWithNotification,
                                         int discardOwnModifications,
                                         [int conflictMode = 0]
                                         )
```

ARGUMENTS

newDerivedProject (out)

New derived project if no error occurred.

versionWithNotification

- 0** Update to the most recent version independent of the "Notify users" setting.
- 1** Update to the most recent version with "Notify users" enabled.

discardOwnModifications

- 0** Merge modifications of new version and derived project.
- 1** Discard modifications of derived project. The Compare and Merge Tool is not used at all.

conflictMode (optional)

Assignment in case of modification conflict (only used if not discarding derived modifications):

- 0** Favour none. Diff browser is shown in case of conflicts. (default)
- 1** Favour new base version.
- 2** Favour derived project.

RETURNS

- 0** Merge finished successfully.
- 1** Merge failed (no further information).
- 2** Merge failed due to failing assignment check.
- 3** Merge failed due to invalid projects (e.g. no derived project).
- 4** Merge was canceled by user.
- 5** Merge completed with errors (see output window).

3.6.31 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
- cstgen - Generator Cost Curves
- effgen - Generator Efficiency Curves

- rnd - Probabilistic Assessment
- cim - CIM Model
- common - Common Mode Failures
- demand - Demand Transfers
- dia - Diagrams
- equip - Equipment Type Library
- fault - Faults
- ras - Remedial Action Schemes
- 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
- ucc - V-Control-Curves

```
string IntPrjfolder.GetProjectFolderType()
```

RETURNS

The type of the project folder as string. For possible return values see list above.

SEE ALSO

[GetProjectFolder\(\)](#)

IsProjectFolderType*

This function checks if a project folder is of given type.

```
int IntPrjfolder.IsProjectFolderType(string 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

[GetProjectFolder\(\)](#), [IntPrjfolder.GetProjectFolderType\(\)](#)

3.6.32 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.).

```
double IntQlim.GetQlim(double p,
                      double v,
                      [double 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:

- 1** minimum value
- 1** maximum value. This is the default 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.

EXAMPLE

```
set generators;
object generator, qlim;
double minlim, maxlim;
int inp;
```

```

! gets the maximum and minimum values of reactive power limits
! of all synchronous generators, only if they have defined
! a capability curve
generators = GetCalcRelevantObjects('*.ElmSym');
for(generator=generators.First(); generator; generator=generators.Next())
{
    qlim = generator:pQlimType;
    if (qlim) {
        inp = qlim:inputmod;
        if (inp=0) { ! MW/MVAR
            minlim = qlim.GetQlim(generator:pgini, 1, -1);
            maxlim = qlim.GetQlim(generator:pgini, 1, 1);
            printf('Limits for %o in %o: min = %f, max= %f', generator, qlim, minlim, maxlim);
        }
        else { ! PU MODE
            minlim = qlim.GetQlim(generator:pgini/generator:t:sgn, 1, -1);
            maxlim = qlim.GetQlim(generator:pgini/generator:t:sgn, 1, 1);
            printf('Limits for %o in %o: min = %f, max= %f', generator, qlim, minlim, maxlim);
        }
    }
}

```

3.6.33 IntRas

Overview

AddEvent
AddTrigger
IsValid

AddEvent

Adds an event to the Remedial Action Scheme.

```
void IntRas.AddEvent(object newEvent)
```

ARGUMENTS

newEvent

The event that shall be added.

AddTrigger

Adds either a condition or a gate to the Remedial Action Scheme.

```
void IntRas.AddTrigger(object newTrigger)
```

ARGUMENTS

newTrigger

The condition or gate that shall be added.

IsValid

Checks if the Remedial Action Scheme is valid.

```
int IntrAs.IsValid(int emitMessage = 0)
```

ARGUMENTS

emitMessage

Emit messages if not equal to zero.

RETURNS

0 If invalid.

1 If valid.

3.6.34 IntRunarrange

Overview

[GetSwitchStatus*](#)

GetSwitchStatus*

Determines the status of the given switch in the running arrangement, without assigning or applying the running arrangement.

```
int IntRunarrange.GetSwitchStatus(object 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 arrangement
- 0** Switch is open
- 1** Switch is closed

EXAMPLE

```
set substations, switches;
object substation, runarrange, switch;
int status;

substations = GetCalcRelevantObjects('ElmSubstat');

for(substation = substations.First(); substation; substation = substations.Next()){
    runarrange = substation:pRA;

    if(runarrange){
        switches = substation.GetChildren(0, '*ElmCoup');
        for(switch = switches.First(); switch; switch = switches.Next()){
            status = runarrange.GetSwitchStatus(switch);
            printf('Status for %o in RA %o in substation %o. %d', switch, runarrange, substation,
            }
    }
}
```

3.6.35 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,
                      [object 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(object applyConfiguration)
int IntScenario.ApplySelective(int requestUserConfirmation,
                                object 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 success

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.

```
set 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 IntScenario.GetOperationValue(int&|double&|string&|object& value,
                                  object obj,
                                  string 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.SetOperationValue(int|double|string|object newvalue,
                                  object obj,
                                  string attribute,
                                  [int toObject = 1]
                                )
```

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 As 0 but value is also updated on object in memory. (default)

RETURNS

0 on success, otherwise 1.

3.6.36 IntScensched

Overview

Activate
 Deactivate
 DeleteRow
 GetScenario*
 GetStartTime*
 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.

```
void 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'.

```
object IntScenario.GetScenario(int iTime)
```

ARGUMENTS

iTime Time (UCTE) to get the corresponding scenario.

RETURNS

NULL No scenario at time 'iTime' defined

IntScenario Scenario will be activated at time 'iTime'

GetStartTime*

Get start and end time of the corresponding scenario.

```
int IntScenario.GetStartTime(object scenario,
                           int& startTime,
                           int& endTime)
```

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(object scenarioObject)
```

ARGUMENTS

scenarioObject
scenario object

RETURNS

- 1 Scenario not found (not part of scenario scheduler).
- ≥ 0 Vector index (row, index of scenario).

3.6.37 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 occurred.

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 NULL if no scheduler is active for this variation (IntScheme).

```
object IntScheme.GetActiveScheduler()
```

NewStage

Adds a new expansion stage into the variation (name = sname).

```
int IntScheme.NewStage(string name,
                      int activationTime,
                      int activate
                     )
```

ARGUMENTS

name Name of the new expansion stage.

activationTime

Activation time of the new expansion stage in seconds since 01.01.1970 00:00:00.

activate

- 1** The actual study time is changed to the parameter iUTCtime and the variation 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.

3.6.38 IntScheduler**Overview**

[Activate](#)
[Deactivate](#)
[Update](#)

Activate

Activates a variation scheduler. An already activated scheduler for same variation will be deactivated automatically.

```
int IntScheduler.Activate()
```

RETURNS

- = 0 On success

≠ 0 If an error has occurred

Deactivate

Deactivates a variation scheduler.

```
int IntScheduler.Deactivate()
```

RETURNS

= 0	on success
≠ 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 IntScheduler.Update()
```

RETURNS

= 0	On success
≠ 0	If an error has occurred

3.6.39 IntStage

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 corresponding time of the stage.

```
int IntStage.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*.

```
object IntSstage.CreateStageObject(int type,
                                    object 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.

EXAMPLE

```
!for a given IntSstage 'stage' and a IntThrating 'rating'

!creates a delta object for that rating
delta = stage.CreateStageObject(2, rating);
printf('Delta object %o created.', delta);

!set new values to delta object
rating.GetSize('MyMatrix', sizex, sizey);
for(x = 0; x < sizex; x+=1) {
    for (y = 0; y < sizey; y+=1) {
        rating.GetVal(val, 'MyMatrix', x, y);

        !get new value, e.g. 75% of old one
        val = 0.75 * val;
        !set value to delta object
        delta.SetVal(val, 'MyMatrix', x, y);
    }
}
```

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.

```
void IntSstage.EnableDiffMode(int enable)
```

ARGUMENTS

enable

- 0** disables the difference/comparison mode
- 1** enables the difference/comparison mode

GetVariation*

Returns variation of expansion stage.

```
object IntSstage.GetVariation()
```

RETURNS

Variation object corresponding to stage.

DEPRECATED NAMES

GetScheme

EXAMPLE

```
!for a given IntSstage 'oStage'
object variation;
variation = oStage.GetVariation();
if (variation) {
    printf('The Variation corresponding to %o is %.', oStage, variation);
}
```

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.

```
double IntSstage.IsExcluded()
```

RETURNS

1	if stage is excluded
0	if stage is considered

EXAMPLE

```
!for a given IntStage 'oStage'
int iStageStatus;
iStageStatus = oStage.IsExcluded();
if (iStageStatus=0) {
    printf('The object %o is considered for activation.', oStage);
}
else {
    printf('The object %o is not considered for activation.', oStage);
}
```

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,
                                  [string 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 IntSstage.ReadValue(double&|string&|object& value,
                        object rootObj,
                        string attributeName)
```

RETURNS

- = 0 On success.
- ≠ 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(double|string|object value,
                        object rootObj,
                        string attributeName)
```

RETURNS

- = 0 On success.
- ≠ 0 Error e.g. wrong data type.

3.6.40 IntSubset**Overview**

[Apply](#)
[ApplySelective](#)
[Clear](#)
[GetConfiguration*](#)
[GetObjects*](#)

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(object applyConfiguration)
int IntSubset.ApplySelective(int requestUserConfirmation,
                            object 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 success

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.

GetConfiguration*

Returns class and attribute configuration for the subset.

```
string IntSubset.GetConfiguration(string subset)
```

ARGUMENTS

subset

RETURNS

Returns a list of classes and attributes for which operational data are stored in subset.

GetObjects*

Returns a set of all objects for which operational data are stored in the subset.

```
set IntSubset.GetObjects()
```

RETURNS

Set of all objects for which operational data are stored in the subset

3.6.41 IntThrating**Overview**

[GetCriticalTimePhase*](#)
[GetRating*](#)

GetCriticalTimePhase*

This function returns the smallest duration (time-phase) for which the power flow is beyond the rating.

```
double IntThrating.GetCriticalTimePhase(double Flow,
                                         double 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.

EXAMPLE

```
! External object with the rating characteristic: objTrf2W
double Time, Flow, Loading
object thermRating;
thermRating = objTrf2W:pRating;
if (thermRating) {
    Flow = 800;
    Loading = 95;
    Time = thermRating.GetCriticalTimePhase(Flow, Loading);
    printf('The critical time-phase for which the power
    %10.3f beyond the loading (%10.3f) is %10.3f', Flow, Loading, Time);
}
```

GetRating*

This function returns the rating in MVA according to the thermal rating table, considering element overloading and its duration (time-phase).

```
double IntThrating.GetRating(double Loading,
                             double 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.

EXAMPLE

```
! External object with the rating characteristic: objTrf2W
double Rating, Loading, TimePhase;
object thermRating;
thermRating = objTrf2W:pRating;
if (thermRating) {
    Loading = 84;
    TimePhase = 10; ! for 10 Minutes
    Rating = thermRating.GetRating(Loading, TimePhase);
    printf('The Rating at Time-Phase %10.3f min when the loading
           is %10.3f %: %10.3f MVA', TimePhase, Loading, Rating);
}
```

3.6.42 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)

3.6.43 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.

```
void IntUser.Purge()
```

SetPassword

Sets the password for the user the function is called on.

Note: Only the administrator user is allowed to use this function. He can (re-)set the password for every user.

```
int IntUser.SetPassword(string newPassword)
```

ARGUMENTS

newpassword
Case sensitive user password to set

RETURNS

Returns whether or not the password has successfully been set. Possible values are:

0 error
1 password set successfully

TerminateSession

Allows the Administrator to log out another user. Prints an error if the current user is not the Administrator.

```
void IntUser.TerminateSession()
```

3.6.44 IntUserman

Overview

CreateGroup
CreateUser
GetGroups*
GetUsers*
UpdateGroups

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.

```
object IntUserman.CreateGroup(string 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.

```
object IntUserman.CreateUser(string 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.

```
set 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.

```
set 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.

```
void IntUserman.UpdateGroups()
```

3.6.45 IntVec

Overview

[Get*](#)
[Init*](#)
[Max*](#)
[Mean*](#)
[Min*](#)
[Resize*](#)
[Save*](#)
[Set*](#)
[Size*](#)
[Sort*](#)

Get*

Get the value in row index. Index is one based, therefore the index of the first entry is 1.

```
double IntVec.Get(int index)
```

ARGUMENTS

index Index in vector, one based.

SEE ALSO

[IntVec.Set\(\)](#)

EXAMPLE

See example of [IntVec.Sort\(\)](#).

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.

```
void IntVec.Init(int size)
```

ARGUMENTS

size The new size of the vector.

EXAMPLE

See example of [IntVec.Sort\(\)](#).

Max*

Gets the maximum value stored in the vector.

```
double 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.

```
double 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.

```
double 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.

```
void IntVec.Resize(int size)
```

ARGUMENTS

size The new size.

EXAMPLE

The following example resize a vector to size 4 and writes a avlue of 10 to the first entry.

```
object vectorObj;
int size,
    row;
double value;

vectorObj = GetFromStudyCase('IntVec');
vectorObj.Resize(0);
vectorObj.Resize(4);
vectorObj.Set(1,10);
size = vectorObj.Size(); ! should be 4
for (row=1; row<=size; row+=1) {
    value = vectorObj.Get(row);
    printf('%d: %f',row,value);
}
```

Save*

Saves the current state of this vector to database.

```
void 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.

```
void IntVec.Set(int index,  
                double value)
```

ARGUMENTS

index Index in vector.

value Value to assign in row index.

SEE ALSO

[IntVec.Get\(\)](#)

EXAMPLE

See example of [IntVec.Sort\(\)](#).

Size*

Returns the size of the vector.

```
int IntVec.Size()
```

RETURNS

The size of the vector.

EXAMPLE

See example of [IntVec.Sort\(\)](#).

Sort*

Sorts the vector.

This operation is performed in memory only. Use [IntVec.Save\(\)](#) to save the modified vector to database.

```
void IntVec.Sort([int ascending = 0])
```

ARGUMENTS

ascending

Sort order:

- 0** Highest value first (descending, default).
- 1** Smallest value first (ascending).

EXAMPLE

The following script fills a vector and reports the initial values, the values after sorting in increasing order and finally in decreasing order.

```
object vectorObj;
int size,
    row;
double value;

vectorObj = GetFromStudyCase('IntVec');
! resize to 4 rows, all containing 0
vectorObj.Init(4);
vectorObj.Set(1,1);
vectorObj.Set(2,3);
vectorObj.Set(3,2);
!vectorObj.Set(4,5); 4. row unchanged -> remains 0

size = vectorObj.Size();
Info('Initial Vector');
for (row=1; row<=size; row+=1) {
    value = vectorObj.Get(row);
    printf('%d: %f',row,value);
}
Info('Increasing');
vectorObj.Sort(1); ! sort increasing
for (row=1; row<=size; row+=1) {
    value = vectorObj.Get(row);
    printf('%d: %f',row,value);
}
Info('Decreasing');
vectorObj.Sort(0); ! sort decreasing
for (row=1; row<=size; row+=1) {
    value = vectorObj.Get(row);
    printf('%d: %f',row,value);
}
```

3.6.46 IntVecobj

Overview

[Get*](#)
[Resize*](#)
[Save*](#)
[Search*](#)
[Set*](#)
[Size*](#)

Get*

Get the object in row index. Index is one based, therefore the index of the first entry is 1.

```
object IntVecobj.Get (int index)
```

ARGUMENTS

index Index in vector, one based.

SEE ALSO

[IntVecobj.Set\(\)](#)

Resize*

Resizes the vector. Inserted new entries are initialized to NULL.

This operation is performed in memory only. Use [IntVecobj.Save\(\)](#) to save the modified vector to database.

```
void IntVecobj.Resize (int size)
```

ARGUMENTS

size The new size.

Save*

Saves the current state of this vector to database.

```
void IntVecobj.Save ()
```

Search*

Search if the object (*obj*) is part of the vector and returns the corresponding index of the vector (one based).

```
int IntVecobj.Search (object obj)
```

RETURNS

1 ... size object found, located at index

0 object not part of vector

Set*

Set the object (*obj*) 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. Object inserted are automatically initialized to a value of NULL.

This operation is performed in memory only. Use [IntVecobj.Save\(\)](#) to save the modified vector to database.

```
void IntVecobj.Set (int index,
                    object obj)
```

ARGUMENTS

- index* Index in vector.
obj Object to assign in row index.

SEE ALSO

[IntVecobj.Get\(\)](#)

Size*

Returns the size of the vector.

```
int IntVecobj.Size()
```

RETURNS

The size of the vector.

3.6.47 IntVersion**Overview**

[CreateDerivedProject](#)
[GetDerivedProjects](#)
[GetHistoricalProject](#)
[Rollback](#)

CreateDerivedProject

Creates a derived project from the version.

```
object IntVersion.CreateDerivedProject(string name,
                                       [object 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.

EXAMPLE

```
object project, version, derivedProject ;
project = GetActiveProject();
if (.not. project) {
  Error('no active project found');
  exit();
}

version = project.GetLatestVersion(0); !consider all versions
if (.not. version) {
```

```

        Error('no version found');
        exit();
    }

derivedProject = version.CreateDerivedProject('DerivedProject');

```

GetDerivedProjects

list of projects derived from this version

```
set IntVersion.GetDerivedProjects()
```

RETURNS

list of derived projects

EXAMPLE

```

object project, version, derivedProject;
set versions, projects, derivedProjects;

project = GetActiveProject();

versions = project.GetVersions();
for (version=versions.First(); version; version=versions.Next()) {
    Info('Derived projects of version %o', version);
    derivedProjects = version.GetDerivedProjects();
    for (derivedProject=derivedProjects.First(); derivedProject; derivedProject=derivedProject.Next()) {
        Info('    %o', derivedProject);
    }
}

```

GetHistoricalProject

Returns historic project within version

```
object IntVersion.GetHistoricalProject()
```

RETURNS

Returns the historic project object

EXAMPLE

```

object project, version, historicProject ;
project = GetActiveProject();
version = project.GetLatestVersion(0); !consider all versions
historicProject = version.GetHistoricalProject();

```

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

EXAMPLE

```
object project, version;
project = GetActiveProject();

! revert all changes since last version
version = project.GetLatestVersion(0);
project.Deactivate(); ! project must be inactive
version.Rollback();
```

3.6.48 IntViewbookmark

Overview

[JumpTo](#)
[UpdateFromCurrentView](#)

JumpTo

Opens the referenced diagram (if not already open) and sets the viewing area.

```
void IntViewbookmark.JumpTo()
```

UpdateFromCurrentView

Updates the bookmark's diagram and view area from the current drawing window.

```
void IntViewbookmark.UpdateFromCurrentView()
```

3.6.49 PltDataseries

Overview

[AddCurve](#)
[AddXYCurve](#)
[ClearCurves](#)
[GetDataSource](#)
[GetIntCalcres](#)

AddCurve

Appends a curve to the plot.

```
void PltDataseries.AddCurve(object element,
                            string varname,
                            [object datasource]
                            )
```

ARGUMENTS

element Element to display

varname Name of the element variable to display

datasource (optional)

Data source to assign to the curve (ElmRes or IntComtrade). If not specified, the plot's default result file will be used for the curve.

EXAMPLE

The following example finds a plot and adds a curve to it. The element `line` is assumed to be passed to the script:

```
object page;
object graphicsBoard;
object plot;
object dataseries;
graphicsBoard = GetGraphicsBoard();
if (graphicsBoard) {
    page = graphicsBoard.GetPage('Voltage');
    if (page) {
        plot = page.GetPlot('RST');
        if (plot) {
            dataseries = plot.GetDataSeries();
            if (dataseries) {
                dataseries.AddCurve(line, 'c:loading');
            }
        }
    }
}
```

AddXYCurve

Appends a curve to a XY plot.

```
void PltDataseries.AddXYCurve(object elementX,
                               string varnameX,
                               object elementY,
                               string varnameY,
                               [object datasource]
                               )
```

ARGUMENTS

elementX Element to display on x-axis

varnameX

Name of the element variable to display on x-axis

elementY Element to display on y-axis

varnameY

Name of the element variable to display on y-axis

datasource (optional)

Data source to assign to the curve (ElmRes or IntComtrade). If not specified, the plot's default result file will be used for the curve.

ClearCurves

Removes all curves from the plot.

```
void PltDataseries.ClearCurves()
```

GetDataSource

Returns the data source that is used for the curve with given index.

```
object PltDataseries.GetDataSource(int curveIndex)
```

ARGUMENTS

int curveIndex

curve index in table, must be ≥ 0

RETURNS

object Data source

NULL No data source found

GetIntCalcres

Gets all user Calculated Result objects (IntCalcres) stored inside the data series.

```
set PltDataseries.GetIntCalcres()
```

RETURNS

All Calculated Result objects (IntCalcres) stored inside the data series.

3.6.50 PltLinebarplot**Overview**

[DoAutoScale](#)
[GetAxisX](#)
[GetAxisY](#)
[GetDataSeries](#)
[GetLegend](#)
[GetTitleObject](#)
[SetAutoScaleModeX](#)
[SetAutoScaleModeY](#)
[SetAxisSharingLevelX](#)
[SetAxisSharingLevelY](#)
[SetScaleTypeX](#)
[SetScaleTypeY](#)
[SetScaleX](#)
[SetScaleY](#)

DoAutoScale

Adapts the plot's axes ranges such that they show the entire data range.

```
void PltLinebarplot.DoAutoScale([int axisDimension])
```

ARGUMENTS*axisDimension (optional)*

Limits auto-scaling to one dimension. Possible values:

- 0** scale only x-axes
- 1** scale only y-axes

GetAxisX

Returns one of the plot's x-axes.

`object PltLinebarplot.GetAxisX([int axisIndex])`**ARGUMENTS***axisIndex=0 (optional)*

Determines which x-axis should be returned:

- 0** main x-axis
- 1** second x-axis (will be created if not yet present)

RETURNS

PltAxis object

GetAxisY

Returns one of the plot's y-axes.

`object PltLinebarplot.GetAxisY([int axisIndex])`**ARGUMENTS***axisIndex=0 (optional)*

Determines which y-axis should be returned:

- 0** main y-axis
- 1** second y-axis (will be created if not yet present)

RETURNS

PltAxis object

GetDataSeries

Returns the plot's data series object (PltDataseries).

`object PltLinebarplot.GetDataSeries()`**RETURNS**

Data series object (PltDataseries)

EXAMPLE

The following example creates a curve plot named 'RST' on the plot page named 'Voltage', and retrieves its data series.

```
object graphicsBoard;
object page;
object plot;
object dataseries;
graphicsBoard = GetGraphicsBoard(); ! Look for open desktop
if (graphicsBoard) {
    page = graphicsBoard.GetPage('Voltage',1);
    if (page) {
        plot = page.GetOrInsertCurvePlot('RST');
        if (plot) {
            dataseries = plot.GetDataSeries();
        }
    }
}
```

GetLegend

Returns the plot's legend object (PltLegend).

```
object PltLinebarplot.GetLegend()
```

RETURNS

Legend object (PltLegend)

GetTitleObject

Returns the plot's title object (PltTitle).

```
object PltLinebarplot.GetTitleObject()
```

RETURNS

Title object (PltTitle)

SetAutoScaleModeX

Defines whether the plot should automatically adapt its main x-axis range on data changes.

```
void PltLinebarplot.SetAutoScaleModeX(int mode)
```

ARGUMENTS

mode Possible values:

- 0** off (do not react to data changes)
- 1** adapt scale after calculation has finished
- 2** adapt scale during live plotting

SetAutoScaleModeY

Defines whether the plot should automatically adapt its main y-axis range on data changes.

```
void PltLinebarplot.SetAutoScaleModeY(int mode)
```

ARGUMENTS

mode Possible values:

- 0** off (do not react to data changes)
- 1** adapt scale after calculation has finished
- 2** adapt scale during live plotting

SetAxisSharingLevelX

Defines whether the plot has its own x-axis or should share its x-axis across the page or the entire graphics board.

```
void PltLinebarplot.SetAxisSharingLevelX(int level)
```

ARGUMENTS

level Possible values:

- 0** local (do not share x-axis)
- 1** use x-axis from page
- 2** use x-axis from graphics board

SetAxisSharingLevelY

Defines whether the plot has its own y-axis or should share its y-axis across the page or the entire graphics board.

```
void PltLinebarplot.SetAxisSharingLevelY(int level)
```

ARGUMENTS

level Possible values:

- 0** local (do not share y-axis)
- 1** use y-axis from page
- 2** use y-axis from graphics board

SetScaleTypeX

Sets the scale type of the plot's main x-axis.

```
void PltLinebarplot.setScaleTypeX(int scaleType)
```

ARGUMENTS*scaleType*

Possible values:

- 0** linear
- 1** logarithmic

SetScaleTypeY

Set the scale type of the plot's main y-axis.

```
void PltLinebarplot.SetScaleTypeY(int scaleType)
```

ARGUMENTS*scaleType*

Possible values:

- 0** linear
- 1** logarithmic
- 2** dB

SetScaleX

Sets the scale of the plot's main x-axis.

```
void PltLinebarplot.SetScaleX(double min,  
                           double max  
)
```

ARGUMENTS*min* Minimum of x-scale.*max* Maximum of x-scale.**SetScaleY**

Sets the scale of the plot's main y-axis.

```
void PltLinebarplot.SetScaleY(double min,  
                           double max  
)
```

ARGUMENTS*min* Minimum of y-scale.*max* Maximum of y-scale.

3.6.51 RelZpol

Overview

[AssumeCompensationFactor](#)

[AssumeReRI](#)

[AssumeXeXI](#)

AssumeCompensationFactor

Triggers a calculation of the complex compensation factor and stores the result.

```
int RelZpol.AssumeCompensationFactor()
```

RETURNS

0 The compensation factor was successfully calculated.

1 An error occurred (e.g. connected branch was not found).

AssumeReRI

Triggers a calculation of the real part of the decoupled compensation factor and stores the result.

```
int RelZpol.AssumeReRI()
```

RETURNS

0 The compensation factor was successfully calculated.

1 An error occurred (e.g. connected branch was not found).

AssumeXeXI

Triggers a calculation of the imaginary part of the decoupled compensation factor and stores the result.

```
int RelZpol.AssumeXeXI()
```

RETURNS

0 The compensation factor was successfully calculated.

1 An error occurred (e.g. connected branch was not found).

3.6.52 ScnFreq

Overview

[GetLimit*](#)

[GetNumberOfViolations*](#)

[GetValue*](#)

[GetVariable*](#)

[GetViolatedElement*](#)

[GetViolationTime*](#)

GetLimit*

Returns the limit value (in p.u.) of the i^{th} violation, given by vIdx.

```
double ScnFreq.GetLimit(int vIdx)
```

ARGUMENTS

vIdx vIdx > 0

RETURNS

The limit value (in p.u.) of the i^{th} violation, given by vIdx. If vIdx is negative, zero or greater than the number of violations, then the function returns 'nan'.

EXAMPLE

See example [ScnFreq.GetViolationTime\(\)](#).

GetNumberOfViolations*

Returns the number of violations.

```
int ScnFreq.GetNumberOfViolations()
```

RETURNS

The number of violations.

EXAMPLE

See example [ScnFreq.GetViolationTime\(\)](#).

GetValue*

Returns the i^{th} value (in p.u.), given by vIdx, which causes the violation.

```
double ScnFreq.GetValue(int vIdx)
```

ARGUMENTS

vIdx vIdx > 0

RETURNS

The i^{th} value (in p.u.), given by vIdx, which causes the violation. If vIdx is negative, zero or greater than the number of violations, then the function returns 'nan'.

EXAMPLE

See example [ScnFreq.GetViolationTime\(\)](#).

GetVariable*

Returns the name of the variable of the i^{th} violation, given by vIdx.

```
string ScnFreq.GetVariable(int vIdx)
```

ARGUMENTS

vIdx vIdx > 0

RETURNS

The name of the variable of the i^{th} violation, given by vIdx. If vIdx is negative, zero or greater than the number of violations, then the function returns "NoVariable".

EXAMPLE

See example [ScnFreq.GetViolationTime\(\)](#).

GetViolatedElement*

Returns the element of the i^{th} violation, given by vIdx.

```
object ScnFreq.GetViolatedElement (int vIdx)
```

ARGUMENTS

vIdx vIdx > 0

RETURNS

The element of the i^{th} violation, given by vIdx. If vIdx is negative, zero or greater than the number of violations, then the function returns NULL.

EXAMPLE

See example [ScnFreq.GetViolationTime\(\)](#).

GetViolationTime*

Returns the time (in seconds) of the i^{th} violation, given by vIdx.

```
double ScnFreq.GetViolationTime (int vIdx)
```

ARGUMENTS

vIdx vIdx > 0

RETURNS

The time (in seconds) of the i^{th} violation, given by vIdx. If vIdx is negative, zero or greater than the number of violations, then the function returns 'nan'.

EXAMPLE

The following example gets the scan modules which have violations. For every violation of each scan module, all relevant information is shown.

```
object oSim, oRef;
set sScanTrigger;
int violation, nbViolation;

oSim = GetCaseObject('ComSim');

sScanTrigger=oSim.GetViolatedScanModules();

for(oRef = sScanTrigger.First(); oRef; oRef = sScanTrigger.Next()){
    printf('%s',oRef.GetFullName(0);
    nbViolation=oRef.GetNumberOfViolations();
    for(violation=1;violation <= nbViolation; violation=violation+1){
        printf('%f s',oRef.GetViolationTime(violation));
```

```

        printf('%o', oRef.GetViolatedElement(violation));
        printf('%s', oRef.GetVariable(violation));
        printf('%f', oRef.GetValue(violation));
        printf('%f', oRef.GetLimit(violation));
    }
}

```

3.6.53 ScnFrт

Overview

[GetLimit*](#)
[GetNumberOfViolations*](#)
[GetValue*](#)
[GetVariable*](#)
[GetViolatedElement*](#)
[GetViolationTime*](#)

GetLimit*

Returns the limit value of the i^{th} violation, given by vIdx.

```
double ScnFrт.GetLimit(int vIdx)
```

ARGUMENTS

vIdx vIdx > 0

RETURNS

The limit value of the i^{th} violation, given by vIdx. If vIdx is negative, zero or greater than the number of violations, then the function returns 'nan'.

EXAMPLE

See example [ScnFreq.GetViolationTime\(\)](#).

GetNumberOfViolations*

Returns the number of violations.

```
int ScnFrт.GetNumberOfViolations()
```

RETURNS

The number of violations.

EXAMPLE

See example [ScnFreq.GetViolationTime\(\)](#).

GetValue*

Returns the i^{th} value, given by vIdx, which causes the violation.

```
double ScnFrт.GetValue(int vIdx)
```

ARGUMENTS

vldx *vldx > 0*

RETURNS

The *ith* value, given by *vldx*, which causes the violation. If *vldx* is negative, zero or greater than the number of violations, then the function returns 'nan'.

EXAMPLE

See example of [ScnFreq.GetViolationTime\(\)](#).

GetVariable*

Returns the name of the variable of the *ith* violation, given by *vldx*.

```
string ScnFrt.GetVariable(int vIdx)
```

ARGUMENTS

vldx *vldx > 0*

RETURNS

The name of the variable of the *ith* violation, given by *vldx*. If *vldx* is negative, zero or greater than the number of violations, then the function returns "NoVariable".

EXAMPLE

See example [ScnFreq.GetViolationTime\(\)](#).

GetViolatedElement*

Returns the element of the *ith* violation, given by *vldx*.

```
object ScnFrt.GetViolatedElement(int vIdx)
```

ARGUMENTS

vldx *vldx > 0*

RETURNS

The element of the *ith* violation, given by *vldx*. If *vldx* is negative, zero or greater than the number of violations, then the function returns NULL.

EXAMPLE

See example [ScnFreq.GetViolationTime\(\)](#).

GetViolationTime*

Returns the time (in seconds) of the *ith* violation, given by *vldx*.

```
double ScnFrt.GetViolationTime(int vIdx)
```

ARGUMENTS

vldx *vldx > 0*

RETURNS

The time (in seconds) of the i^{th} violation, given by vIdx. If vIdx is negative, zero or greater than the number of violations, then the function returns 'nan'.

EXAMPLE

See example [ScnFreq.GetViolationTime\(\)](#).

3.6.54 ScnSpeed

Overview

[GetLimit*](#)
[GetNumberOfViolations*](#)
[GetValue*](#)
[GetVariable*](#)
[GetViolatedElement*](#)
[GetViolationTime*](#)

GetLimit*

Returns the limit value (in p.u.) of the i^{th} violation, given by vIdx.

```
double ScnSpeed.GetLimit(int vIdx)
```

ARGUMENTS

vIdx vIdx > 0

RETURNS

The limit value (in p.u.) of the i^{th} violation, given by vIdx. If vIdx is negative, zero or greater than the number of violations, then the function returns 'nan'.

EXAMPLE

See example [ScnFreq.GetViolationTime\(\)](#).

GetNumberOfViolations*

Returns the number of violations.

```
int ScnSpeed.GetNumberOfViolations()
```

RETURNS

The number of violations.

EXAMPLE

See example [ScnFreq.GetViolationTime\(\)](#).

GetValue*

Returns the i^{th} value (in p.u.), given by vIdx, which causes the violation.

```
double ScnSpeed.GetValue(int vIdx)
```

ARGUMENTS

vldx vldx > 0

RETURNS

The i^{th} value (in p.u.), given by vldx, which causes the violation. If vldx is negative, zero or greater than the number of violations, then the function returns 'nan'.

EXAMPLE

See example [ScnFreq.GetViolationTime\(\)](#).

GetVariable*

Returns the name of the variable of the i^{th} violation, given by vldx.

```
string ScnSpeed.GetVariable(int vIdx)
```

ARGUMENTS

vldx vldx > 0

RETURNS

The name of the variable of the i^{th} violation, given by vldx. If vldx is negative, zero or greater than the number of violations, then the function returns "NoVariable".

EXAMPLE

See example [ScnFreq.GetViolationTime\(\)](#).

GetViolatedElement*

Returns the element of the i^{th} violation, given by vldx.

```
object ScnSpeed.GetViolatedElement(int vIdx)
```

ARGUMENTS

vldx vldx > 0

RETURNS

The element of the i^{th} violation, given by vldx. If vldx is negative, zero or greater than the number of violations, then the function returns NULL.

EXAMPLE

See example [ScnFreq.GetViolationTime\(\)](#).

GetViolationTime*

Returns the time (in seconds) of the i^{th} violation, given by vldx.

```
double ScnSpeed.GetViolationTime(int vIdx)
```

ARGUMENTS

vldx vldx > 0

RETURNS

The time (in seconds) of the i^{th} violation, given by vldx. If vldx is negative, zero or greater than the number of violations, then the function returns 'nan'.

EXAMPLE

See example [ScnFreq.GetViolationTime\(\)](#).

3.6.55 ScnSync

Overview

[GetLimit*](#)
[GetNumberOfViolations*](#)
[GetValue*](#)
[GetVariable*](#)
[GetViolatedElement*](#)
[GetViolationTime*](#)

GetLimit*

Returns the limit value of the i^{th} violation, given by vldx.

```
double ScnSync.GetLimit(int vIdx)
```

ARGUMENTS

vIdx vIdx > 0

RETURNS

The limit value of the i^{th} violation, given by vldx. If vldx is negative, zero or greater than the number of violations, then the function returns 'nan'.

EXAMPLE

See example [ScnFreq.GetViolationTime\(\)](#).

GetNumberOfViolations*

Returns the number of violations.

```
int ScnSync.GetNumberOfViolations()
```

RETURNS

The number of violations.

EXAMPLE

See example [ScnFreq.GetViolationTime\(\)](#).

GetValue*

Returns the i^{th} value, given by vldx, which causes the violation.

```
double ScnSync.GetValue(int vIdx)
```

ARGUMENTS

vIdx vIdx > 0

RETURNS

The i^{th} value, given by vIdx, which causes the violation. If vIdx is negative, zero or greater than the number of violations, then the function returns 'nan'.

EXAMPLE

See example [ScnFreq.GetViolationTime\(\)](#).

GetVariable*

Returns the name of the variable of the i^{th} violation, given by vIdx.

```
string ScnSync.GetVariable(int vIdx)
```

ARGUMENTS

vIdx vIdx > 0

RETURNS

The name of the variable of the i^{th} violation, given by vIdx. If vIdx is negative, zero or greater than the number of violations, then the function returns "NoVariable".

EXAMPLE

See example [ScnFreq.GetViolationTime\(\)](#).

GetViolatedElement*

Returns the element of the i^{th} violation, given by vIdx.

```
object ScnSync.GetViolatedElement(int vIdx)
```

ARGUMENTS

vIdx vIdx > 0

RETURNS

The element of the i^{th} violation, given by vIdx. If vIdx is negative, zero or greater than the number of violations, then the function returns NULL.

EXAMPLE

See example [ScnFreq.GetViolationTime\(\)](#).

GetViolationTime*

Returns the time (in seconds) of the i^{th} violation, given by vIdx.

```
double ScnSync.GetViolationTime(int vIdx)
```

ARGUMENTS

vIdx vIdx > 0

RETURNS

The time (in seconds) of the i^{th} violation, given by vIdx. If vIdx is negative, zero or greater than the number of violations, then the function returns 'nan'.

EXAMPLE

See example [ScnFreq.GetViolationTime\(\)](#).

3.6.56 ScnVar

Overview

[GetLimit*](#)
[GetNumberOfViolations*](#)
[GetValue*](#)
[GetVariable*](#)
[GetViolatedElement*](#)
[GetViolationTime*](#)

GetLimit*

Returns the limit value of the i^{th} violation, given by vIdx.

```
double ScnVar.GetLimit(int vIdx)
```

ARGUMENTS

vIdx vIdx > 0

RETURNS

The limit value of the i^{th} violation, given by vIdx. If vIdx is negative, zero or greater than the number of violations, then the function returns 'nan'.

EXAMPLE

See example [ScnFreq.GetViolationTime\(\)](#).

GetNumberOfViolations*

Returns the number of violations.

```
int ScnVar.GetNumberOfViolations()
```

RETURNS

The number of violations.

EXAMPLE

See example [ScnFreq.GetViolationTime\(\)](#).

GetValue*

Returns the i^{th} value, given by vIdx, which causes the violation.

```
double ScnVar.GetValue(int vIdx)
```

ARGUMENTS

vldx *vldx > 0*

RETURNS

The *ith* value, given by *vldx*, which causes the violation. If *vldx* is negative, zero or greater than the number of violations, then the function returns 'nan'.

EXAMPLE

See example [ScnFreq.GetViolationTime\(\)](#).

GetVariable*

Returns the name of the variable of the *ith* violation, given by *vldx*.

```
string ScnVar.GetVariable(int vIdx)
```

ARGUMENTS

vldx *vldx > 0*

RETURNS

The name of the variable of the *ith* violation, given by *vldx*. If *vldx* is negative, zero or greater than the number of violations, then the function returns "NoVariable".

EXAMPLE

See example [ScnFreq.GetViolationTime\(\)](#).

GetViolatedElement*

Returns the element of the *ith* violation, given by *vldx*.

```
object ScnVar.GetViolatedElement(int vIdx)
```

ARGUMENTS

vldx *vldx > 0*

RETURNS

The element of the *ith* violation, given by *vldx*. If *vldx* is negative, zero or greater than the number of violations, then the function returns NULL.

EXAMPLE

See example [ScnFreq.GetViolationTime\(\)](#).

GetViolationTime*

Returns the time (in seconds) of the *ith* violation, given by *vldx*.

```
double ScnVar.GetViolationTime(int vIdx)
```

ARGUMENTS

vldx *vldx > 0*

RETURNS

The time (in seconds) of the i^{th} violation, given by vIdx. If vIdx is negative, zero or greater than the number of violations, then the function returns 'nan'.

EXAMPLE

See example [ScnFreq.GetViolationTime\(\)](#).

3.6.57 ScnVolt

Overview

[GetLimit*](#)
[GetNumberOfViolations*](#)
[GetValue*](#)
[GetVariable*](#)
[GetViolatedElement*](#)
[GetViolationTime*](#)

GetLimit*

Returns the limit value (in p.u.) of the i^{th} violation, given by vIdx.

```
double ScnVolt.GetLimit(int vIdx)
```

ARGUMENTS

vIdx vIdx > 0

RETURNS

The limit value (in p.u.) of the i^{th} violation, given by vIdx. If vIdx is negative, zero or greater than the number of violations, then the function returns 'nan'.

EXAMPLE

See example [ScnFreq.GetViolationTime\(\)](#).

GetNumberOfViolations*

Returns the number of violations.

```
int ScnVolt.GetNumberOfViolations()
```

RETURNS

The number of violations.

EXAMPLE

See example [ScnFreq.GetViolationTime\(\)](#).

GetValue*

Returns the i^{th} value (in p.u.), given by vIdx, which causes the violation.

```
double ScnVolt.GetValue(int vIdx)
```

ARGUMENTS

vIdx *vIdx > 0*

RETURNS

The *ith* value (in p.u.), given by *vIdx*, which causes the violation. If *vIdx* is negative, zero or greater than the number of violations, then the function returns 'nan'.

EXAMPLE

See example [ScnFreq.GetViolationTime\(\)](#).

GetVariable*

Returns the name of the variable of the *ith* violation, given by *vIdx*.

```
string ScnVolt.GetVariable(int vIdx)
```

ARGUMENTS

vIdx *vIdx > 0*

RETURNS

The name of the variable of the *ith* violation, given by *vIdx*. If *vIdx* is negative, zero or greater than the number of violations, then the function returns "NoVariable".

EXAMPLE

See example [ScnFreq.GetViolationTime\(\)](#).

GetViolatedElement*

Returns the element of the *ith* violation, given by *vIdx*.

```
object ScnVolt.GetViolatedElement(int vIdx)
```

ARGUMENTS

vIdx *vIdx > 0*

RETURNS

The element of the *ith* violation, given by *vIdx*. If *vIdx* is negative, zero or greater than the number of violations, then the function returns NULL.

EXAMPLE

See example [ScnFreq.GetViolationTime\(\)](#).

GetViolationTime*

Returns the time (in seconds) of the *ith* violation, given by *vIdx*.

```
double ScnVolt.GetViolationTime(int vIdx)
```

ARGUMENTS

vIdx *vIdx > 0*

RETURNS

The time (in seconds) of the i^{th} violation, given by vldx. If vldx is negative, zero or greater than the number of violations, then the function returns 'nan'.

EXAMPLE

See example [ScnFreq.GetViolationTime\(\)](#).

3.6.58 StoMaint

Overview

[SetElms](#)

SetElms

Sets the maintenance elements.

```
void StoMaint.SetElms(object singleElement)
void StoMaint.SetElms(set multipleElements)
```

ARGUMENTS

singleElement

single Element for Maintenance

multipleElements

multiple Elements for Maintenance

3.6.59 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 TypAsm.CalcElParams()
```

RETURNS

- 0** Calculated successfully.
- 1** Error.

3.6.60 TypCtcore

Overview

AddRatio
RemoveRatio
RemoveRatioByIndex

AddRatio

Adds the given ratio to the core. The ratio is added so that the sorting remains in ascending order. The accuracy parameters will be copied from the previous row. If the new ratio is to be the first ratio, the accuracy parameters will be copied from the next row instead.

```
int TypCtcore.AddRatio(double ratio)
```

ARGUMENTS

ratio New ratio to add.

RETURNS

0 New ratio was added.
1 An error occurred.

RemoveRatio

Removes the given ratio from the core. The last remaining ratio can never be removed.

```
int TypCtcore.RemoveRatio(double ratio)
```

ARGUMENTS

ratio Ratio to remove.

RETURNS

0 Ratio was removed.
1 An error occurred.

RemoveRatioByIndex

Removes the ratio with the given index from the core. The index is zero based. The last remaining ratio can never be removed.

```
int TypCtcore.RemoveRatio(int index)
```

ARGUMENTS

index Index to remove.

RETURNS

0 Index was removed.
1 An error occurred.

3.6.61 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

EXAMPLE

The following example reports all cable types.

```
set cables;
object type;
int i;
cables = GetCalcRelevantObjects();
type = cables.Firstmatch('TypLne');
while (type) {
    i = type.IsCable();
    if (i) type.ShowFullName();
    type = cables.Nextmatch();
}
```

3.6.62 TypQdsI

Overview

[Encrypt](#)

[IsEncrypted*](#)

[ResetThirdPartyModule](#)

[SetThirdPartyModule](#)

Encrypt

Encrypts a type. An encrypted type can be used without password but decrypted only with password. If no password is given a 'Choose Password' dialog appears.

```
int TypQdsI.Encrypt([string password = ''],
                     [int removeObjectHistory = 1],
                     [int masterCode = 0])
```

ARGUMENTS

password (optional)

Password for decryption. If no password is given a 'Choose Password' dialog appears.

removeObjectHistory (optional)

Handling of unencrypted object history in database, e.g. used by project versions or by undo:

- 0** Do not remove.
- 1** Do remove (default).
- 2** Show dialog and ask.

masterCode (optional)

Used for re-selling types. Third party licence codes already set in the type will be overwritten by this value (default = 0).

RETURNS

- 0** On success.
- 1** On error.

SEE ALSO

[TypQdsl.IsEncrypted\(\)](#)

IsEncrypted*

Returns the encryption state of the type.

```
int TypQdsl.IsEncrypted()
```

RETURNS

- 1** Type is encrypted.
- 0** Type is not encrypted.

SEE ALSO

[TypQdsl.Encrypt\(\)](#)

ResetThirdPartyModule

Resets the third party licence. Only possible for non-encrypted models. Requires masterkey licence for third party module currently set.

```
int TypQdsl.ResetThirdPartyModule()
```

RETURNS

- 0** On success.
- 1** On error.

SetThirdPartyModule

Sets the third party licence to a specific value. Only possible for non-encrypted models with no third party licence set so far. Requires masterkey licence for third party module to be set.

```
int TypQdsl.SetThirdPartyModule(string companyCode,
                                string moduleCode
                               )
```

ARGUMENTS

companyCode

D isplay name or numeric value of company code.

moduleCode

D isplay name or numeric value of third party module.

RETURNS

- 0** On success.
- 1** On error.

3.6.63 TypTr2**Overview**[GetZeroSequenceHVLVT*](#)**GetZeroSequenceHVLVT***

Returns the calculated star equivalent of the zero sequence impedances.

```
int TypTr2.GetZeroSequenceHVLVT(double& hvReal,
                                 double& hvImag,
                                 double& lvReal,
                                 double& lvImag,
                                 double& tReal ,
                                 double& tImag
                               )
```

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.

3.6.64 VisBdia

Overview

AddObjs
 AddResObjs
 Clear
 SetScaleY
 SetXVariable
 SetYVariable

AddObjs

Adds objects to elements column in table 'Bars'.

```
void VisBdia.AddObjs(set elements)
```

ARGUMENTS

elements Elements to add in table.

EXAMPLE

The following example gets a distortion analysis diagram and adds all elements of the selection

```
object page;
object graphBoard;
object plot;
set elements;
graphBoard=GetGraphBoard(); ! Look for open graphics board.
if (graphBoard) {
    page=graphBoard.GetPage('Distortion',1); ! get panel named Distortion
    if (page) {
        plot=page.GetVI('Analysis','VisBdia',1); ! get distortion analysis plot
        if (plot) {
            elements = SEL.All();
            plot.AddObjs(elements);
        }
    }
}
```

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.

```
void VisBdia.AddResObjs(object resultFileObj,
                        set elements
                      )
```

ARGUMENTS

resultFileObj

The result file to assign. Must be an object of class ElmRes.

elements Elements to add in table.

EXAMPLE

The following example gets a distortion analysis diagram and adds all elements of the selection with result file taken from study case.

```
object page;
object graphBoard;
object plot;
object resFile;
set elements;
graphBoard=GetGraphBoard(); ! Look for open graphics board.
if (graphBoard) {
    page=graphBoard.GetPage('Distortion',1); ! get panel named Distortion
    if (page) {
        plot=page.GetVI('Analysis','VisBdia',1); ! get distortion analysis plot
        if (plot) {
            resFile = GetFromStudyCase('ElmRes');
            elements = SEL.All();
            plot.AddResObjs(elements);
        }
    }
}
```

Clear

Removes all elements from plot by erasing all rows from the table named 'Bars'.

```
void VisBdia.Clear()
```

EXAMPLE

The following example gets the distortion analysis diagram and removes all elements.

```
object page;
object graphBoard;
object plot;
graphBoard=GetGraphBoard(); ! Look for open graphics board.
if (graphBoard) {
    page=graphBoard.GetPage('Distortion',1); ! get panel named Distortion
    if (page) {
        plot=page.GetVI('Analysis','VisBdia',1); ! get distortion analysis plot
        if (plot) {
            plot.Clear();
        }
    }
}
```

SetScaleY

Sets y-axis scale limits.

```
void VisBdia.SetScaleY(double min,
                      double max,
                      [int log]
                      )
```

ARGUMENTS

min (optional)

Minimum of y-scale.

max (optional)

Maximum of y-scale.

log (optional)

Possible values:

0 linear**1** logarithmic

EXAMPLE

The following example gets a distortion analysis diagram and sets the y-axis scale to a logarithmic scale in the range of [1,1000]

```
object page;
object graphBoard;
object plot;
object scale;
graphBoard=GetGraphBoard(); ! Look for open graphics board.
if (graphBoard) {
    page=graphBoard.GetPage('Distortion',1); ! get panel named Distortion
    if (page) {
        plot=page.GetVI('Analysis','VisBdia',1); ! get distortion analysis plot
        if (plot) {
            plot.setScaleY(1.,1000.,1);
        }
    }
}
```

SetXVariable

Set the x-axis Variable of the Distortion Analysis Diagram

```
int VisBdia.SetXVariable(string 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,

EXAMPLE

The following example gets a distortion analysis diagram and sets the x-axis variable to 'order'.

```
object page;
object graphBoard;
object plot;
object scale;
graphBoard=GetGraphBoard(); ! Look for open graphics board.
if (graphBoard) {
    page=graphBoard.GetPage('Distortion',1); ! get panel named Distortion
```

```

if (page) {
    plot=page.GetVI('Analysis','VisBdia',1); ! get distortion analysis plot
    if (plot) {
        plot.SetXVariable('order');
    }
}
}

```

SetYVariable

Set the y-axis variable of the Distortion Analysis Diagram

```
int VisBdia.SetYVariable(string 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,

EXAMPLE

The following example gets a distortion analysis diagram and sets the y-axis variable to 'm:u:bu1'.

```

object page;
object graphBoard;
object plot;
object scale;
graphBoard=GetGraphBoard(); ! Look for open graphics board.
if (graphBoard) {
    page=graphBoard.GetPage('Distortion',1); ! get panel named Distortion
    if (page) {
        plot=page.GetVI('Analysis','VisBdia',1); ! get distortion analysis plot
        if (plot) {
            plot.SetYVariable('m:u:bu1');
        }
    }
}

```

3.6.65 VisDraw

Overview

[AddRelay](#)
[AddRelays](#)
[CentreOrigin](#)
[Clear](#)
[DoAutoScaleOnAll](#)
[DoAutoScaleOnCharacteristics](#)
[DoAutoScaleOnImpedances](#)
[DoAutoScaleX](#)
[DoAutoScaleY](#)

AddRelay

Adds a relay to the plot and optionally sets the drawing style.

```
void VisDraw.AddRelay(object relay,
                      [double colour,]
                      [double style,]
                      [double width])
```

ARGUMENTS

relay The protection device 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.

```
void VisDraw.AddRelays(set relays)
```

ARGUMENTS

relays The protection devices (ElmRelay or RelFuse) to be added.

CentreOrigin

Centre the origin of the plot

```
void VisDraw.CentreOrigin()
```

Clear

Removes all protection devices from the plot.

```
void VisDraw.Clear()
```

DoAutoScaleOnAll

Scales the plot automatically under consideration of relay characteristics, simulation curves and short circuit arrows. The function works for local scales only.

```
int VisDraw.DoAutoScaleOnAll()
```

RETURNS

- 0** Automatic scaling was executed.
- 1** An Error occurred.

DoAutoScaleOnCharacteristics

Scales the plot automatically under consideration of relay characteristics. Same as button named Characteristics. The function works for local scales only.

```
int VisDraw.DoAutoScaleOnCharacteristics()
```

RETURNS

- 0** Automatic scaling was executed.
- 1** An Error occurred.

DoAutoScaleOnImpedances

Scales the plot automatically under consideration of branch impedances. Same as button named Impedances. The function works for local scales only.

```
int VisDraw.DoAutoScaleOnImpedances()
```

RETURNS

- 0** Automatic scaling was executed.
- 1** An Error occurred.

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.

3.6.66 VisHrm

Overview

Clear
 DoAutoScaleX
 DoAutoScaleY
 GetDataSource
 GetScaleObjX
 GetScaleObjY
 SetAutoScaleX
 SetAutoScaleY
 SetCrvDesc
 SetDefScaleX
 SetDefScaleY

Clear

Removes all curves by clearing table named 'Curves'.

```
void VisHrm.Clear()
```

EXAMPLE

The following example gets the waveform plot and removes all curves.

```
object page;
object graphBoard;
object plot;
graphBoard=GetGraphBoard(); ! Look for open graphics board.
if (graphBoard) {
    page=graphBoard.GetPage('Distortion',1); ! get panel named Distortion
    if (page) {
        plot=page.GetVI('Analysis','VisHrm',1); ! get waveform plot
        if (plot) {
            plot.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

EXAMPLE

The following example gets the waveform plot and scales the x-axis automatically

```
object page;
object graphBoard;
object plot;
```

```

graphBoard=GetGraphBoard(); ! Look for open graphics board.
if (graphBoard) {
    page=graphBoard.GetPage('Waveform',1); ! get panel named 'Waveform'
    if (page) {
        plot=page.GetVI('Analysis','VisHrm',1); ! get waveform plot
        if (plot) {
            plot.DoAutoScaleX();
        }
    }
}

```

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

EXAMPLE

The following example gets the waveform plot and scales the y-axis automatically

```

object page;
object graphBoard;
object plot;
graphBoard=GetGraphBoard(); ! Look for open graphics board.
if (graphBoard) {
    page=graphBoard.GetPage('Waveform',1); ! get panel named Distortion
    if (page) {
        plot=page.GetVI('Analysis','VisHrm',1); ! get waveform plot
        if (plot) {
            plot.DoAutoScaleY();
        }
    }
}

```

GetDataSource

Get data source for curve with index

```
object VisHrm.GetDataSource(int curveIndex)
```

ARGUMENTS

int curveIndex

Possible values:

- < **0** invalid
- ≥ 0 curve index in table

RETURNS

object Data source

NULL No data source found

EXAMPLE

The following example looks for a plot and gets the data source for the first curve.

```
object dataSource;
object page;
object graphBoard;
object plot;
graphBoard=GetGraphBoard(); ! Look for open graphics board.
if (graphBoard) {
    page=graphBoard.GetPage('Voltage',1); ! get panel named Voltage
    if (page) {
        plot=page.GetVI('RST','VisHrm',1); ! get plot
        if (plot) {
            dataSource = plot.GetDataSource(0);
        }
    }
}
```

GetScaleObjX

Gets the object used for scaling the x-axis.

```
object 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'.

EXAMPLE

The following example gets the object specifying the x-scale of the waveform plot.

```
object page;
object graphBoard;
object plot;
object scaleObjX;
graphBoard=GetGraphBoard(); ! Look for open graphics board.
if (graphBoard) {
    page=graphBoard.GetPage('Waveform',1); ! get panel named Distortion
    if (page) {
        plot=page.GetVI('Analysis','VisHrm',1); ! get waveform plot
        if (plot) {
            scaleObjX = plot.GetScaleObjX();
            scaleObjX.ShowFullName();
        }
    }
}
```

GetScaleObjY

Gets the object used for scaling the y-axis.

```
object VisHrm.GetScaleObjY()
```

RETURNS

this object In case that 'Use local Axis' is enabled.

the plot type In case that 'Use local axis' is disabled.

EXAMPLE

The following example gets the object specifying the y-scale of the waveform plot.

```
object page;
object graphBoard;
object plot;
object scaleObjY;
graphBoard=GetGraphBoard(); ! Look for open graphics board.
if (graphBoard) {
    page=graphBoard.GetPage('Waveform',1); ! get panel named Distortion
    if (page) {
        plot=page.GetVI('Analysis','VisHrm',1); ! get waveform plot
        if (plot) {
            scaleObjY = plot.GetScaleObjY();
            scaleObjY.ShowFullName();
        }
    }
}
```

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.

```
void VisHrm.SetAutoScaleX(int mode)
```

ARGUMENTS

mode Possible values:

- | | |
|---|------------------|
| 0 | never |
| 1 | after simulation |

EXAMPLE

The following example gets the waveform plot and sets the Auto Scale option of the x-axis to On.

```
object page;
object graphBoard;
object plot;
graphBoard=GetGraphBoard(); ! Look for open graphics board.
if (graphBoard) {
    page=graphBoard.GetPage('Waveform',1); ! get panel named Distortion
    if (page) {
        plot=page.GetVI('Analysis','VisHrm',1); ! get waveform plot
        if (plot) {
            plot.SetAutoScaleX(1);
        }
    }
}
```

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.

```
void VisHrm.SetAutoScaleY(int mode)
```

ARGUMENTS

mode Possible values:

- 0** never
- 1** after simulation

EXAMPLE

The following example gets the waveform plot and sets the Auto Scale option of the y-axis to On.

```
object page;
object graphBoard;
object plot;
graphBoard=GetGraphBoard(); ! Look for open graphics board.
if (graphBoard) {
    page=graphBoard.GetPage('Waveform',1); ! get panel named Distortion
    if (page) {
        plot=page.GetVI('Analysis','VisHrm',1); ! get waveform plot
        if (plot) {
            plot.SetAutoScaleY(1);
        }
    }
}
```

SetCrvDesc

Sets the user defined description of a curve.

```
void VisHrm.SetCrvDesc(int curveIndex, string curveDescription)
```

ARGUMENTS

curveIndex

Curve index; first curve in table is index 1.

curveDescription

Description to set

EXAMPLE

The following example gets the waveform plot and sets the description of the first variable to example.

```
object page;
object graphBoard;
object plot;
graphBoard=GetGraphBoard(); ! Look for open graphics board.
if (graphBoard) {
    page=graphBoard.GetPage('Waveform',1); ! get panel named Waveform
    if (page) {
        plot=page.GetVI('Analysis','VisHrm',1); ! get waveform plot
```

```
    if (plot) {
        plot.SetCrvDesc(1, 'example');
    }
}
```

SetDefScaleX

Sets the x-scale to be used to the graphics board.

```
void VisHrm.SetDefScaleX()
```

EXAMPLE

The following example gets the waveform plot and sets the x-scale to the graphics board.

```
object page;
object graphBoard;
object plot;
graphBoard=GetGraphBoard(); ! Look for open graphics board.
if (graphBoard) {
    page=graphBoard.GetPage('Waveform',1); ! get panel named Distortion
    if (page) {
        plot=page.GetVI('Analysis','VisHrm',1); ! get waveform plot
        if (plot) {
            plot.SetDefScaleX();
        }
    }
}
```

SetDefScaleY

Sets the y-scale to be used to the plot type.

```
void VisHrm.SetDefScaleY()
```

EXAMPLE

The following example gets the waveform plot and sets the y-scale to the plot type.

```
object page;
object graphBoard;
object plot;
graphBoard=GetGraphBoard(); ! Look for open graphics board.
if (graphBoard) {
    page=graphBoard.GetPage('Waveform',1); ! get panel named Distortion
    if (page) {
        plot=page.GetVI('Analysis','VisHrm',1); ! get waveform plot
        if (plot) {
            plot.SetDefScaleY();
        }
    }
}
```

3.6.67 VisMagndiffplt

Overview

AddRelay
 AddRelays
 Clear
 DoAutoSizeX
 DoAutoSizeY
 Refresh

AddRelay

Adds a relay to the plot and optionally sets the drawing style.

```
void VisMagndiffplt.AddRelay(object relay,
                               [double colour,]
                               [double style,]
                               [double width])
```

ARGUMENTS

relay The protection device 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.

```
void VisMagndiffplt.AddRelays(set relays)
```

ARGUMENTS

relays The protection devices (ElmRelay or RelFuse) to be added.

Clear

Removes all protection devices from the plot.

```
void VisMagndiffplt.Clear()
```

DoAutoSizeX

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.DoAutoSizeX()
```

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.

```
void VisMagndiffplt.Refresh()
```

3.6.68 VisOcplot**Overview**

[AddRelay](#)
[AddRelays](#)
[Clear](#)
[DoAutoScaleX](#)
[DoAutoScaleY](#)
[Refresh](#)

AddRelay

Adds a relay to the plot and optionally sets the drawing style.

```
void VisOcplot.AddRelay(object relay,
                        [double colour,]
                        [double style,]
                        [double width])
```

ARGUMENTS

relay The protection device 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.

```
void VisOcplot.AddRelays(set relays)
```

ARGUMENTS

relays The protection devices (ElmRelay or RelFuse) to be added.

Clear

Removes all protection devices from the plot.

```
void 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.

```
void VisOcplot.Refresh()
```

3.6.69 VisPath

Overview

Clear
 DoAutoSizeX
 DoAutoSizeY
 SetAdaptX
 SetAdaptY
 SetScaleX
 SetScaleY

Clear

Removes all curves by clearing table named 'Variables' on page 'Curves'.

```
void VisPath.Clear()
```

EXAMPLE

The following example gets the plot and removes all curves.

```
object page;
object graphBoard;
object plot;
graphBoard=GetGraphBoard(); ! Look for open graphics board.
if (graphBoard) {
    page=graphBoard.GetPage('Voltage',1); ! get panel named Voltage
    if (page) {
        plot=page.GetVI('Analysis','VisPath',1); ! get plot
        if (plot) {
            plot.Clear();
        }
    }
}
```

DoAutoSizeX

Scales x-axis automatically.

```
int VisPath.DoAutoSizeX()
```

RETURNS

Always 0

EXAMPLE

The following example gets the plot and scales the x-axis automatically

```
object page;
object plot;
object graphBoard;
graphBoard=GetGraphBoard(); ! Look for open graphics board.
if (graphBoard) {
    page=graphBoard.GetPage('Voltage',1); ! get panel named 'Voltage'
    if (page) {
        plot=page.GetVI('Analysis','VisPath',1); ! get plot
        if (plot) {
```

```

        plot.DoAutoScaleX();
    }
}
}
```

DoAutoScaleY

Scales y-axis automatically.

```
int VisPath.DoAutoScaleY()
```

RETURNS

Always 0

EXAMPLE

The following example gets the plot and scales the x-axis automatically

```

object page;
object graphBoard;
graphBoard=GetGraphBoard(); ! Look for open graphics board.
if (graphBoard) {
    page=graphBoard.GetPage('Voltage',1); ! get panel named 'Voltage'
    if (page) {
        plot=page.GetVI('Analysis','VisPath',1); ! get plot
        if (plot) {
            plot.DoAutoScaleY();
        }
    }
}
```

SetAdaptX

Sets the Adapt Scale option of the x-scale.

```
void VisPath.SetAdaptX(int mode)
```

ARGUMENTS

mode Possible values:

0	off
1	on

EXAMPLE

The following example gets the plot and sets Adapt Scale of the x-axis to On

```

object graphBoard;
object page;
object plot;
graphBoard=GetGraphBoard(); ! Look for open graphics board.
if (graphBoard) {
    page=graphBoard.GetPage('Voltage',1); ! get panel named 'Voltage'
    if (page) {
        plot=page.GetVI('Analysis','VisPath',1); ! get plot

```

```
    if (plot) {
        plot.SetAdaptX(1);
    }
}
```

SetAdaptY

Sets the Adapt Scale option of the x-scale.

```
void VisPath.SetAdaptY(int mode)
```

ARGUMENTS

mode Possible values:

0 off
1 on

EXAMPLE

The following example gets the plot and sets Adapt Scale of the y-axis to On

```
object graphBoard;
object page;
object plot;
graphBoard=GetGraphBoard(); ! Look for open graphics board.
if (graphBoard) {
    page=graphBoard.GetPage('Voltage',1); ! get panel named 'Voltage'
    if (page) {
        plot=page.GetVI('Analysis','VisPath',1); ! get plot
        if (plot) {
            plot.SetAdaptY(1);
        }
    }
}
```

SetScaleX

Sets x-axis scale.

ARGUMENTS

min Minimum of x-scale.

max Maximum of x-scale.

EXAMPLE

The following example sets the minimum to 1 and maximum to 10.

```

object graphBoard;
object page;
object plot;
graphBoard=GetGraphBoard(); ! Look for open graphics board.
if (graphBoard) {
    page=graphBoard.GetPage('Voltage',1); ! get panel named 'Voltage'
    if (page) {
        plot=page.GetVI('Analysis','VisPath',1); ! get plot
        if (plot) {
            plot.SetScaleX(1,10);
        }
    }
}

```

SetScaleY

Sets y-axis scale limits.

```

void VisPath.setScaleY (double min,
                      double max,
                      [int log]
)

```

ARGUMENTS

min Minimum of y-scale.

max Maximum of y-scale.

log (optional)

Possible values:

0 linear

1 logarithmic

EXAMPLE

The following example gets the plot and sets the y-axis scale to a logarithmic scale in the range of [1,1000]

```

object page;
object graphBoard;
object plot;
object scale;
graphBoard=GetGraphBoard(); ! Look for open graphics board.
if (graphBoard) {
    page=graphBoard.GetPage('Voltage',1); ! get panel named Distortion
    if (page) {
        plot=page.GetVI('Analysis','VisPath',1); ! get distortion analysis plot
        if (plot) {
            plot.setScaleY(1.,1000.,1);
        }
    }
}

```

3.6.70 VisPcompdifffplt

Overview

AddRelay
 AddRelays
 CentreOrigin
 Clear
 DoAutoScaleX
 DoAutoScaleY

AddRelay

Adds a relay to the plot and optionally sets the drawing style.

```
void VisPcompdifffplt.AddRelay(object relay,
                                 [double colour,]
                                 [double style,]
                                 [double width])
```

ARGUMENTS

relay The protection device 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.

```
void VisPcompdifffplt.AddRelays(set relays)
```

ARGUMENTS

relays The protection devices (ElmRelay or RelFuse) to be added.

CentreOrigin

Centre the origin of the plot

```
void VisPcompdifffplt.CentreOrigin()
```

Clear

Removes all protection devices from the plot.

```
void VisPcompdifffplt.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 VisPcompdifffplt.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 VisPcompdifffplt.DoAutoScaleY()
```

RETURNS

- 0** Automatic scaling was executed.
- 1** An Error occurred.

3.6.71 VisPlot**Overview**

AddResVars
 AddVars
 Clear
 DoAutoScaleX
 DoAutoScaleY
 GetDataSource
 GetIntCalcres
 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.

```
void VisPlot.AddResVars(object elmRes,
                        object element,
                        string varname
                      )
```

```
)
```

ARGUMENTS

elmRes Result object, classname ElmRes.
element Element to add.
varname Variable name.

EXAMPLE

The following examples looks for a plot and adds a variable.

```
object graphBoard;
object page;
object plot;
object resObj;
graphBoard=GetGraphBoard(); ! Look for open desktop.
if (graphBoard) {
    page=graphBoard.GetPage('Voltage',1); ! Get panel named 'voltage'
    if (page) {
        plot=page.GetVI('RST','VisPlot',1); ! get plot
        if (plot) {
            resObj = GetFromStudyCase('*.*ElmRes');
            plot.AddResVars(resObj, line, 'c:loading');
        }
    }
}
```

AddVars

Appends variables to the plot. Variables which are already in the plot are not added.

```
void VisPlot.AddVars(object element,
                     string varname)
)
```

ARGUMENTS

element Element to add.
varname Variable name.

EXAMPLE

The following examples looks for a plot and adds a variable.

```
object graphBoard;
object page;
object plot;
graphBoard=GetGraphBoard(); ! Look for open desktop.
if (graphBoard) {
    page=graphBoard.GetPage('Voltage',1); ! Get panel named 'voltage'
    if (page) {
        plot=page.GetVI('RST','VisPlot',1); ! get plot
        if (plot) {
            plot.AddVars(line, 'c:loading');
        }
    }
}
```

Clear

Removes all curves from plot.

```
void VisPlot.Clear()
```

EXAMPLE

The following example gets the plot and removes all curves.

```
object page;
object graphBoard;
object plot;
graphBoard=GetGraphBoard(); ! Look for open graphics board.
if (graphBoard) {
    page=graphBoard.GetPage('Voltage',1); ! get panel
    if (page) {
        plot=page.GetVI('RST','VisPlot',1); ! get plot
        if (plot) {
            plot.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

EXAMPLE

The following example gets the plot and scales the x-axis automatically

```
object page;
object graphBoard;
object plot;
graphBoard=GetGraphBoard(); ! Look for open graphics board.
if (graphBoard) {
    page=graphBoard.GetPage('Voltage',1); ! get panel named 'Voltage'
    if (page) {
        plot=page.GetVI('U','VisPlot',1); ! get plot
        if (plot) {
            plot.DoAutoScaleX();
        }
    }
}
```

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

EXAMPLE

The following example gets the plot and scales the y-axis automatically

```
object page;
object graphBoard;
object plot;
graphBoard=GetGraphBoard(); ! Look for open graphics board.
if (graphBoard) {
    page=graphBoard.GetPage('Voltage',1); ! get panel named 'Voltage'
    if (page) {
        plot=page.GetVI('U','VisPlot',1); ! get plot
        if (plot) {
            plot.DoAutoScaleY();
        }
    }
}
```

GetDataSource

Get data source for curve with index

```
object VisPlot.GetDataSource(int curveIndex)
```

ARGUMENTS

int curveIndex

Possible values:

- < **0** invalid
- ≥ **0** curve index in table

RETURNS

object Data source

NULL No data source found

EXAMPLE

The following example looks for a plot and gets the data source for the first curve.

```
object dataSource;
object page;
object graphBoard;
object plot;
graphBoard=GetGraphBoard(); ! Look for open graphics board.
if (graphBoard) {
    page=graphBoard.GetPage('Voltage',1); ! get panel named Voltage
    if (page) {
        plot=page.GetVI('RST','VisPlot',1); ! get plot
```

```

        if (plot) {
            dataSource = plot.GetDataSource(0);
        }
    }
}

```

GetIntCalcres

Gets all user Calculated Result objects (IntCalcres) stored inside plot

```
object VisPlot.GetIntCalcres()
```

RETURNS

All Calculated Result objects (IntCalcres) stored inside plot.

GetScaleObjX

Gets the object used for scaling the x-axis.

```
object 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'.

EXAMPLE

The following example gets the object specifying the x-scale of plot.

```

object page;
object graphBoard;
object plot;
object scaleObjX;
graphBoard=GetGraphBoard(); ! Look for open graphics board.
if (graphBoard) {
    page=graphBoard.GetPage('Voltage',1); ! get panel named Voltage
    if (page) {
        plot=page.GetVI('RST','VisPlot',1); ! get plot
        if (plot) {
            scaleObjX = plot.GetScaleObjX();
            scaleObjX.ShowFullName();
        }
    }
}

```

GetScaleObjY

Gets the object used for scaling the y-axis.

```
object VisPlot.GetScaleObjY()
```

RETURNS

this object In case that 'Use local Axis' is enabled.
the plot type In case that 'Use local axis' is disabled.

EXAMPLE

The following example gets the object specifying the y-scale of the plot.

```
object page;
object graphBoard;
object plot;
object scaleObjY;
graphBoard=GetGraphBoard(); ! Look for open graphics board.
if (graphBoard) {
    page=graphBoard.GetPage('Voltage',1); ! get panel named Voltage
    if (page) {
        plot=page.GetVI('RST','VisPlot',1); ! get plot
        if (plot) {
            scaleObjY = plot.GetScaleObjY();
            scaleObjY.ShowFullName();
        }
    }
}
```

SetAdaptX

Sets the Adapt Scale option of the local x-scale.

```
void VisPlot.SetAdaptX(int mode,
                        [double trigger]
                        )
```

ARGUMENTS

mode Possible values:

0	off
1	on

trigger (optional)

Trigger value, unused if mode is off or empty

EXAMPLE

The following example looks for a plot and sets its adapt scale option.

```
object page;
object graphBoard;
object plot;
graphBoard=GetGraphBoard(); ! Look for open graphics board.
if (graphBoard) {
    page=graphBoard.GetPage('Voltage',1); ! get panel named Voltage
    if (page) {
        plot=page.GetVI('RST','VisPlot',1); ! get plot
        if (plot) {
            plot.SetAdaptX(1,3); ! Turn on adapt scale, use a trigger value of 3
            plot.SetAdaptX(0,3); ! Turn off adapt scale
            plot.SetAdaptX(1);   ! Turn on adapt scale again, do not change the trigger value
        }
    }
}
```

```
    }
}
```

SetAdaptY

Sets the Adapt Scale option of the local y-scale.

```
void VisPlot.SetAdaptY(int mode,
                      [double offset]
                      )
```

ARGUMENTS

mode Possible values:

- | | |
|----------|-----|
| 0 | off |
| 1 | on |

offset (optional)

Offset value, unused if mode is off or empty

EXAMPLE

The following example looks for a plot and sets its adapt scale option.

```
object page;
object graphBoard;
object plot;
graphBoard=GetGraphBoard(); ! Look for open graphics board.
if (graphBoard) {
    page=graphBoard.GetPage('Voltage',1); ! get panel named Voltage
    if (page) {
        plot=page.GetVI('RST','VisPlot',1); ! get plot
        if (plot) {
            plot.SetAdaptY(1,3); ! Turn on adapt scale, use an offset value of 3
            plot.SetAdaptY(0,3); ! Turn off adapt scale
            plot.SetAdaptY(1);   ! Turn on adapt scale again, do not change the offset value
        }
    }
}
```

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.

```
void VisPlot.SetAutoScaleX(int mode)
```

ARGUMENTS

mode Possible values:

- | | |
|----------|-------------------|
| 0 | never |
| 1 | after simulation |
| 2 | during simulation |

EXAMPLE

The following example gets the plot and sets the Auto Scale option of the x-axis to On.

```
object page;
object graphBoard;
object plot;
graphBoard=GetGraphBoard(); ! Look for open graphics board.
if (graphBoard) {
    page=graphBoard.GetPage('Voltage',1); ! get panel named Voltage
    if (page) {
        plot=page.GetVI('RST','VisPlot',1); ! get plot
        if (plot) {
            plot.SetAutoScaleX(1);
        }
    }
}
```

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.

```
void VisPlot.SetAutoScaleY(int mode)
```

ARGUMENTS

mode Possible values:

- 0** never
- 1** after simulation
- 2** during simulation

EXAMPLE

The following example gets the plot and sets the Auto Scale option of the y-axis to On.

```
object page;
object graphBoard;
object plot;
graphBoard=GetGraphBoard(); ! Look for open graphics board.
if (graphBoard) {
    page=graphBoard.GetPage('Voltage',1); ! get panel named Voltage
    if (page) {
        plot=page.GetVI('RST','VisPlot',1); ! get plot
        if (plot) {
            plot.SetAutoScaleY(1);
        }
    }
}
```

SetCrvDesc

Sets the user defined description of a curve.

```
void VisPlot.SetCrvDesc(int curveIndex,
                        string curveDescription
                      )
```

ARGUMENTS*curveIndex*

Curve index; first curve in table is index 1.

curveDescription

Description to set.

EXAMPLE

The following example gets a plot and sets the description of the first variable to example.

```
object page;
object graphBoard;
object plot;
graphBoard=GetGraphBoard(); ! Look for open graphics board.
if (graphBoard) {
    page=graphBoard.GetPage('Voltage',1); ! get panel named Voltage
    if (page) {
        plot=page.GetVI('Plot','VisPlot',1); ! get plot
        if (plot) {
            plot.SetCrvDesc(1,'example');
        }
    }
}
```

SetDefScaleX

Sets the x-scale to be used to the graphics board.

```
void VisPlot.SetDefScaleX()
```

EXAMPLE

The following example gets the plot and sets the x-scale to the graphics board.

```
object graphBoard;
object page;
object plot;
graphBoard=GetGraphBoard(); ! Look for open desktop.
if (graphBoard) {
    page=graphBoard.GetPage('Voltage',1); ! Get panel named 'Voltage'
    if (page) {
        plot=page.GetVI('RST','VisPlot',1); ! get plot
        if (plot) {
            plot.SetDefScaleX();
        }
    }
}
```

SetDefScaleY

Sets the y-scale to be used to the plot type.

```
void VisPlot.SetDefScaleY()
```

EXAMPLE

The following example gets the plot and sets the y-scale to the plot type.

```
object graphBoard;
object page;
object plot;
graphBoard=GetGraphBoard(); ! Look for open desktop.
if (graphBoard) {
    page=graphBoard.GetPage('Voltage',1); ! Get panel named 'Voltage'
    if (page) {
        plot=page.GetVI('RST','VisPlot',1); ! get plot
        if (plot) {
            plot.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.

```
void VisPlot.setScaleX()
void VisPlot.setScaleX(double min,
                      double max,
                      [int log]
                      )
```

ARGUMENTS

min (optional)

Minimum of x-scale.

max (optional)

Maximum of x-scale.

log (optional)

Possible values:

0 linear

1 logarithmic

EXAMPLE

The following examples look for a Subplot named 'RST' and set its x-scale. There are three different examples. 1. Example: Set x-axis to local and change 'Auto Scale' of x axis to 'On'. 2. Example: Set minimum to 0 and maximum to 20. 3. Example: Set minimum to 1 and maximum to 1000. Changes to a logarithmic scale.

```
! Set Auto Scale to On
object graphBoard;
object page;
object plot;
graphBoard=GetGraphBoard(); ! Look for open graphics board.
if (graphBoard) {
    page=graphBoard.GetPage('Voltage',1); ! get panel named 'Voltage'
    if (page) {
        plot=page.GetVI('RST','VisPlot',1); ! get plot named RST
        if (plot) {
```

```

        plot.SetScaleX(); ! set auto scale mode to On
    }
}
}

! Set minimum and maximum without changing linear/log
object graphBoard;
object page;
object plot;
graphBoard=GetGraphBoard(); ! Look for open graphics board.
if (graphBoard) {
    page=graphBoard.GetPage('Voltage',1); ! get panel named 'Voltage'
    if (page) {
        plot=page.GetVI('RST','VisPlot',1); ! get plot named RST
        if (plot) {
            plot.SetScaleX(0,20); ! Set minimum and maximum
        }
    }
}

! Set minimum and maximum, change to logarithmic scale
object graphBoard;
object page;
object plot;
graphBoard=GetGraphBoard(); ! Look for open graphics board.
if (graphBoard) {
    page=graphBoard.GetPage('Voltage',1); ! get panel named 'Voltage'
    if (page) {
        plot=page.GetVI('RST','VisPlot',1); ! get plot named RST
        if (plot) {

            plot.SetScaleX(1,1000,1); ! Set minimum and maximum, change to log scale
        }
    }
}
}

```

SetScaleY

Sets the local y-axis scale. A function call without arguments sets the Auto Scale setting to On without changing the scale itself.

```

void VisPlot.SetScaleY()
void VisPlot.SetScaleY(double min,
                      double max,
                      [int log]
)

```

ARGUMENTS

min (optional)

Minimum of y-scale.

max (optional)

Maximum of y-scale.

log (optional)

Possible values:

0 linear

1 logarithmic

EXAMPLE

The following examples look for a Subplot named 'RST' and set its y-scale. There are three different examples. 1. Example: Set y-axis to local and change 'Auto Scale' of y axis to 'On'. 2. Example: Set minimum to 0 and maximum to 20. 3. Example: Set minimum to 1 and maximum to 1000. Changes to a logarithmic scale.

```

! Set Auto Scale to On
object graphBoard;
object page;
object plot;
graphBoard=GetGraphBoard(); ! Look for open graphics board.
if (graphBoard) {
    page=graphBoard.GetPage('Voltage',1); ! get panel named 'Voltage'
    if (page) {
        plot=page.GetVI('RST','VisPlot',1); ! get plot named RST
        if (plot) {
            plot.setScaleY(); ! set auto scale mode to On
        }
    }
}

! Set minimum and maximum without changing linear/log
object graphBoard;
object page;
object plot;
graphBoard=GetGraphBoard(); ! Look for open graphics board.
if (graphBoard) {
    page=graphBoard.GetPage('Voltage',1); ! get panel named 'Voltage'
    if (page) {
        plot=page.GetVI('RST','VisPlot',1); ! get plot named RST
        if (plot) {
            plot.setScaleY(0,20); ! Set minimum and maximum
        }
    }
}

! Set minimum and maximum, change to logarithmic scale
object graphBoard;
object page;
object plot;
graphBoard=GetGraphBoard(); ! Look for open graphics board.
if (graphBoard) {
    page=graphBoard.GetPage('Voltage',1); ! get panel named 'Voltage'
    if (page) {
        plot=page.GetVI('RST','VisPlot',1); ! get plot named RST
        if (plot) {

            plot.setScaleY(1,1000,1); ! Set minimum and maximum, change to log scale
        }
    }
}

```

SetXVar

Sets the local x-axis variable. If The default x-axis variable (time) is set if no argument is passed.

```
void VisPlot.SetXVar()
void VisPlot.SetXVar(object obj,]
                     string varname
                     )
```

ARGUMENTS

obj (*optional*)
 x-axis object

varname (*optional*)
 variable of *obj*

EXAMPLE

The following examples look for a subplot named RST and set its x-axis variable. The first example sets an user defined x-axis variable of the plot. The second one sets the default x-axis variable (time).

```
! set x-axis variable 'm:U1:bus1' of object line
object graphBoard;
object page;
object plot;
graphBoard=GetGraphBoard(); ! Look for open desktop.
if (graphBoard) {
    page=graphBoard.GetPage('Voltage',1); ! Get panel named 'Voltage'
    if (page) {
        plot=page.GetVI('RST','VisPlot',1); ! get plot
        if (plot) {
            plot.SetXVar(line,'m:U1:bus1'); ! Set x-scale variable
        }
    }
}
! set default x-axis variable
object graphBoard;
object page;
object plot;
graphBoard=GetGraphBoard(); ! Look for open desktop.
if (graphBoard) {
    page=graphBoard.GetPage('Voltage',1); ! Get panel named 'Voltage'
    if (page) {
        plot=page.GetVI('RST','VisPlot',1); ! get plot
        if (plot) {
            plot.SetXVar(); ! Set default x-scale variable (time)
        }
    }
}
```

3.6.72 VisPlot2

Overview

[AddResVars](#)
[AddVars](#)
[Clear](#)
[DoAutoScaleX](#)
[DoAutoScaleY](#)

```

DoAutoSizeY2
GetDataSource
GetScaleObjX
GetScaleObjY
SetAdaptX
SetAdaptY
SetAutoSizeX
SetAutoSizeY
SetCrvDesc
SetDefScaleX
SetDefScaleY
SetScaleX
SetScaleY
SetXVar
ShowY2

```

AddResVars

Appends variables to the plot. Variables which are already in the plot are not added.

```

void VisPlot2.AddResVars(object elmRes
                         object element,
                         string varname,
                         [int y2]
                         )
)

```

ARGUMENTS

elmRes Result object, classanme ElmRes

element Element to add

varname Variable name

y2 (optional)

Possible values:

- 1 y1-axis, default value
- 2 y2 axis

EXAMPLE

The following examples looks for a plot and adds a variable to the second y axis

```

object graphBoard;
object page;
object plot;
object resObj;
graphBoard=GetGraphBoard(); ! Look for open desktop.
if (graphBoard) {
    page=graphBoard.GetPage('Voltage',1); ! Get panel named 'voltage'
    if (page) {
        plot=page.GetVI('RST','VisPlot2',1); ! get plot
        if (plot) {
            resObj = GetFromStudyCase('*.'ElmRes');
            plot.AddResVars(resObj, line,'c:loading',2);
        }
    }
}

```

AddVars

Appends variables to the plot. Variables which are already in the plot are not added.

```
void VisPlot2.AddVars(object element,
                      string varname,
                      [int y2]
)
)
```

ARGUMENTS

element Element to add

varname Variable name

y2 (optional)

Possible values:

- 1** y1-axis, default value
- 2** y2 axis

EXAMPLE

The following examples looks for a plot and adds a variable to the second y axis

```
object graphBoard;
object page;
object plot;
graphBoard=GetGraphBoard(); ! Look for open desktop.
if (graphBoard) {
    page=graphBoard.GetPage('Voltage',1); ! Get panel named 'voltage'
    if (page) {
        plot=page.GetVI('RST','VisPlot2',1); ! get plot
        if (plot) {
            plot.AddVars(line,'c:loading',2);
        }
    }
}
```

Clear

Removes variables from plot

```
void VisPlot2.Clear([int y2])
```

ARGUMENTS

y2 (optional)

Possible values:

- 1** y1-axis, default value
- 2** y2 axis

EXAMPLE

The following example looks for a plot and removes all variables of y2 from plot

```

object page;
object graphBoard;
object plot;
graphBoard=GetGraphBoard(); ! Look for open graphics board.
if (graphBoard) {
    page=graphBoard.GetPage('Voltage',1); ! get panel named 'Voltage'
    if (page) {
        plot=page.GetVI('RST','VisPlot',1); ! get plot
        if (plot) {
            plot.Clear(2);
        }
    }
}

```

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

EXAMPLE

The following example gets the plot and scales the x-axis automatically

```

object page;
object graphBoard;
object plot;
graphBoard=GetGraphBoard(); ! Look for open graphics board.
if (graphBoard) {
    page=graphBoard.GetPage('Voltage',1); ! get panel named 'Voltage'
    if (page) {
        plot=page.GetVI('U','VisPlot2',1); ! get plot
        if (plot) {
            plot.DoAutoScaleX();
        }
    }
}

```

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

EXAMPLE

The following example gets the plot and scales the y1-axis automatically

```
object page;
object graphBoard;
object plot;
graphBoard=GetGraphBoard(); ! Look for open graphics board.
if (graphBoard) {
    page=graphBoard.GetPage('Voltage',1); ! get panel named 'Voltage'
    if (page) {
        plot=page.GetVI('RST','VisPlot2',1); ! get plot
        if (plot) {
            plot.DoAutoScaleY();
        }
    }
}
```

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

EXAMPLE

The following example gets the plot and scales the y1-axis automatically

```
object page;
object graphBoard;
object plot;
graphBoard=GetGraphBoard(); ! Look for open graphics board.
if (graphBoard) {
    page=graphBoard.GetPage('Voltage',1); ! get panel named 'Voltage'
    if (page) {
        plot=page.GetVI('RST','VisPlot2',1); ! get plot
        if (plot) {
            plot.DoAutoScaleY2();
        }
    }
}
```

GetDataSource

Get data source for curve with index

```
object VisPlot.GetDataSource(int curveIndex, int yaxis)
```

ARGUMENTS

int curveIndex

Possible values:

< 0 invalid
≥ 0 curve index in table

int yaxis Possible values:

1 y1 axis
2 y2 axis

RETURNS

object Data source

NULL No data source found

EXAMPLE

The following example looks for a plot and gets the data source for the first curve of y2 axis.

```
object dataSource;
object page;
object graphBoard;
object plot;
graphBoard=GetGraphBoard(); ! Look for open graphics board.
if (graphBoard) {
    page=graphBoard.GetPage('Voltage',1); ! get panel named Voltage
    if (page) {
        plot=page.GetVI('RST','VisPlot2',1); ! get plot
        if (plot) {
            dataSource = plot.GetDataSource(0,2);
        }
    }
}
```

GetScaleObjX

Gets the object used for scaling the x-axis.

```
object 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'.

EXAMPLE

The following example gets the object specifying the x-scale of plot.

```
object page;
object graphBoard;
object plot;
object scaleObjX;
graphBoard=GetGraphBoard(); ! Look for open graphics board.
if (graphBoard) {
    page=graphBoard.GetPage('Voltage',1); ! get panel named Voltage
    if (page) {
        plot=page.GetVI('RST','VisPlot2',1); ! get plot
        if (plot) {
            scaleObjX = plot.GetScaleObjX();
```

```

        scaleObjX.ShowFullName();
    }
}
}
```

GetScaleObjY

Returns used object defining y-scale. The returned object is either the plot itself or the plot type (IntPlot).

```
object 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.

EXAMPLE

The following examples look for a Subplot named 'RST' and get the used y2-scale object.

```

object page;
object graphBoard;
object plot;
object scale;
graphBoard=GetGraphBoard(); ! Look for open graphics board.
if (graphBoard) {
    page=graphBoard.GetPage('Voltage',1); ! get panel named 'Voltage'
    if (page) {
        plot=page.GetVI('RST','VisPlot2',1); ! get plot
        if (plot) {
            scale=plot.GetScaleObjY(2);
            scale.ShowFullName();
        }
    }
}
```

SetAdaptX

Sets the Adapt Scale option of the local x-scale.

```
void VisPlot2.SetAdaptX(int mode,
                        [double trigger]
                        )
```

ARGUMENTS

mode Possible values:

0	off
1	on

trigger (optional)

Trigger value, unused if mode is off or empty

EXAMPLE

The following example looks for a plot and sets its adapt scale option.

```
object page;
object graphBoard;
object plot;
graphBoard=GetGraphBoard(); ! Look for open graphics board.
if (graphBoard) {
    page=graphBoard.GetPage('Voltage',1); ! get panel named Voltage
    if (page) {
        plot=page.GetVI('RST','VisPlot2',1); ! get plot
        if (plot) {
            plot.SetAdaptX(1,3); ! Turn on adapt scale, use a trigger value of 3
            plot.SetAdaptX(0,3); ! Turn off adapt scale
            plot.SetAdaptX(1); ! Turn on adapt scale again, do not change the trigger value
        }
    }
}
```

SetAdaptY

Sets the Adapt Scale option of the local y-scale.

```
void VisPlot2.SetAdaptY(int mode,
                        [double 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

EXAMPLE

The following example looks for a plot and sets the adapt scale option of y2 to 1, trigger value is 4

```
object page;
object graphBoard;
object plot;
graphBoard=GetGraphBoard(); ! Look for open graphics board.
if (graphBoard) {
    page=graphBoard.GetPage('Voltage',1); ! get panel named Voltage
    if (page) {
        plot=page.GetVI('RST','VisPlot2',1); ! get plot
        if (plot) {
            plot.SetAdaptY(1,4,2);
```

```

    }
}
}
```

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.

```
void VisPlot2.SetAutoScaleX(int mode)
```

ARGUMENTS

mode Possible values:

- 0** never
- 1** after simulation
- 2** during simulation

EXAMPLE

The following example gets the plot and sets the Auto Scale option of the x-axis to On.

```
object page;
object graphBoard;
object plot;
graphBoard=GetGraphBoard(); ! Look for open graphics board.
if (graphBoard) {
    page=graphBoard.GetPage('Voltage',1); ! get panel named Voltage
    if (page) {
        plot=page.GetVI('RST','VisPlot',1); ! get plot
        if (plot) {
            plot.SetAutoScaleX(1);
        }
    }
}
```

SetAutoScaleY

Sets automatic scaling mode of the y-scale. The axis given in the second argument is automatically set to local.

```
void 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

EXAMPLE

The following example looks for a and change its auto scale mode to Online

```
object graphBoard;
object page;
object plot;
! Look for opened graphics board.
graphBoard=GetGraphBoard();
if (graphBoard) {
    page=graphBoard.GetPage('Voltage',1); ! get panel named 'Voltage'
    if (page) {
        plot=page.GetVI('RST','VisPlot2',1); ! get plot
        if (plot) {
            plot.SetAutoScaleY(2); ! Turn off automatic scaling of y-scale
        }
    }
}
```

SetCrvDesc

Sets the user defined description of a curve.

```
void VisPlot2.SetCrvDesc(int curveIndex, string curveDescription)
```

ARGUMENTS

curveIndex

Curve index; first curve in table is index 1.

curveDescription

Description to set

EXAMPLE

The following example gets a plot and sets the description of the first variable to example.

```
object page;
object graphBoard;
object plot;
graphBoard=GetGraphBoard(); ! Look for open graphics board.
if (graphBoard) {
    page=graphBoard.GetPage('Voltage',1); ! get panel named Voltage
    if (page) {
        plot=page.GetVI('Plot','VisPlot2',1); ! get plot
        if (plot) {
            plot.SetCrvDesc(1,'example');
        }
    }
}
```

SetDefScaleX

Sets the x-scale to be used to the graphics board.

```
void VisPlot2.SetDefScaleX()
```

EXAMPLE

The following example gets the plot and sets the x-scale to the graphics board.

```
object graphBoard;
object page;
object plot;
graphBoard=GetGraphBoard(); ! Look for open graphics board.
if (graphBoard) {
    page=graphBoard.GetPage('Voltage',1); ! Get panel named 'Voltage'
    if (page) {
        plot=page.GetVI('RST','VisPlot2',1); ! get plot
        if (plot) {
            plot.SetDefScaleX();
        }
    }
}
```

SetDefScaleY

Sets the y-scale to be used to the plot type.

```
void VisPlot2.SetDefScaleY([int y2])
```

ARGUMENTS

y2 (optional)

Possible values:

- 1 y1-axis, default value
- 2 y2 axis

EXAMPLE

The following example gets the plot and sets the y-scale to the plot type.

```
object graphBoard;
object page;
object plot;
graphBoard=GetGraphBoard(); ! Look for open desktop.
if (graphBoard) {
    page=graphBoard.GetPage('Voltage',1); ! Get panel named 'Voltage'
    if (page) {
        plot=page.GetVI('RST','VisPlot2',1); ! get plot
        if (plot) {
            plot.SetDefScaleY(1);
        }
    }
}
```

SetScaleX

Sets the local x-axis scale. A function call without arguments sets the Auto Scale setting to On without changing the scale itself.

```
void VisPlot.SetScaleX()
void VisPlot.SetScaleX(double min,
                      double max,
                      [int log]
)
```

ARGUMENTS

min (optional)

Minimum of x-scale.

max (optional)

Maximum of x-scale.

log (optional)

Possible values:

0 linear

1 logarithmic

EXAMPLE

The following examples look for a and sets its x-scale. There are three different examples.

1. Example: Set x-axis to local and change 'Auto Scale' of x axis to 'On'.
2. Example: Set minimum to 0 and maximum to 20.
3. Example: Set minimum to 1 and maximum to 1000. Changes to a logarithmic scale.

```
! Set Auto Scale to On
object graphBoard;
object page;
object plot;
graphBoard=GetGraphBoard(); ! Look for open graphics board.
if (graphBoard) {
    page=graphBoard.GetPage('Voltage',1); ! get panel named 'Voltage'
    if (page) {
        plot=page.GetVI('RST','VisPlot2',1); ! get plot named RST
        if (plot) {
            plot.setScaleX(); ! set auto scale mode to On
        }
    }
}

! Set minimum and maximum without changing linear/log
object graphBoard;
object page;
object plot;
graphBoard=GetGraphBoard(); ! Look for open graphics board.
if (graphBoard) {
    page=graphBoard.GetPage('Voltage',1); ! get panel named 'Voltage'
    if (page) {
        plot=page.GetVI('RST','VisPlot2',1); ! get plot named RST
        if (plot) {
            plot.setScaleX(0,20); ! Set minimum and maximum
        }
    }
}
```

```

! Set minimum and maximum, change to logarithmic scale
object graphBoard;
object page;
object plot;
graphBoard=GetGraphBoard(); ! Look for open graphics board.
if (graphBoard) {
    page=graphBoard.GetPage('Voltage',1); ! get panel named 'Voltage'
    if (page) {
        plot=page.GetVI('RST','VisPlot2',1); ! get plot named RST
        if (plot) {

            plot.SetScaleX(1,1000,1); ! Set minimum and maximum, change to log scale
        }
    }
}

```

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.

```

void VisPlot2.setScaleY()
void VisPlot2.setScaleY(double min,
                      double 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

EXAMPLE

The following examples looks for a plot and sets its y1 scale to [1,1000] and log.

```

object graphBoard;
object page;
object plot;
graphBoard=GetGraphBoard(); ! Look for open graphics board.
if (graphBoard) {
    page=graphBoard.GetPage('Voltage',1); ! Get panel named 'Voltage'
    if (page) {

```

```

    plot=page.GetVI('RST','VisPlot',1); ! get plot
    if (plot) {
        plot.SetScaleY(1,1000,1,1);
    }
}
}

```

SetXVar

Sets the local x-axis variable. If The default x-axis variable (time) is set if no argument is passed.

```

void VisPlot.SetXVar()
void VisPlot.SetXVar(object obj,]
                     string varname
                     )

```

ARGUMENTS

obj (*optional*)
x-axis object

varname (*optional*)
variable of *obj*

EXAMPLE

The following examples look for a plot and set its x-axis variable. The first example sets an user defined x-axis variable of the plot. The second one sets the default x-axis variable (time).

```

! set x-axis variable 'm:U1:bus1' of object line
object graphBoard;
object page;
object plot;
graphBoard=GetGraphBoard(); ! Look for open desktop.
if (graphBoard) {
    page=graphBoard.GetPage('Voltage',1); ! Get panel named 'Voltage'
    if (page) {
        plot=page.GetVI('RST','VisPlot2',1); ! get plot
        if (plot) {
            plot.SetXVar(line,'m:U1:bus1'); ! Set x-scale variable
        }
    }
    ! set default x-axis variable
    object graphBoard;
    object page;
    object plot;
    graphBoard=GetGraphBoard(); ! Look for open desktop.
    if (graphBoard) {
        page=graphBoard.GetPage('Voltage',1); ! Get panel named 'Voltage'
        if (page) {
            plot=page.GetVI('RST','VisPlot2',1); ! get plot
            if (plot) {
                plot.SetXVar(); ! Set default x-scale variable (time)
            }
        }
    }
}

```

ShowY2

Enables or disables the y2 axis.

```
void VisPlot2.ShowY2([int show])
```

ARGUMENTS

show (*optional*)

Possible values:

- 0** hide y2 axis
- 1** show y2 axis (default)

EXAMPLE

The following example looks for a plot and hides the y2 axis.

```
object graphBoard;
object page;
object plot;
graphBoard=GetGraphBoard(); ! Look for open desktop.
if (graphBoard) {
    page=graphBoard.GetPage('Voltage',1); ! Get panel named 'Voltage'
    if (page) {
        plot=page.GetVI('RST','VisPlot2',1); ! get plot
        if (plot) {
            plot.ShowY2(0);
        }
    }
}
```

3.6.73 VisPlottz

Overview

[AddRelay](#)
[AddRelays](#)
[Clear](#)
[DoAutoScaleX](#)
[DoAutoScaleY](#)

AddRelay

Adds a relay to the plot and optionally sets the drawing style.

```
void VisPlottz.AddRelay(object relay,
                        [double colour,]
                        [double style,]
                        [double width])
```

ARGUMENTS

relay The protection device 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.

```
void VisPlottz.AddRelays(set relays)
```

ARGUMENTS

relays The protection devices (ElmRelay or RelFuse) to be added.

Clear

Removes all protection devices from the plot.

```
void 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.

3.6.74 VisVec

Overview

[CentreOrigin](#)

CentreOrigin

Centre the origin of the plot

```
void VisVec.CentreOrigin()
```

3.6.75 VisVecres

Overview

[CentreOrigin](#)

CentreOrigin

Centre the origin of the plot

```
void VisVecres.CentreOrigin()
```

3.6.76 VisXyplot

Overview

[Clear](#)
[DoAutoScaleX](#)
[DoAutoScaleY](#)
[GetDataSource](#)
[SetCrvDescX](#)
[SetCrvDescY](#)

Clear

Removes all curves from plot.

```
void VisXyplot.Clear()
```

EXAMPLE

The following example gets the plot and removes all curves.

```
object page;
object graphBoard;
object plot;
graphBoard=GetGraphBoard(); ! Look for open graphics board.
if (graphBoard) {
    page=graphBoard.GetPage('Current',1); ! get panel
    if (page) {
        plot=page.GetVI('RST','VisXyplot',1); ! get plot
        if (plot) {
            plot.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

EXAMPLE

The following example gets the plot and scales the x-axes automatically

```
object page;
object graphBoard;
object plot;
graphBoard=GetGraphBoard(); ! Look for open graphics board.
if (graphBoard) {
    page=graphBoard.GetPage('Voltage',1); ! get panel named 'Voltage'
    if (page) {
        plot=page.GetVI('U','VisXyplot',1); ! get plot
        if (plot) {
            plot.DoAutoScaleX();
        }
    }
}
```

DoAutoScaleY

Scales all used y-axes automatically.

```
int VisXyplot.DoAutoScaleY()
```

RETURNS

- 0** Ok, call to DoAutoScaleY() was successfull
- 1** Failed, because the y-scales are not local

EXAMPLE

The following example gets the plot and scales the y-axes automatically

```
object page;
object graphBoard;
object plot;
graphBoard=GetGraphBoard(); ! Look for open graphics board.
if (graphBoard) {
    page=graphBoard.GetPage('Voltage',1); ! get panel named 'Voltage'
    if (page) {
        plot=page.GetVI('U','VisXyplot',1); ! get plot
        if (plot) {
            plot.DoAutoScaleY();
```

```

        }
    }
}
```

GetDataSource

Get data source for curve with index

```
object VisXyplot.GetDataSource(int curveIndex)
```

ARGUMENTS

int curveIndex

Possible values:

- < **0** invalid
- ≥ **0** curve index in table

RETURNS

object Data source

NULL No data source found

EXAMPLE

The following example looks for a plot and gets the data source for the first curve.

```

object dataSource;
object page;
object graphBoard;
object plot;
graphBoard=GetGraphBoard(); ! Look for open graphics board.
if (graphBoard) {
    page=graphBoard.GetPage('Voltage',1); ! get panel named Voltage
    if (page) {
        plot=page.GetVI('RST','VisXyplot',1); ! get plot
        if (plot) {
            dataSource = plot.GetDataSource(0);
        }
    }
}
```

SetCrvDescX

Sets the user defined description of a curve for the x-variable.

```
void VisXyplot.SetCrvDescX(int curveIndex, string curveDescription)
```

ARGUMENTS

curveIndex

Curve index; first curve in table is index 1.

curveDescription

Description to set

EXAMPLE

The following example gets a plot and sets the description of the first variable to example.

```
object page;
object graphBoard;
object plot;
graphBoard=GetGraphBoard(); ! Look for open graphics board.
if (graphBoard) {
    page=graphBoard.GetPage('Current',1); ! get panel
    if (page) {
        plot=page.GetVI('Plot','VisXyplot',1); ! get plot
        if (plot) {
            plot.SetCrvDescX(1,'xvalue');
        }
    }
}
```

SetCrvDescY

Sets the user defined description of a curve for the y-variable.

```
void VisXyplot.SetCrvDescY(int curveIndex, string curveDescription)
```

ARGUMENTS

curveIndex

Curve index; first curve in table is index 1.

curveDescription

Description to set

EXAMPLE

The following example gets a plot and sets the description of the first variable to example.

```
object page;
object graphBoard;
object plot;
graphBoard=GetGraphBoard(); ! Look for open graphics board.
if (graphBoard) {
    page=graphBoard.GetPage('Current',1); ! get panel
    if (page) {
        plot=page.GetVI('Plot','VisXyplot',1); ! get plot
        if (plot) {
            plot.SetCrvDescY(1,'yvalue');
        }
    }
}
```

4 Set Routines

Overview

Add*
Clear*
Count*
First*
FirstFilt*
IsIn*
MarkInGraphics
Next*
NextFilt*
Obj*
OutputFlexibleData*
Remove*
ShowModalBrowser
ShowModalSelectBrowser
ShowModelessBrowser
SortToClass*
SortToName*
SortToVar*

Add*

Adds an object or all objects from a set to the set.
An object is only added if not already contained.

```
int set.Add(object obj)
int set.Add(set objects)
```

ARGUMENTS

obj Object to add.
objects Set of objects to add.

RETURNS

0 On success.
≠ 0 On error, e.g. object is already contained in the set.

SEE ALSO

[set.IsIn\(\)](#), [set.Remove\(\)](#)

EXAMPLE

The following example collects all not selected lines.

```
set selected, relevant, notselected;
object line;
int isIn;
selected = SEL.AllLines();
relevant = GetCalcRelevantObjects();
line = relevant.FirstFilt('* ElmLne');
while (line) {
    isIn = selected.IsIn(line);
    if (isIn = 0) {
        notselected.Add(line);
    }
    line = relevant.NextFilt();
```

Clear*

Removes all objects from the set.

```
void set.Clear()
```

SEE ALSO

[set.Remove\(\)](#)

Count*

Returns the number of objects in the set.

```
int set.Count()
```

RETURNS

The number (≥ 0) of objects in the set.

EXAMPLE

The following example outputs the number of calculation relevant lines (in an active project).

```
int n;
set lines;
lines = GetCalcRelevantObjects('ElmLne');
n = lines.Count();
printf('There are %d calculation relevant line elements inside active project', n);
```

First*

Returns the first object in the set.

```
object set.First()
```

RETURNS

≠ NULL First object in set.

NULL Set is empty.

SEE ALSO

[set.Next\(\)](#), [set.FirstFilt\(\)](#), [set.NextFilt\(\)](#)

EXAMPLE

The following example writes the full names of all lines in the general selection to the output window.

```
set lines;
object line;
lines = SEL.AllLines();
line = lines.First();
while (line) {
    line.ShowFullName();
    line = lines.Next();
}
```

FirstFilt*

Returns the first object from the set which name matches the filter.

The filter describes an object name (case-sensitive, optionally including a class name) and can contain the following wildcards:

- ? Represents exactly one arbitrary character.
- * Represents zero or finitely many arbitrary characters.

Using ',' as separator combines multiple filters as logical disjunction.

Examples:

- 'S*' matches e.g. 'Switch.ElmCoup' or 'Shunt.ElmShnt'.
- 'S*.ElmCoup' restricts the previous matches to switches-breakers.
- '*.ElmTr2, *.ElmTr3' equals '*.ElmTr?' and matches all two and three winding transformers.

```
object set.FirstFilt(string filter)
```

ARGUMENTS

filter Object name filter, possibly containing wildcards.

RETURNS

≠ **NULL** The first object matching the filter.

NULL No object matches the filter.

SEE ALSO

[set.NextFilt\(\)](#), [set.First\(\)](#), [set.Next\(\)](#)

EXAMPLE

The following example writes all two and three winding transformers (class name *ElmTr2* and *ElmTr3*) whose names start with a 'T' to the output window.

```
set transformers;
object transformer;
transformers = GetCalcRelevantObjects();
transformer = transformers.FirstFilt('T*.ElmTr?');
while (transformer) {
    transformer.ShowFullName();
    transformer = transformers.NextFilt();
```

IsIn*

Checks if the set contains given object.

```
int set.IsIn(object obj)
```

ARGUMENTS

obj Object to check.

RETURNS

0 Object is not contained.
1 Object is contained.

For NULL, the result is always 0.

SEE ALSO

[set.Add\(\)](#), [set.Remove\(\)](#)

EXAMPLE

See [set.Add\(\)](#).

MarkInGraphics

This function is deprecated. Please use [MarkInGraphics\(\)](#) instead.

```
void set.MarkInGraphics([int searchAllDiagramsAndSelect])
```

Next*

Returns the next object in the set.

```
object set.Next()
```

RETURNS

≠ NULL Next object in set.
NULL Last object has been reached.

SEE ALSO

[set.First\(\)](#)

EXAMPLE

See [set.First\(\)](#).

NextFilt*

Returns the next object from the set which name matches the filter given in [set.FirstFilt\(\)](#).

```
int set.NextFilt()
```

RETURNS

≠ NULL Next object matching the filter.

NULL No next object matches the filter.

SEE ALSO

[set.FirstFilt\(\)](#), [set.First\(\)](#), [set.Next\(\)](#)

EXAMPLE

See [set.FirstFilt\(\)](#).

Obj*

Returns the object at the given index in the set.

```
int set.Obj(int index)
```

ARGUMENTS

index the index of the object.

RETURNS

The object at the given index in the set, when 'index' is in range, NULL otherwise.

OutputFlexibleData*

This function is deprecated. Please use [OutputFlexibleData\(\)](#) instead.

```
void set.OutputFlexibleData([string flexibleDataPage = ''])
```

Remove*

Removes an object from the set.

```
int set.Remove(object obj)
```

ARGUMENTS

obj Object to remove.

RETURNS

0 On success.

1 On error.

SEE ALSO

[set.Add\(\)](#), [set.IsIn\(\)](#)

EXAMPLE

The following example removes all lines shorter than 1km from a set.

```
set lines;
object line;
double length;
lines = SEL.AllLines();
line = lines.First();
while (line) {
    length = line:dline;
    if (length < 1) {
        lines.Remove(line);
    }
    line = lines.Next();
}
```

ShowModalBrowser

This function is deprecated. Please use [ShowModalBrowser\(\)](#) instead.

```
void set.ShowModalBrowser([int detailMode = 0,
                           [string title = '',
                            [string page = ''])
```

ShowModalSelectBrowser

This function is deprecated. Please use [ShowModalSelectBrowser\(\)](#) instead.

```
set.set.ShowModalSelectBrowser([string title,
                                [string classFilter,])
```

ShowModelessBrowser

This function is deprecated. Please use [ShowModelessBrowser\(\)](#) instead.

```
void set.ShowModelessBrowser([int detailMode = 0,
                               [string title = '',
                                [string page = ''])
```

SortToClass*

Sorts the objects in the set according their class names.

```
void set.SortToClass(int descending)
```

ARGUMENTS

descending

- | | |
|----------|-------------------------|
| 0 | Sorts ascending A...Z. |
| 1 | Sorts descending Z...A. |

EXAMPLE

The following example writes all lines and terminals to the output window, ascending sorted according their class names.

```
set objects;
object obj;
objects = GetCalcRelevantObjects('*.ElmLne, *.ElmTerm');
objects.SortToClass(0);
obj = objects.First();
while (obj) {
    obj.ShowFullName();
    obj = objects.Next();
}
```

SortToName*

Sorts the objects in the set according their names ('loc.name').

```
int set.SortToName(int descending)
```

ARGUMENTS

descending

- | | |
|----------|-------------------------|
| 0 | Sorts ascending A...Z. |
| 1 | Sorts descending Z...A. |

EXAMPLE

The following example writes all lines and terminals to the output window, ascending sorted according their names.

```
set objects;
object obj;
objects = GetCalcRelevantObjects('*.ElmLne, *.ElmTerm');
objects.SortToName(0);
obj = objects.First();
while (obj) {
    obj.ShowFullName();
    obj = objects.Next();
}
```

SortToVar*

Sorts the objects in the set according the values of the given variable names.

The first variable name defines the principle sorting. Each further variable name defines a sub-sorting.

```
void set.SortToVar(int descending,
                   string variable1,
                   [...,]
                   [string variable5]
)
```

ARGUMENTS

descending

- 0** Sorts ascending.
- 1** Sorts descending.

variable1 Variable name of principle sorting.

variable2, ..., variable5 (optional)

2nd..5th variable names for sub-sortings.

EXAMPLE

The following example writes all lines to the output window, sorted by derating factor and length.

```
set lines;
object line;
lines = GetCalcRelevantObjects('*ElmLne');
lines.SortToVar(0, 'fline', 'dline');
line = lines.First();
while (line) {
    printf('%10s %f %f', line:loc_name, line:fline, line:dline);
    line = lines.Next();
}
```

Index

abs
 Mathematical Functions, [65](#)
acos
 Mathematical Functions, [65](#)
Activate
 ElmNet, [213](#)
 IntCase, [594](#)
 IntLibrary, [634](#)
 IntPrj, [652](#)
 IntScenario, [668](#)
 IntScensched, [671](#)
 IntScheme, [673](#)
 IntScheduler, [674](#)
 IntStage, [675](#)
ActivateProject
 Global Functions, [3](#)
ActiveCase
 Global Functions, [10](#)
ActiveProject
 Global Functions, [8](#)
AdaptWidth
 SetLevelvis, [548](#)
Add
 Set Routines, [768](#)
AddBreaker
 StaCubic, [294](#)
AddButton
 ComTablereport, [466](#)
AddCellAction
 ComTablereport, [467](#)
AddCntcy
 ComSimoutage, [458](#)
AddColumn
 ComTablereport, [468](#)
AddConfiguration
 SetDataext, [536](#)
AddContingencies
 ComSimoutage, [459](#)
AddCopy
 General Object Methods, [133](#)
AddCubicle
 ElmBoundary, [183](#)
AddCurve
 ComTablereport, [468](#)
 PltDataseries, [691](#)
AddDouble
 IntAddonvars, [590](#)
AddDoubleMatrix
 IntAddonvars, [590](#)
AddDoubleVector
 IntAddonvars, [591](#)
AddEvent
 IntRas, [666](#)
AddHeader
 ComTablereport, [469](#)
AddInteger
 IntAddonvars, [591](#)
AddIntegerVector
 IntAddonvars, [592](#)
AddInvisibleFilter
 ComTablereport, [469](#)
AddListFilter
 ComTablereport, [470](#)
AddListFilterEntries
 ComTablereport, [470](#)
AddMultiListFilter
 ComTablereport, [471](#)
AddObject
 IntAddonvars, [592](#)
AddObjectVector
 IntAddonvars, [593](#)
AddObjs
 VisBdia, [716](#)
AddPage
 SetDesktop, [539](#)
AddPlot
 ComTablereport, [471](#)
AddProjectToCombined
 IntPrj, [652](#)
AddProjectToRemoteDatabase
 IntPrj, [653](#)
AddRas
 ComSimoutage, [459](#)
AddRatio
 TypCtcore, [712](#)
AddRef
 ComNmink, [431](#)
 IntDataset, [614](#)
 SetSelect, [554](#)
AddRelay

VisDraw, 720
 VisMagndiffplt, 728
 VisOcplot, 729
 VisPcompdiffplt, 735
 VisPlottz, 762
AddRelays
 VisDraw, 720
 VisMagndiffplt, 728
 VisOcplot, 730
 VisPcompdiffplt, 735
 VisPlottz, 763
AddResObjs
 VisBdia, 716
AddResVars
 VisPlot, 736
 VisPlot2, 749
AddRow
 ComTablereport, 472
AddString
 IntAddonvars, 593
AddTable
 ComTablereport, 473
AddTabularFilter
 ComTablereport, 473
AddTextFilter
 ComTablereport, 474
AddToUpdatePages
 ComProtgraphic, 440
AddTrigger
 IntGate, 628
 IntRas, 666
AddVar
 IntMon, 643
AddVariable
 ElmRes, 225
AddVars
 ElmRes, 225
 IntMon, 643
 VisPlot, 737
 VisPlot2, 750
AddXLabel
 ComTablereport, 474
AddXYCurve
 PltDataseries, 692
Align
 SetLevelvis, 548
All
 IntDataset, 614
 SetSelect, 555
AllAsm
 SetSelect, 555
AllBars
 SetSelect, 556
AllBreakers
 SetPath, 551
 SetSelect, 556
AllClosedBreakers
 SetPath, 552
 SetSelect, 556
AllElm
 SetSelect, 557
AllLines
 SetSelect, 557
AllLoads
 SetSelect, 558
AllOpenBreakers
 SetPath, 552
 SetSelect, 558
AllProtectionDevices
 SetPath, 552
AllRelevant
 Global Functions, 11
AllSym
 SetSelect, 558
AllTypLne
 SetSelect, 559
AnalyseElmRes
 ComRel3, 445
AppendCommand
 ComTasks, 506
AppendStudyCase
 ComTasks, 507
Apply
 ElmBmu, 182
 IntOutage, 647
 IntScenario, 668
 IntSubset, 679
ApplyAll
 IntOutage, 647
ApplyAndResetRA
 ElmSubstat, 249
ApplyNetworkState
 IntCase, 594
ApplySelective
 IntScenario, 669
 IntSubset, 679
ApplyStudyTime
 IntCase, 595
Archive
 IntPrj, 653
AreDistParamsPossible
 ElmLne, 203
asin
 Mathematical Functions, 65
AssumeCompensationFactor
 RelZpol, 698
AssumeReRI
 RelZpol, 698
AssumeXeXI
 RelZpol, 698
atan2
 Mathematical Functions, 65

atan
 Mathematical Functions, 65

BeginDataExtensionModification
 IntPrj, 653

BinaryAnd
 Mathematical Functions, 66

BinaryOr
 Mathematical Functions, 66

BlkDef, 572
 CalculateCheckSum, 573
 Compile, 572
 Encrypt, 572
 GetCheckSum, 573
 Pack, 573
 PackAsMacro, 573
 ResetThirdPartyModule, 573
 SetThirdPartyModule, 574

BlkSig, 574
 GetFromSigName, 574
 GetToSigName, 574

BlockSwitch
 ComShctrace, 453

BuildExportStructure
 ComUcteexp, 520

BuildNodeNames
 ComUcteexp, 520

CalcAggrVarsInRadFeed
 ElmFeeder, 191

CalcCluster
 SetCluster, 528
 SetDistrstate, 547

CalcContributions
 ComRelpost, 447

CalcEfficiency
 ElmAsm, 172
 ElmGenstat, 197
 ElmPvsys, 215
 ElmSym, 256
 ElmXnet, 288

CalcEIPParams
 TypAsmo, 711

CalcMaxHostedPower
 ComHostcap, 417

CalcShiftedReversedBoundary
 ElmBoundary, 183

CalculateCheckSum
 BlkDef, 573

CalculateInterchangeTo
 ElmArea, 169
 ElmNet, 213
 ElmZone, 290

CalcWeibullPar
 Mathematical Functions, 67

CanAddProjectToRemoteDatabase
 IntPrj, 653
 CanSubscribeProjectReadOnly
 IntPrj, 654
 CanSubscribeProjectReadWrite
 IntPrj, 654

ceil
 Mathematical Functions, 68

CentreOrigin
 VisDraw, 720
 VisPcompdifffplt, 735
 VisVec, 764
 VisVecres, 764

ChangeFont
 SetLevelvis, 548

ChangeFrameAndWidth
 SetLevelvis, 549

ChangeLayer
 SetLevelvis, 549

ChangeRefPoints
 SetLevelvis, 549

ChangeWidthVisibilityAndColour
 SetLevelvis, 549

ChaVecfile, 575
 Update, 575

Check
 ComAuditlog, 381
 IntOutage, 647
 SetTboxconfig, 560

CheckAll
 IntOutage, 647

CheckAssignments
 ComMerge, 425

CheckBbPath
 ElmBbone, 177

CheckControllers
 ComLdf, 420

CheckFileExists
 File System Functions, 35

CheckRanges
 ElmRelay, 219

CheckSyntax
 ComDpl, 407

CheckUrl
 IntExtaccess, 627

CimArchive, 575
 ConvertToBusBranch, 575

CimModel, 575
 DeleteParameterAtIndex, 576
 GetAttributeEnumerationType, 576
 GetModelsReferencingThis, 576
 GetParameterCount, 576
 GetParameterNamespace, 576
 GetParameterValue, 577
 HasParameter, 577
 RemoveParameter, 577
 SetAssociationValue, 578

SetAttributeEnumeration, 578, 579
 SetAttributeValue, 579
CimObject, 580
 DeleteParameterAtIndex, 580
 GetAttributeEnumerationType, 580
 GetObjectsReferencingThis, 580
 GetObjectsWithSameId, 581
 GetParameterCount, 581
 GetParameterNamespace, 581
 GetParameterValue, 581
 GetPfObjects, 582
 HasParameter, 582
 RemoveParameter, 582
 SetAssociationValue, 582, 583
 SetAttributeEnumeration, 583
 SetAttributeValue, 584
Clear
 ComNmink, 431
 ElmBoundary, 184
 ElmRes, 226
 IntDataset, 615
 IntDplmap, 616
 IntDplvec, 623
 IntSubset, 679
 Set Routines, 769
 SetSelect, 559
 VisBdia, 717
 VisDraw, 720
 VisHrm, 722
 VisMagndiffplt, 728
 VisOcplot, 730
 VisPath, 731
 VisPcompdiffplt, 735
 VisPlot, 738
 VisPlot2, 750
 VisPlottz, 763
 VisXyplot, 764
ClearAll
 ComTableReport, 474
ClearCommands
 Global Functions, 4
ClearCont
 ComSimoutage, 459
ClearCurves
 PltDataseries, 693
ClearData
 IntGrfgroup, 628
 IntGrflayer, 630
ClearInvalidReferences
 IntPrj, 654
ClearOutput
 Output Window Functions, 117
ClearOutputWindow
 Output Window Functions, 117
ClearRecycleBin
 Global Functions, 4
 ClearUpdatePages
 ComProtgraphic, 440
ClearVars
 IntMon, 644
Close
 ElmCoup, 188
 ElmGndswt, 201
 ElmRes, 230
 IntGrfnet, 632
 SetDeskpage, 538
 SetDesktop, 540
 SetVipage, 564
 StaSwitch, 366
CloseTableReports
 Dialogue Boxes Functions, 50
ColLbl
 IntMat, 635, 636
ComAddlabel, 368
 Execute, 368
ComAddon, 369
 CreateModule, 369
 DefineDouble, 369
 DefineDoubleMatrix, 370
 DefineDoublePerConnection, 371
 DefineDoubleVector, 371
 DefineDoubleVectorPerConnection, 372
 DefineInteger, 372
 DefineIntegerPerConnection, 373
 DefineIntegerVector, 374
 DefineIntegerVectorPerConnection, 374
 DefineObject, 375
 DefineObjectPerConnection, 376
 DefineObjectVector, 376
 DefineObjectVectorPerConnection, 377
 DefineString, 377
 DefineStringPerConnection, 378
 DeleteModule, 379
 FinaliseModule, 379
 GetActiveModule, 379
 ModuleExists, 379
 SetActiveModule, 380
ComAmpacity, 380
 ExecuteAmpacityCalc, 380
ComAuditlog, 381
 Check, 381
ComBoundary, 381
 GetCreatedBoundaries, 381
ComCapo, 381
 ConnectShuntToBus, 381
 LossCostAtBusTech, 382
 TotalLossCost, 384
ComCheck, 385
 GetNextLoop, 385
ComCimdbexp, 385
 Execute, 385
ComCimdbimp, 385

Execute, 385
 ImportAndConvert, 386
ComCimvalidate, 386
 Execute, 387
 GetClassType, 387
 GetDescriptionText, 387
 GetInputObject, 387
 GetModel, 388
 GetModelId, 388
 GetNumberOfValidationMessages, 388
 GetObject, 388
 GetObjectId, 389
 GetProfile, 389
 GetSeverity, 389
 GetType, 389
ComConreq, 390
 Execute, 390
ComContingency, 390
 ContinueTrace, 391
 CreateRecoveryInformation, 391
 GetGeneratorEvent, 391
 GetInterruptedPowerAndCustomersForStage, 392
 GetInterruptedPowerAndCustomersForTimeStep, 392
 GetLoadEvent, 393
 GetNumberOfGeneratorEventsForTimeStep, 393
 GetNumberOfLoadEventsForTimeStep, 393
 GetNumberOfSwitchEventsForTimeStep, 394
 GetNumberOfTimeSteps, 394
 GetObj, 394
 GetSwitchEvent, 394
 GetTimeOfStepInSeconds, 395
 GetTotalInterruptedPower, 395
 JumpToLastStep, 395
 RemoveEvents, 395
 StartTrace, 396
 StopTrace, 396
ComCoordreport, 396
 DevicesToReport, 397
 HasResultsForDirectionalBackup, 397
 HasResultsForNonDirectionalBackup, 397
 HasResultsForOverreach, 398
 HasResultsForZone, 398
 MaxZoneNumberFor, 398
 ResultForDirectionalBackupVariable, 399
 ResultForNonDirectionalBackupVariable, 399
 ResultForOverreachVariable, 400
 ResultForZoneVariable, 400
 TopologyForDirectionalBackupVariable, 401
 TopologyForNonDirectionalBackupVariable, 402
 TopologyForOverreachVariable, 402
 TopologyForZoneVariable, 402
 TransferDirectionalBackupResultsTo, 403
 TransferNonDirectionalBackupResultsTo, 403
 TransferOverreachResultsTo, 404
 TransferResultsTo, 405
 TransferZoneResultsTo, 405
ComDiff, 406
 Start, 406
 Stop, 406
ComDlmanager, 406
 Report, 406
ComDpl, 407
 CheckSyntax, 407
 Encrypt, 407
 Execute, 408
 GetExternalObject, 409
 GetInputParameterDouble, 409
 GetInputParameterInt, 409
 GetInputParameterString, 410
 IsEncrypted, 410
 ResetThirdPartyModule, 411
 SetExternalObject, 411
 SetInputParameterDouble, 411
 SetInputParameterInt, 412
 SetInputParameterString, 412
 setResultString, 413
 SetThirdPartyModule, 413
 ComFlickermeter, 413
 Execute, 414
ComGenrelin, 414
 GetCurrentIteration, 414
 GetMaxNumIterations, 415
ComGridtocim, 415
 ConvertAndExport, 415
 SetAuthorityUri, 416
 SetBoundaries, 416
 SetGridsToExport, 416
ComHostcap, 416
 CalcMaxHostedPower, 417
ComImport, 418
 GetCreatedObjects, 418
 GetModifiedObjects, 418
ComInc, 419
 ZeroDerivative, 419
ComLdf, 419
 CheckControllers, 420
 DoNotResetCalc, 420
 EstimateOutage, 420
 Execute, 420
 IsAC, 421
 IsBalanced, 421
 IsDC, 421
 PrintCheckResults, 421
 SetOldDistributeLoadMode, 421
ComLink, 422
 GetMicroSCADASatus, 422
 LoadMicroSCADAFile, 422
 ReceiveData, 423

SendData, 423
SentDataStatus, 423
SetMicroSCADAStatus, 424
SetOPCReceiveQuality, 424
SetSwitchShcEventMode, 424
Commands Methods, 368
 Execute, 368
ComMerge, 425
 CheckAssignments, 425
 Compare, 425
 CompareActive, 425
 ExecuteRecording, 426
 ExecuteWithActiveProject, 426
 GetCorrespondingObject, 426
 GetModification, 426
 GetModificationResult, 427
 GetModifiedObjects, 427
 Merge, 428
 PrintComparisonReport, 428
 PrintModifications, 428
 Reset, 428
 SetAutoAssignmentForAll, 428
 SetObjectsToCompare, 429
 ShowBrowser, 429
 WereModificationsFound, 429
CommitTransaction
 Global Functions, 4
CommitTx_
 Global Functions, 4
ComMot, 430
 GetMotorConnections, 430
 GetMotorSwitch, 430
 GetMotorTerminal, 430
ComNmink, 430
 AddRef, 431
 Clear, 431
 GenerateContingenciesForAnalysis, 431
 GetAll, 432
ComOmr, 432
 GetFeeders, 432
 GetOMR, 433
 GetRegionCount, 434
ComOpc, 435
 ReceiveData, 435
 SendData, 435
ComOutage, 436
 ContinueTrace, 436
 ExecuteTime, 436
 GetObject, 436
 RemoveEvents, 437
 SetObjs, 438
 StartTrace, 438
 StopTrace, 438
Compare
 ComMerge, 425
CompareActive
ComMerge, 425
ComPfdimport, 439
 GetImportedObjects, 439
Compile
 BlkDef, 572
ComPrjconnector, 439
 GetSuccessfullyConnectedItems, 439
 GetUnsuccessfullyConnectedItems, 439
ComProtgraphic, 440
 AddToUpdatePages, 440
 ClearUpdatePages, 440
ComPvcurves, 440
 FindCriticalBus, 440
ComPython, 440
 GetExternalObject, 441
 GetInputParameterDouble, 441
 GetInputParameterInt, 441
 GetInputParameterString, 442
 SetExternalObject, 442
 SetInputParameterDouble, 443
 SetInputParameterInt, 443
 SetInputParameterString, 443
ComRed, 444
 ReductionInMemory, 444
 ResetReductionInMemory, 444
ComRel3, 445
 AnalyseElmRes, 445
 ExeEvt, 446
 OvlAlleviate, 446
 RemoveEvents, 446
 RemoveOutages, 446
 ValidateConstraints, 447
ComRelpost, 447
 CalcContributions, 447
 GetContributionOfComponent, 448
ComRelreport, 449
 GetContingencies, 449
 GetContributionOfComponent, 449
ComRes, 449
 ExportFullRange, 449
 FileNmResNm, 450
ComShc, 450
 ExecuteRXSweep, 450
 GetFaultType, 451
 GetOverLoadedBranches, 451
 GetOverLoadedBuses, 452
ComShctrace, 453
 BlockSwitch, 453
 ExecuteAllSteps, 453
 ExecuteInitialStep, 453
 ExecuteNextStep, 454
 GetBlockedSwitches, 454
 GetCurrentTimeStep, 454
 GetDeviceSwitches, 454
 GetDeviceTime, 455
 GetNonStartedDevices, 455

GetStartedDevices, 455
 GetSwitchTime, 455
 GetTrippedDevices, 455
 NextStepAvailable, 456
ComSim, 456
 GetSimulationTime, 456
 GetTotalWarnA, 456
 GetTotalWarnB, 456
 GetTotalWarnC, 457
 GetViolatedScanModules, 457
 LoadSimulationState, 457
 LoadSnapshot, 457
 SaveSimulationState, 457
 SaveSnapshot, 457
ComSimoutage, 458
 AddCntcy, 458
 AddContingencies, 459
 AddRas, 459
 ClearCont, 459
 CreateFaultCase, 460
 Execute, 460
 ExecuteAndCheck, 460
 GetNTopLoadedElms, 462
 MarkRegions, 462
 RemoveAllRas, 462
 RemoveContingencies, 462
 RemoveRas, 462
 Reset, 463
 SetLimits, 463
 Update, 463
ComSvgexport, 463
 SetFileName, 464
 SetObject, 464
 SetObjects, 464
ComSvgimport, 464
 SetFileName, 464
 SetObject, 464
ComTablereport, 465
 AddButton, 466
 AddCellAction, 467
 AddColumn, 468
 AddCurve, 468
 AddHeader, 469
 AddInvisibleFilter, 469
 AddListFilter, 470
 AddListFilterEntries, 470
 AddMultiListFilter, 471
 AddPlot, 471
 AddRow, 472
 AddTable, 473
 AddTabularFilter, 473
 AddTextFilter, 474
 AddXLabel, 474
 ClearAll, 474
 DisableAutomaticRowNumbering, 475
 DisableSingleFilterRefresh, 475
 DisplayButtonsCollapsible, 475
 DisplayFiltersCollapsible, 475
 DisplayFiltersIn2Columns, 476
 DisplayHeadersCollapsible, 476
 EnableAutomaticRowNumbering, 476
 ExportToExcel, 476
 ExportToHTML, 477
 FindNextCell, 477
 FindNextRow, 477
 FindPreviousCell, 478
 FindPreviousRow, 478
 GetCellAccessObject, 479
 GetCellAlignment, 479
 GetCellBackgroundColor, 480
 GetCellColor, 480
 GetCellEditObjects, 481
 GetCellFontStyle, 481
 GetCellFormat, 482
 GetCellHelpText, 482
 GetCellValueType, 483
 GetCheckboxCellValue, 483
 GetColumnFilter, 483
 GetColumnHeader, 484
 GetColumnId, 484
 GetDateCellValue, 485
 GetDoubleCellValue, 485
 GetIntCellValue, 486
 GetNumberOfColumns, 486
 GetNumberOfRows, 486
 GetNumberOfSelectedCells, 487
 GetObjectCellValue, 487
 GetRowCaption, 488
 GetRowId, 488
 GetSelectedCell, 488
 GetSelectedElements, 489
 GetSelection, 489
 GetStringCellValue, 489
 GoToCell, 490
 HasCell, 490
 HasColumnAutoFilter, 490
 IsColumnScrollable, 491
 IsColumnSortable, 491
 IsRowVisible, 491
 NCol, 486
 NRow, 486
 RemoveRow, 492
 SetBarLimits, 492
 SetCellAccess, 493
 SetCellEdit, 493
 SetCellValueToBar, 494
 SetCellValueToCheckbox, 495
 SetCellValueToDate, 496
 SetCellValue.ToDouble, 497
 SetCellValueToInt, 498
 SetCellValueToObject, 499
 SetCellValueToString, 500

SetColumnHeader, 501
 SetCurveValue, 501
 SetDialogSize, 502
 SetExportHtml, 502
 SetExportXls, 502
 SetListFilterSelection, 503
 SetNumberFormatForPlot, 503
 SetRefresh, 504
 SetSorting, 504
 SetStatusText, 504
 SetTextAxisDistForPlot, 504
 SetTicksForPlot, 505
 SetTitle, 505
 SizeX, 486
 SizeY, 486
ComTasks, 506
 AppendCommand, 506
 AppendStudyCase, 507
 GetCommandsForStudyCase, 508
 GetNumberOfCommandsForStudyCase, 509
 GetNumberOfStudyCases, 509
 GetStudyCases, 509
 IsAdditionalResultsFlagSetForCommand, 510
 IsCommandIgnored, 510
 IsStudyCaseIgnored, 511
 RemoveCmdsForStudyCaseRow, 511
 RemoveCommand, 512
 RemoveStudyCase, 512
 RemoveStudyCases, 513
 SetAdditionalResultsFlagForCommand, 513
 SetIgnoreFlagForCommand, 515
 SetIgnoreFlagForStudyCase, 516
 SetResultsFolder, 517
ComTececo, 518
 UpdateTablesByCalcPeriod, 518
ComTransfer, 518
 GetTransferCalcData, 519
 IsLastIterationFeasible, 519
ComUcte, 520
 SetBatchMode, 520
ComUcteexp, 520
 BuildExportStructure, 520
 BuildNodeNames, 520
 DeleteCompleteQuickAccess, 521
 ExportAndInitQuickAccess, 522
 GetConnectedBranches, 522
 GetFromToNodeNames, 523
 GetOrderCode, 524
 GetUcteNodeName, 525
 InitQuickAccess, 526
 QuickAccessAvailable, 526
 ResetQuickAccess, 527
 SetGridSelection, 527
ComWktimp, 527
 GetCreatedObjects, 527
 GetModifiedObjects, 528
 ConnectShuntToBus
 ComCapo, 381
 Consolidate
 IntCase, 595
 IntScheme, 673
 Contains
 IntDplmap, 617
 ContainsNonAsciiCharacters
 General Object Methods, 134
 ContinueTrace
 ComContingency, 391
 ComOutage, 436
 ConvertAndExport
 ComGridtocim, 415
 ConvertToASCIIFormat
 IntComtrade, 596
 ConvertToBinaryFormat
 IntComtrade, 596
 ConvertToBusBranch
 CimArchive, 575
 CopyData
 General Object Methods, 134
CopyDataExtensionFrom
 IntPrj, 654
CopyExtMeaStatusToStatusTmp
 StaExtbrkmea, 301
 StaExtcmdmea, 306
 StaExtdatmea, 311
 StaExtfmea, 316
 StaExtfuelmea, 321
 StaExttimea, 326
 StaExtpfmea, 331
 StaExtpmea, 336
 StaExtqmea, 341
 StaExtsmea, 346
 StaExttapmea, 351
 StaExtv3mea, 356
 StaExtvmea, 361
CopyFile
 File System Functions, 36
cos
 Mathematical Functions, 68
cosh
 Mathematical Functions, 68
Count
 Set Routines, 769
Create
 SetPath, 552
CreateCBEevents
 IntEvt, 626
CreateDerivedProject
 IntVersion, 689
CreateDir
 File System Functions, 37
CreateEvent
 ElmTr2, 269

ElmTr3, 273
 ElmTr4, 277
 ElmVoltreg, 286
 StaExtdatmea, 311
 CreateFaultCase
 ComSimoutage, 460
 Global Functions, 4
 CreateFeederWithRoutes
 ElmLne, 204
 CreateFilter
 SetColscheme, 530
 CreateGroup
 IntUserman, 683
 CreateModule
 ComAddon, 369
 CreateObject
 General Object Methods, 135
 CreateProject
 Global Functions, 5
 CreateRecoveryInformation
 ComContingency, 391
 CreateStageObject
 IntSstage, 676
 CreateUser
 IntUserman, 683
 CreateVersion
 IntPrj, 655
 CreateVI
 SetVipage, 566
 Date/Time Functions, 44
 FormatDate, 49
 FormatDateLT, 44
 FormatDateUTC, 45
 GetStudyTimeObject, 46
 GetSystemTime, 46
 GetSystemTimeUTC, 46
 GetTime, 47
 ParseDateLT, 47
 ParseDateUTC, 48
 strftime, 49
 Date
 SetTime, 562
 Deactivate
 ElmNet, 214
 IntCase, 595
 IntLibrary, 635
 IntPrj, 656
 IntScenario, 669
 IntScensched, 671
 IntScheme, 673
 IntScheduler, 675
 DefineBoundary
 ElmArea, 169
 ElmNet, 215
 ElmZone, 291
 DefineDouble
 ComAddon, 369
 DefineDoubleMatrix
 ComAddon, 370
 DefineDoublePerConnection
 ComAddon, 371
 DefineDoubleVector
 ComAddon, 371
 DefineDoubleVectorPerConnection
 ComAddon, 372
 DefineInteger
 ComAddon, 372
 DefineIntegerPerConnection
 ComAddon, 373
 DefineIntegerVector
 ComAddon, 374
 DefineIntegerVectorPerConnection
 ComAddon, 374
 DefineObject
 ComAddon, 375
 DefineObjectPerConnection
 ComAddon, 376
 DefineObjectVector
 ComAddon, 376
 DefineObjectVectorPerConnection
 ComAddon, 377
 DefineString
 ComAddon, 377
 DefineStringPerConnection
 ComAddon, 378
 Delete
 Global Functions, 5
 DeleteCompleteQuickAccess
 ComUcteexp, 521
 DeleteModule
 ComAddon, 379
 DeleteParameterAtIndex
 CimModel, 576
 CimObject, 580
 DeleteRow
 IntScensched, 672
 DeleteUntouchedObjects
 Global Functions, 6
 Derate
 ElmGenstat, 197
 ElmPvsys, 216
 ElmSym, 257
 DevicesToReport
 ComCoordreport, 397
 Dialogue Boxes Functions, 50
 CloseTableReports, 50
 GetTableReports, 50
 input, 50
 MessageBox, 51
 ShowModalBrowser, 52
 ShowModalOpenFileDialog, 53

ShowModalSaveFileDialog, 54
 ShowModalSelectBrowser, 55
 ShowModalSelectFolderDialog, 56
 ShowModelessBrowser, 57
 UpdateTableReports, 57
DisableAutomaticRowNumbering
 ComTablereport, 475
DisableSingleFilterRefresh
 ComTablereport, 475
DiscardChanges
 IntScenario, 669
Disconnect
 ElmGenstat, 198
 ElmPvsys, 216
 ElmSym, 257
 ElmXnet, 288
DisplayButtonsCollapsible
 ComTablereport, 475
DisplayFiltersCollapsible
 ComTablereport, 475
DisplayFiltersIn2Columns
 ComTablereport, 476
DisplayHeadersCollapsible
 ComTablereport, 476
DoAutoScale
 GrpPage, 585
 PltLinebarplot, 693
DoAutoScaleOnAll
 VisDraw, 720
DoAutoScaleOnCharacteristics
 VisDraw, 721
DoAutoScaleOnImpedances
 VisDraw, 721
DoAutoSizeX
 GrpPage, 585
 SetDesktop, 540
 SetVipage, 564
 VisDraw, 721
 Vishrm, 722
 VisMagndiffplt, 728
 VisOcplot, 730
 VisPath, 731
 VisPcompdifffplt, 736
 VisPlot, 738
 VisPlot2, 751
 VisPlotz, 763
 VisXyplot, 765
DoAutoScaleY2
 VisPlot2, 752
DoAutoSizeY
 GrpPage, 585
 SetVipage, 565
 VisDraw, 721
 Vishrm, 723
 VisMagndiffplt, 729
 VisOcplot, 730
 VisPath, 732
 VisPcompdifffplt, 736
 VisPlot, 739
 VisPlot2, 751
 VisPlotz, 763
 VisXyplot, 765
DoNotResetCalc
 ComLdf, 420
EchoOff
 Environment Functions, 58
EchoOn
 Environment Functions, 58
Edit
 General Object Methods, 164
ElmAra, 169
 CalculateInterchangeTo, 169
 DefineBoundary, 169
 GetAll, 170
 GetBranches, 170
 GetBuses, 171
 GetObjs, 171
ElmAsm, 172
 CalcEfficiency, 172
 GetAvailableGenPower, 172
 GetElecTorque, 173
 GetGroundingImpedance, 174
 GetMechTorque, 174
 GetMotorStartingFlag, 174
 GetStepupTransformer, 175
 IsPQ, 175
ElmAsmsc, 175
 GetAvailableGenPower, 175
 GetGroundingImpedance, 176
 GetStepupTransformer, 177
ElmBbone, 177
 CheckBbPath, 177
 GetBbOrder, 178
 GetCompleteBbPath, 179
 GetFOR, 179
 GetMeanCs, 180
 GetMinCs, 180
 GetTieOpenPoint, 181
 GetTotLength, 181
 HasGnrlMod, 182
ElmBmu, 182
 Apply, 182
 Update, 182
ElmBoundary, 183
 AddCubicle, 183
 CalcShiftedReversedBoundary, 183
 Clear, 184
 GetInterior, 184
 IsSplitting, 184
 Resize, 185
 Update, 185

ElmBranch, 185
 Update, 185
ElmCabsys, 186
 FitParams, 186
 GetLineCable, 186
 Update, 187
ElmComp, 187
 slotupd, 187
 SlotUpdate, 187
ElmCoup, 188
 Close, 188
 GetRemoteBreakers, 188
 IsBreaker, 189
 IsClosed, 189
 IsOpen, 189
 Open, 190
ElmDsl, 190
 ExportToClipboard, 190
 ExportToFile, 191
ElmFeeder, 191
 CalcAggrVarsInRadFeed, 191
 GetAll, 192
 GetBranches, 193
 GetBuses, 193
 GetNodesBranches, 194
 GetObjs, 195
ElmFile, 195
 LoadFile, 195
 SaveFile, 196
ElmFilter, 196
 GetGroundingImpedance, 196
ElmGenstat, 197
 CalcEfficiency, 197
 Derate, 197
 Disconnect, 198
 GetAvailableGenPower, 198
 GetGroundingImpedance, 199
 GetStepupTransformer, 199
 IsConnected, 200
 Reconnect, 200
 ResetDerating, 201
ElmGndswt, 201
 Close, 201
 GetGroundingImpedance, 201
 IsClosed, 202
 IsOpen, 202
 Open, 202
ElmLne, 203
 AreDistParamsPossible, 203
 CreateFeederWithRoutes, 204
 FitParams, 204
 GetIthr, 205
 GetType, 205
 GetY0m, 205
 GetY1m, 206
 GetZ0m, 207
 GetZ1m, 208
 GetZmatDist, 208
 HasRoutes, 209
 HasRoutesOrSec, 209
 IsCable, 210
 IsNetCoupling, 210
 MeasureLength, 211
 SetDetailed, 211
ElmLnsec, 212
 IsCable, 212
ElmNec, 213
 GetGroundingImpedance, 213
ElmNet, 213
 Activate, 213
 CalculateInterchangeTo, 213
 Deactivate, 214
 DefineBoundary, 215
ElmPvsys, 215
 CalcEfficiency, 215
 Derate, 216
 Disconnect, 216
 GetAvailableGenPower, 217
 GetGroundingImpedance, 217
 IsConnected, 218
 Reconnect, 218
 ResetDerating, 219
ElmRelay, 219
 CheckRanges, 219
 GetCalcRX, 220
 GetMaxFdetectCalcl, 220
 GetSlot, 221
 GetUnom, 221
 IsStarted, 221
 SetImpedance, 221
 SetMaxI, 222
 SetMaxIearth, 223
 SetMinI, 223
 SetMinIearth, 223
 SetOutOfService, 223
 SetTime, 224
 slotupd, 224
 SlotUpdate, 224
ElmRes, 224
 AddVariable, 225
 AddVars, 225
 Clear, 226
 Close, 230
 FindColumn, 226
 FindMaxInColumn, 227
 FindMaxOfVariableInRow, 228
 FindMinInColumn, 229
 FindMinOfVariableInRow, 229
 FinishWriting, 230
 Flush, 230
 GetDescription, 231
 GetFirstValidObject, 231

GetFirstValidObjectVariable, 233
 GetFirstValidVariable, 233
 GetNextValidObject, 234
 GetNextValidObjectVariable, 235
 GetNextValidVariable, 236
 GetNumberOfColumns, 236
 GetNumberOfRows, 236
 GetObj, 236
 GetObject, 237
 GetObjectValue, 237
 GetRelCase, 238
 GetSubElmRes, 239
 GetUnit, 239
 GetValue, 240
 GetVariable, 240
 Init, 241
 InitialiseWriting, 241
 Load, 242
 NCol, 236
 NRow, 236
 Release, 242
 SetAsDefault, 243
 SetObj, 243
 SetSubElmResKey, 244
 SizeX, 236
 SizeY, 236
 SortAccordingToColumn, 244
 Write, 245
 WriteDraw, 245
 ElmShnt, 246
 GetGroundingImpedance, 246
 ElmStactrl, 246
 GetControlledHVNode, 247
 GetControlledLVNode, 247
 GetStepupTransformer, 248
 Info, 248
 ElmSubstat, 249
 ApplyAndResetRA, 249
 GetSplit, 249
 GetSplitCal, 251
 GetSplitIndex, 252
 GetSuppliedElements, 253
 OverwriteRA, 254
 ResetRA, 254
 SaveAsRA, 254
 SetRA, 255
 ElmSvs, 255
 GetStepupTransformer, 255
 ElmSym, 256
 CalcEfficiency, 256
 Derate, 257
 Disconnect, 257
 GetAvailableGenPower, 257
 GetGroundingImpedance, 258
 GetMotorStartingFlag, 259
 GetStepupTransformer, 259
 IsConnected, 259
 Reconnect, 260
 ResetDerating, 260
 ElmTerm, 261
 GetBusType, 261
 GetCalcRelevantCubicles, 261
 GetConnectedBrkCubicles, 261
 GetConnectedCubicles, 262
 GetConnectedMainBuses, 262
 GetConnectionInfo, 262
 GetEllaNodeName, 265
 GetEquivalentTerminals, 263
 GetMinDistance, 264
 GetNextHVBus, 264
 GetNodeName, 265
 GetSepStationAreas, 265
 HasCreatedCalBus, 266
 IsElectrEquivalent, 266
 IsEquivalent, 267
 IsInternalNodeInStation, 268
 IsInternalNodeInSubStation, 268
 UpdateSubstationTerminals, 268
 ElmTr2, 268
 CreateEvent, 269
 GetGroundingImpedance, 269
 GetSuppliedElements, 269
 GetTapPhi, 270
 GetTapRatio, 270
 GetZ0pu, 271
 GetZpu, 271
 IsQuadBooster, 271
 NTap, 272
 ElmTr3, 272
 CreateEvent, 273
 GetGroundingImpedance, 273
 GetSuppliedElements, 273
 GetTapPhi, 274
 GetTapRatio, 274
 GetTapZDependentSide, 275
 GetZ0pu, 275
 GetZpu, 275
 IsQuadBooster, 276
 NTap, 276
 ElmTr4, 276
 CreateEvent, 277
 GetGroundingImpedance, 277
 GetSuppliedElements, 278
 GetTapPhi, 278
 GetTapRatio, 279
 GetTapZDependentSide, 279
 GetZ0pu, 279
 GetZpu, 280
 IsQuadBooster, 280
 NTap, 281
 ElmTrfstat, 281
 GetSplit, 281

GetSplitCal, 283
 GetSplitIndex, 284
 GetSuppliedElements, 285
ElmVac, 286
 GetGroundingImpedance, 286
ElmVoltreg, 286
 CreateEvent, 286
 GetGroundingImpedance, 287
 GetZpu, 287
 NTap, 287
ElmXnet, 288
 CalcEfficiency, 288
 Disconnect, 288
 GetGroundingImpedance, 289
 GetStepupTransformer, 289
 Reconnect, 290
ElmZone, 290
 CalculateInterchangeTo, 290
 DefineBoundary, 291
 GetAll, 291
 GetBranches, 292
 GetBuses, 292
 GetObjs, 293
 SetLoadScaleAbsolute, 293
EnableAutomaticRowNumbering
 ComTablereport, 476
EnableDiffMode
 IntStage, 676
EnableUserBreak
 Environment Functions, 63
Encrypt
 BlkDef, 572
 ComDpl, 407
 TypQdsl, 713
EndDataExtensionModification
 IntPrj, 656
Energize
 General Object Methods, 135
Environment Functions, 57
 EchoOff, 58
 EchoOn, 58
 EnableUserBreak, 63
 GetDiffMode, 58
 IsAutomaticCalculationResetEnabled, 58
 IsFinalEchoOnEnabled, 59
 NoFinalUpdate, 59
 SetAutomaticCalculationResetEnabled, 59
 SetConsistencyCheck, 59
 SetDiffMode, 60
 SetEnableUserBreak, 63
 SetFinalEchoOnEnabled, 61
 SetGraphicUpdate, 61
 SetGuiUpdateEnabled, 62
 SetProgressBarUpdatesEnabled, 62
 SetRescheduleFlag, 62
 SetUserBreakEnabled, 63
Error
 Output Window Functions, 117
EstimateOutage
 ComLdf, 420
Exe
 Global Functions, 6
Execute
 ComAddlabel, 368
 ComCimdbexp, 385
 ComCimdbimp, 385
 ComCimvalidate, 387
 ComConreq, 390
 ComDpl, 408
 ComFlickermeter, 414
 ComLdf, 420
 Commands Methods, 368
 ComSimoutage, 460
ExecuteAllSteps
 ComShctrace, 453
ExecuteAmpacityCalc
 ComAmpacity, 380
ExecuteAndCheck
 ComSimoutage, 460
ExecuteCmd
 Global Functions, 6
ExecuteInitialStep
 ComShctrace, 453
ExecuteNextStep
 ComShctrace, 454
ExecuteRecording
 ComMerge, 426
ExecuteRXSweep
 ComShc, 450
ExecuteTime
 ComOutage, 436
ExecuteWithActiveProject
 ComMerge, 426
ExEvt
 ComRel3, 446
exit
 Global Functions, 6
exp
 Mathematical Functions, 69
Export
 IntDocument, 615
 IntGrfgroup, 629
 IntGrflayer, 630
 IntIcon, 634
 ExportAndInitQuickAccess
 ComUcteexp, 522
ExportFullRange
 ComRes, 449
ExportToClipboard
 ElmDsl, 190
ExportToExcel
 ComTablereport, 476

ExportToFile
 ElmDsl, 191
 ExportToHTML
 ComTablereport, 477
 ExportToVec
 IntGrfgroup, 629
 IntGrflayer, 631

 fclose
 File System Functions, 37
 fflush
 File System Functions, 37
 File System Functions, 35
 CheckFileExists, 35
 CopyFile, 36
 CreateDir, 37
 fclose, 37
 fflush, 37
 fopen, 37
 fprintf, 38
 fscanf, 39
 fscanfsep, 40
 GetDirectories, 41
 GetFileDate, 42
 GetFiles, 42
 GetInstallationDirectory, 43
 GetInstallDir, 43
 GetTempDir, 43
 GetTemporaryDirectory, 43
 GetWorkingDir, 43
 GetWorkspaceDirectory, 43

 FileNmResNm
 ComRes, 450
 FinaliseModule
 ComAddon, 379
 FindColumn
 ElmRes, 226
 IntComtrade, 597
 IntComtradeset, 606
 FindCriticalBus
 ComPvcycles, 440
 FindMaxInColumn
 ElmRes, 227
 IntComtrade, 597
 IntComtradeset, 606
 FindMaxOfVariableInRow
 ElmRes, 228
 FindMinInColumn
 ElmRes, 229
 IntComtrade, 598
 IntComtradeset, 607
 FindMinOfVariableInRow
 ElmRes, 229
 FindNextCell
 ComTablereport, 477
 FindNextRow

 ComTablereport, 477
 FindPreviousCell
 ComTablereport, 478
 FindPreviousRow
 ComTablereport, 478
 FinishWriting
 ElmRes, 230
 First
 IntDplmap, 617
 Set Routines, 769
 FirstFilt
 Set Routines, 770
 FitParams
 ElmCabsys, 186
 ElmLne, 204
 floor
 Mathematical Functions, 69
 Flush
 ElmRes, 230
 fopen
 File System Functions, 37
 Format String Syntax, 122
 FormatDate
 Date/Time Functions, 49
 FormatDateLT
 Date/Time Functions, 44
 FormatDateUTC
 Date/Time Functions, 45
 fprintf
 File System Functions, 38
 frac
 Mathematical Functions, 69
 fRand
 Mathematical Functions, 70
 Freeze
 SetDesktop, 540
 fscanf
 File System Functions, 39
 fscanfsep
 File System Functions, 40
 fWrite
 Output Window Functions, 119

 General Object Methods, 132
 AddCopy, 133
 ContainsNonAsciiCharacters, 134
 CopyData, 134
 CreateObject, 135
 Edit, 164
 Energize, 135
 GetChildren, 136
 GetClass, 137
 GetClassName, 137
 GetCombinedProjectSource, 137
 GetConnectedElements, 137
 GetConnectionCount, 138

GetContents, 138
 GetControlledNode, 139
 GetCubicle, 140
 GetFullName, 140
 GetImpedance, 141
 GetInom, 142
 GetNet, 142
 GetNode, 143
 GetOperator, 143
 GetOwner, 143
 GetParent, 144
 GetReferences, 144
 GetRegion, 145
 GetSize, 146
 GetSupplyingSubstations, 147
 GetSupplyingTransformers, 147
 GetSupplyingTrfstations, 147
 GetSystemGround, 147
 GetSystemGrounding, 147
 GetUnom, 148
 GetVal, 149
 GetVarType, 149
 GetZeroImpedance, 150
 HasResults, 150
 Inom, 142
 IsCalcRelevant, 151
 IsClass, 152
 IsDeleted, 152
 IsEarthed, 153
 IsEnergized, 153
 IsHidden, 154
 IsInFeeder, 154
 IsNetworkDataFolder, 155
 IsNode, 156
 IsObjectActive, 156
 IsObjectModifiedByVariation, 156
 Isolate, 157
 IsOutOfService, 157
 IsReducible, 158
 IsRelevant, 151
 IsShortCircuited, 158
 Inm, 158
 MarkInGraphics, 159
 Move, 160
 PasteCopy, 160
 PurgeUnusedObjects, 161
 ReplaceNonAsciiCharacters, 161
 ReportNonAsciiCharacters, 162
 ReportUnusedObjects, 162
 SearchObject, 162
 SetSize, 163
 SetVal, 163
 ShowEditDialog, 164
 ShowFullName, 165
 ShowModalSelectTree, 165
 snm, 166
 SwitchOff, 166
 SwitchOn, 167
 umm, 167
 VarExists, 168
 GenerateContingenciesForAnalysis
 ComNmink, 431
 Get
 IntDplvec, 623
 IntMat, 635
 IntVec, 684
 IntVecobj, 688
 SetFilt, 548
 GetActiveCalculationStr
 Global Functions, 7
 GetActiveModule
 ComAddon, 379
 GetActiveNetworkVariations
 Global Functions, 8
 GetActivePage
 SetDesktop, 540
 GetActiveProject
 Global Functions, 8
 GetActiveScenario
 Global Functions, 9
 GetActiveScenarioScheduler
 Global Functions, 9
 GetActiveScheduler
 IntScheme, 674
 GetActiveStages
 Global Functions, 9
 GetActiveStudyCase
 Global Functions, 10
 GetAll
 ComNmink, 432
 ElmArea, 170
 ElmFeeder, 192
 ElmZone, 291
 IntDataset, 615
 SetPath, 553
 SetSelect, 560
 StaCubic, 295
 GetAllUsers
 Global Functions, 10
 GetAnalogueDescriptions
 IntComtrade, 599
 IntComtradeset, 608
 GetAttributeEnumerationType
 CimModel, 576
 CimObject, 580
 GetAvailableButtons
 SetTboxconfig, 560
 GetAvailableGenPower
 ElmAsm, 172
 ElmAsmsc, 175
 ElmGenstat, 198
 ElmPvsys, 217

ElmSym, 257
GetAxisX
 PltLinebarplot, 694
GetAxisY
 PltLinebarplot, 694
GetBbOrder
 ElmBbone, 178
GetBlockedSwitches
 ComShctrace, 454
GetBorderCubicles
 Global Functions, 10
GetBranch
 StaCubic, 296
GetBranches
 ElmArea, 170
 ElmFeeder, 193
 ElmZone, 292
 SetPath, 553
GetBrowserSelection
 Global Functions, 11
GetBuses
 ElmArea, 171
 ElmFeeder, 193
 ElmZone, 292
 SetPath, 554
GetBusType
 ElmTerm, 261
GetCalcRelevantCubicles
 ElmTerm, 261
GetCalcRelevantObjects
 Global Functions, 11
GetCalcRX
 ElmRelay, 220
GetCanvasSize
 SetDesktop, 540
GetCaseCommand
 Global Functions, 15
GetCaseObject
 Global Functions, 15
GetCellAccessObject
 ComTablereport, 479
GetCellAlignment
 ComTablereport, 479
GetCellBackgroundColor
 ComTablereport, 480
GetCellColor
 ComTablereport, 480
GetCellEditObjects
 ComTablereport, 481
GetCellFontStyle
 ComTablereport, 481
GetCellFormat
 ComTablereport, 482
GetCellHelpText
 ComTablereport, 482
GetCellValueType

ComTablereport, 483
GetCheckboxCellValue
 ComTablereport, 483
GetCheckSum
 BlkDef, 573
GetChildren
 General Object Methods, 136
GetClass
 General Object Methods, 137
GetClassDescription
 Global Functions, 12
GetClassName
 General Object Methods, 137
GetClassType
 ComCimvalidate, 387
GetColumnFilter
 ComTablereport, 483
GetColumnHeader
 ComTablereport, 484
GetColumnId
 ComTablereport, 484
GetColumnLabel
 IntMat, 636
GetColumnLabelIndex
 IntMat, 637
GetCombinedProjectSource
 General Object Methods, 137
GetCommandsForStudyCase
 ComTasks, 508
GetCompleteBbPath
 ElmBbone, 179
GetConfiguration
 IntSubset, 679
 SetDataext, 537
GetConfigurations
 SetDataext, 537
GetConnectedBranches
 ComUcteexp, 522
GetConnectedBrkCubicles
 ElmTerm, 261
GetConnectedCubicles
 ElmTerm, 262
GetConnectedElements
 General Object Methods, 137
GetConnectedMainBuses
 ElmTerm, 262
GetConnectedMajorNodes
 StaCubic, 296
GetConnectionCount
 General Object Methods, 138
GetConnectionInfo
 ElmTerm, 262
GetConnections
 StaCubic, 297
GetContents
 General Object Methods, 138

GetContingencies
 ComRelreport, 449
 GetContributionOfComponent
 ComRelpost, 448
 ComRelreport, 449
 GetControlledHVNode
 ElmStactrl, 247
 GetControlledLVNode
 ElmStactrl, 247
 GetControlledNode
 General Object Methods, 139
 GetCorrespondingObject
 ComMerge, 426
 GetCreatedBoundaries
 ComBoundary, 381
 GetCreatedObjects
 ComImport, 418
 ComWktmp, 527
 GetCriticalTimePhase
 IntThrating, 680
 GetCubicle
 General Object Methods, 140
 GetCurrentDiagram
 Global Functions, 12
 GetCurrentIteration
 ComGenrelin, 414
 GetCurrentSelection
 Global Functions, 12
 GetCurrentTimeStep
 ComShctrace, 454
 GetCurrentUser
 Global Functions, 12
 GetCurrentZoomScaleLevel
 Global Functions, 13
 GetDataFolder
 Global Functions, 13
 GetDataSeries
 PltLinebarplot, 694
 GetDataSource
 PltDataseries, 693
 VisHrm, 723
 VisPlot, 739
 VisPlot2, 752
 VisXyplot, 766
 GetDateCellValue
 ComTablereport, 485
 GetDerivedProjects
 IntPrj, 656
 IntVersion, 690
 GetDescription
 ElmRes, 231
 Global Functions, 14
 IntComtrade, 599
 IntComtradeset, 608
 GetDescriptionText
 ComCimvalidate, 387
 GetDeviceSwitches
 ComShctrace, 454
 GetDeviceTime
 ComShctrace, 455
 GetDiagramSelection
 Global Functions, 15
 GetDiffMode
 Environment Functions, 58
 GetDigitalDescriptions
 IntComtrade, 599
 IntComtradeset, 608
 GetDirectories
 File System Functions, 41
 GetDisplayedButtons
 SetTboxconfig, 561
 GetDoubleCellValue
 ComTablereport, 485
 GetElecTorque
 ElmAsm, 173
 GetEllaNodeName
 ElmTerm, 265
 GetEquivalentTerminals
 ElmTerm, 263
 GetExternalObject
 ComDpl, 409
 ComPython, 441
 GetExternalReferences
 IntPrj, 656
 GetFaultType
 ComShc, 451
 GetFeeders
 ComOmr, 432
 GetFileDate
 File System Functions, 42
 GetFiles
 File System Functions, 42
 GetFirstValidObject
 ElmRes, 231
 GetFirstValidObjectVariable
 ElmRes, 233
 GetFirstValidVariable
 ElmRes, 233
 GetFlowOrientation
 Global Functions, 15
 GetFOR
 ElmBbone, 179
 GetFromSigName
 BlkSig, 574
 GetFromStudyCase
 Global Functions, 15
 GetFromToNodeNames
 ComUcteexp, 523
 GetFullName
 General Object Methods, 140
 GetGeneratorEvent
 ComContingency, 391

GetGeoCoordinateSystem
 IntPrj, 657

GetGlobalLib
 Global Functions, 16

GetGlobalLibrary
 Global Functions, 16

GetGraphBoard
 Global Functions, 17

GetGraphicsBoard
 Global Functions, 17

GetGroundingImpedance
 ElmAsm, 174
 ElmAsmsc, 176
 ElmFilter, 196
 ElmGenstat, 199
 ElmGndswt, 201
 ElmNec, 213
 ElmPvsys, 217
 ElmShnt, 246
 ElmSym, 258
 ElmTr2, 269
 ElmTr3, 273
 ElmTr4, 277
 ElmVac, 286
 ElmVoltreg, 287
 ElmXnet, 289

GetGroups
 IntUserman, 683

GetHistoricalProject
 IntVersion, 690

GetImpedance
 General Object Methods, 141

GetImportedObjects
 ComPfdimport, 439

GetInnom
 General Object Methods, 142

GetInputObject
 ComCimvalidate, 387

GetInputParameterDouble
 ComDpl, 409
 ComPython, 441

GetInputParameterInt
 ComDpl, 409
 ComPython, 441

GetInputParameterString
 ComDpl, 410
 ComPython, 442

GetInstallationDirectory
 File System Functions, 43

GetInstallDir
 File System Functions, 43

GetIntCalcres
 PltDataseries, 693
 VisPlot, 740

GetIntCellValue
 ComTablereport, 486

GetInterior
 ElmBoundary, 184

GetInterruptedPowerAndCustomersForStage
 ComContingency, 392

GetInterruptedPowerAndCustomersForTimeStep
 ComContingency, 392

GetIthr
 ElmLne, 205

GetLanguage
 Global Functions, 17

GetLastCmd
 Global Functions, 17

GetLatestVersion
 IntPrj, 657

GetLegend
 PltLinebarplot, 695

GetLimit
 ScnFreq, 699
 ScnFr, 701
 ScnSpeed, 703
 ScnSync, 705
 ScnVar, 707
 ScnVolt, 709

GetLineCable
 ElmCabsys, 186

GetLoadEvent
 ComContingency, 393

GetLocalLib
 Global Functions, 18

GetLocalLibrary
 Global Functions, 18

GetMaxFdetectCalcl
 ElmRelay, 220

GetMaxNumIterations
 ComGenrelin, 415

GetMeanCs
 ElmBbone, 180

GetMeaValue
 StaExtbrkmea, 302
 StaExtcmdmea, 307
 StaExtdatmea, 312
 StaExtfmea, 316
 StaExtfuelmea, 321
 StaExtmea, 326
 StaExtpfmea, 331
 StaExtpmea, 336
 StaExtqmea, 341
 StaExttapmea, 351
 StaExtv3mea, 356
 StaExtvmea, 361

GetMechTorque
 ElmAsm, 174

GetMem
 Global Functions, 18

GetMicroSCADASatus
 ComLink, 422

GetMinCs
 ElmBbone, 180

GetMinDistance
 ElmTerm, 264

GetModel
 ComCimvalidate, 388

GetModelId
 ComCimvalidate, 388

GetModelsReferencingThis
 CimModel, 576

GetModification
 ComMerge, 426

GetModificationResult
 ComMerge, 427

GetModifiedObjects
 ComImport, 418
 ComMerge, 427
 ComWktmp, 528

GetMotorConnections
 ComMot, 430

GetMotorStartingFlag
 ElmAsm, 174
 ElmSym, 259

GetMotorSwitch
 ComMot, 430

GetMotorTerminal
 ComMot, 430

GetNearestBusbars
 StaCubic, 297

GetNet
 General Object Methods, 142

GetNextHVBus
 ElmTerm, 264

GetNextLoop
 ComCheck, 385

GetNextValidObject
 ElmRes, 234

GetNextValidObjectVariable
 ElmRes, 235

GetNextValidVariable
 ElmRes, 236

getNode
 General Object Methods, 143

GetnodeName
 ElmTerm, 265

GetNodesBranches
 ElmFeeder, 194

GetNonStartedDevices
 ComShctrace, 455

GetNTopLoadedElms
 ComSimoutage, 462

GetNumberOfAnalogueSignalDescriptions
 IntComtrade, 600
 IntComtradeset, 609

GetNumberOfClusters
 SetCluster, 529

GetNumberOfColumns
 ComTablereport, 486
 ElmRes, 236
 IntComtrade, 600
 IntComtradeset, 609
 IntMat, 637

GetNumberOfCommandsForStudyCase
 ComTasks, 509

GetNumberOfDigitalSignalDescriptions
 IntComtrade, 600
 IntComtradeset, 609

GetNumberOfGeneratorEventsForTimeStep
 ComContingency, 393

GetNumberOfLoadEventsForTimeStep
 ComContingency, 393

GetNumberOfRows
 ComTablereport, 486
 ElmRes, 236
 IntComtrade, 600
 IntComtradeset, 609
 IntMat, 637

GetNumberOfSelectedCells
 ComTablereport, 487

GetNumberOfStudyCases
 ComTasks, 509

GetNumberOfSwitchEventsForTimeStep
 ComContingency, 394

GetNumberOfTimeSteps
 ComContingency, 394

GetNumberOfValidationMessages
 ComCimvalidate, 388

GetNumberOfViolations
 ScnFreq, 699
 ScnFrT, 701
 ScnSpeed, 703
 ScnSync, 705
 ScnVar, 707
 ScnVolt, 709

GetNumProcesses
 SetUser, 563

GetNumSlave
 SetParalman, 550

GetObj
 ComContingency, 394
 ElmRes, 236

GetObject
 ComCimvalidate, 388
 ComOutage, 436
 ElmRes, 237

GetObjectCellValue
 ComTablereport, 487

GetObjectId
 ComCimvalidate, 389

GetObjects
 IntScenario, 669
 IntSubset, 680

GetObjectsReferencingThis
 CimObject, 580

GetObjectsWithSameId
 CimObject, 581

GetObjectValue
 ElmRes, 237
 IntComtrade, 601
 IntComtradeset, 609

GetObjs
 ElmArea, 171
 ElmFeeder, 195
 ElmZone, 293

GetOMR
 ComOmr, 433

GetOperationValue
 IntScenario, 670

GetOperator
 General Object Methods, 143

GetOrderCode
 ComUcteexp, 524

GetOrInsertCurvePlot
 GrpPage, 585

GetOrInsertDiscreteBarPlot
 GrpPage, 586

GetOrInsertPlot
 SetVipage, 565

GetOrInsertXYPlot
 GrpPage, 586

GetOverLoadedBranches
 ComShc, 451

GetOverLoadedBuses
 ComShc, 452

GetOwner
 General Object Methods, 143

GetPage
 SetDesktop, 541

GetPageLen
 Global Functions, 19

GetParameterCount
 CimModel, 576
 CimObject, 581

GetParameterNamespace
 CimModel, 576
 CimObject, 581

GetParameterValue
 CimModel, 577
 CimObject, 581

GetParent
 General Object Methods, 144

GetPathFolder
 SetPath, 554

GetPathToNearestBusbar
 StaCubic, 298

GetPfObjects
 CimObject, 582

GetPFVersion

Global Functions, 19

GetPlot
 GrpPage, 586

GetProfile
 ComCimvalidate, 389

GetProjectFolder
 Global Functions, 19

GetProjectFolderType
 IntPrjfolder, 663

GetQlim
 IntQlim, 665

GetRandomNumber
 Mathematical Functions, 69

GetRandomNumberEx
 Mathematical Functions, 70

GetRating
 IntThrating, 681

GetRecordingStage
 Global Functions, 20

GetReferences
 General Object Methods, 144

GetRegion
 General Object Methods, 145

GetRegionCount
 ComOmr, 434

GetRelCase
 ElmRes, 238

GetRemoteBorderCubicles
 StaCubic, 299

GetRemoteBreakers
 ElmCoup, 188

GetResData
 Global Functions, 20

GetResDesc
 Global Functions, 20

GetResObj
 Global Functions, 20

GetResUnit
 Global Functions, 21

GetResVar
 Global Functions, 21

GetRowCaption
 ComTablereport, 488

GetRowId
 ComTablereport, 488

GetRowLabel
 IntMat, 638

GetRowLabelIndex
 IntMat, 638

GetScaleObjX
 VisHrm, 724
 VisPlot, 740
 VisPlot2, 753

GetScaleObjY
 VisHrm, 724
 VisPlot, 740

VisPlot2, 754
 GetScenario
 IntScensched, 672
 GetScheme
 IntSstage, 677
 GetSelectedCell
 ComTablereport, 488
 GetSelectedElements
 ComTablereport, 489
 GetSelection
 ComTablereport, 489
 GetSepStationAreas
 ElmTerm, 265
 GetSettings
 Global Functions, 21
 GetSeverity
 ComCimvalidate, 389
 GetSignalHeader
 IntComtrade, 601
 IntComtradeset, 610
 GetSimulationTime
 ComSim, 456
 GetSize
 General Object Methods, 146
 GetSlot
 ElmRelay, 221
 GetSplit
 ElmSubstat, 249
 ElmTrfstat, 281
 GetSplitCal
 ElmSubstat, 251
 ElmTrfstat, 283
 GetSplitIndex
 ElmSubstat, 252
 ElmTrfstat, 284
 GetStartedDevices
 ComShctrace, 455
 GetStartEndTime
 IntScensched, 672
 GetStatus
 StaExtbrkmea, 302
 StaExtcmdmea, 307
 StaExtdatmea, 312
 StaExtfmea, 317
 StaExtfuelmea, 322
 StaExtmea, 327
 StaExtpfmea, 331
 StaExtpmea, 336
 StaExtqmea, 341
 StaExtmsmea, 346
 StaExttapmea, 351
 StaExtv3mea, 356
 StaExtvmea, 361
 GetStatusTmp
 StaExtbrkmea, 302
 StaExtcmdmea, 307
 StaExtdatmea, 312
 StaExtfmea, 317
 StaExtfuelmea, 322
 StaExtmea, 327
 StaExtpfmea, 332
 StaExtpmea, 337
 StaExtqmea, 342
 StaExtmsmea, 346
 StaExttapmea, 352
 StaExtv3mea, 357
 StaExtvmea, 362
 GetStepupTransformer
 ElmAsm, 175
 ElmAsmsc, 177
 ElmGenstat, 199
 ElmStactrl, 248
 ElmSvs, 255
 ElmSym, 259
 ElmXnet, 289
 GetStringCellValue
 ComTablereport, 489
 GetStudyCases
 ComTasks, 509
 GetStudyTimeObject
 Date/Time Functions, 46
 GetSubElmRes
 ElmRes, 239
 GetSuccessfullyConnectedItems
 ComPrjconnector, 439
 GetSummaryGrid
 Global Functions, 22
 GetSuppliedElements
 ElmSubstat, 253
 ElmTr2, 269
 ElmTr3, 273
 ElmTr4, 278
 ElmTrfstat, 285
 GetSupplyingSubstations
 General Object Methods, 147
 GetSupplyingTransformers
 General Object Methods, 147
 GetSupplyingTrfstations
 General Object Methods, 147
 GetSwitch
 StaCubic, 300
 GetSwitchEvent
 ComContingency, 394
 GetSwitchStatus
 IntRunarrange, 667
 GetSwitchTime
 ComShctrace, 455
 GetSystemGround
 General Object Methods, 147
 GetSystemGrounding
 General Object Methods, 147
 GetSystemTime

Date/Time Functions, 46
GetSystemTimeUTC
 Date/Time Functions, 46
GetTableReports
 Dialogue Boxes Functions, 50
GetTapPhi
 ElmTr2, 270
 ElmTr3, 274
 ElmTr4, 278
GetTapRatio
 ElmTr2, 270
 ElmTr3, 274
 ElmTr4, 279
GetTapZDependentSide
 ElmTr3, 275
 ElmTr4, 279
GetTempDir
 File System Functions, 43
GetTemporaryDirectory
 File System Functions, 43
GetTieOpenPoint
 ElmBbone, 181
GetTime
 Date/Time Functions, 47
GetTimeOfStepInSeconds
 ComContingency, 395
GetTitleObject
 PltLinebarplot, 695
GetToSigName
 BlkSig, 574
GetTotalInterruptedPower
 ComContingency, 395
GetTotalWarnA
 ComSim, 456
GetTotalWarnB
 ComSim, 456
GetTotalWarnC
 ComSim, 457
GetTotLength
 ElmBbone, 181
GetTransferCalcData
 ComTransfer, 519
GetTrippedDevices
 ComShctrace, 455
GetType
 ComCimvalidate, 389
 ElmLne, 205
GetUcteNodeName
 ComUcteexp, 525
GetUnit
 ElmRes, 239
 IntComtrade, 601
 IntComtradeset, 610
GetUnom
 ElmRelay, 221
 General Object Methods, 148
GetUnsuccessfullyConnectedItems
 ComPrjconnector, 439
GetUserManager
 Global Functions, 22
GetUsers
 IntUserman, 683
GetVal
 General Object Methods, 149
GetValue
 ElmRes, 240
 IntComtrade, 602
 IntComtradeset, 611
 IntDplmap, 618
 ScnFreq, 699
 ScnFrt, 701
 ScnSpeed, 703
 ScnSync, 705
 ScnVar, 707
 ScnVolt, 709
GetVar
 IntMon, 644
GetVariable
 ElmRes, 240
 IntComtrade, 602
 IntComtradeset, 611
 ScnFreq, 699
 ScnFrt, 702
 ScnSpeed, 704
 ScnSync, 706
 ScnVar, 708
 ScnVolt, 710
GetVariation
 IntSstage, 677
GetVarType
 General Object Methods, 149
GetVersions
 IntPrj, 657
GetVI
 SetVipage, 565
GetViolatedElement
 ScnFreq, 700
 ScnFrt, 702
 ScnSpeed, 704
 ScnSync, 706
 ScnVar, 708
 ScnVolt, 710
GetViolatedScanModules
 ComSim, 457
GetViolationTime
 ScnFreq, 700
 ScnFrt, 702
 ScnSpeed, 704
 ScnSync, 706
 ScnVar, 708
 ScnVolt, 710
GetWorkingDir

File System Functions, 43
 GetWorkspaceDirectory
 File System Functions, 43
 GetY0m
 ElmLne, 205
 GetY1m
 ElmLne, 206
 GetZ0m
 ElmLne, 207
 GetZ0pu
 ElmTr2, 271
 ElmTr3, 275
 ElmTr4, 279
 GetZ1m
 ElmLne, 208
 GetZeroImpedance
 General Object Methods, 150
 GetZeroSequenceHVLVT
 TypTr2, 715
 GetZmatDist
 ElmLne, 208
 GetZpu
 ElmTr2, 271
 ElmTr3, 275
 ElmTr4, 280
 ElmVoltreg, 287
 Global Functions, 2
 ActivateProject, 3
 ActiveCase, 10
 ActiveProject, 8
 AllRelevant, 11
 ClearCommands, 4
 ClearRecycleBin, 4
 CommitTransaction, 4
 CommitTx_, 4
 CreateFaultCase, 4
 CreateProject, 5
 Delete, 5
 DeleteUntouchedObjects, 6
 Exe, 6
 ExecuteCmd, 6
 exit, 6
 GetActiveCalculationStr, 7
 GetActiveNetworkVariations, 8
 GetActiveProject, 8
 GetActiveScenario, 9
 GetActiveScenarioScheduler, 9
 GetActiveStages, 9
 GetActiveStudyCase, 10
 GetAllUsers, 10
 GetBorderCubicles, 10
 GetBrowserSelection, 11
 GetCalcRelevantObjects, 11
 GetCaseCommand, 15
 GetCaseObject, 15
 GetClassDescription, 12
 GetCurrentDiagram, 12
 GetCurrentSelection, 12
 GetCurrentUser, 12
 GetCurrentZoomScaleLevel, 13
 GetDataFolder, 13
 GetDescription, 14
 GetDiagramSelection, 15
 GetFlowOrientation, 15
 GetFromStudyCase, 15
 GetGlobalLib, 16
 GetGlobalLibrary, 16
 GetGraphBoard, 17
 GetGraphicsBoard, 17
 GetLanguage, 17
 GetLastCmd, 17
 GetLocalLib, 18
 GetLocalLibrary, 18
 GetMem, 18
 GetPageLen, 19
 GetPFVersion, 19
 GetProjectFolder, 19
 GetRecordingStage, 20
 GetResData, 20
 GetResDesc, 20
 GetResObj, 20
 GetResUnit, 21
 GetResVar, 21
 GetSettings, 21
 GetSummaryGrid, 22
 GetUserManager, 22
 HttpGet, 22
 ImportDz, 23
 ImportSnapshot, 24
 IsLdfValid, 24
 IsRmsValid, 25
 IsScenarioAttribute, 25
 IsShcValid, 26
 IsSimValid, 26
 LoadProfile, 26
 LoadResData, 27
 MarkInGraphics, 27
 OutputFlexibleData, 28
 PostCommand, 28
 PrepForUntouchedDelete, 29
 Rebuild, 29
 ReleaseResData, 29
 ReloadProfile, 29
 ResetCalculation, 29
 ResFirstValidObject, 30
 ResFirstValidObjectVar, 30
 ResFirstValidVar, 30
 ResGetMax, 30
 ResGetMin, 30
 ResIndex, 31
 ResNextValidObject, 31
 ResNextValidObjectVar, 31

ResNextValidVar, 31
 ResNval, 31
 ResNvars, 31
 ResSortToVar, 32
 SaveAsScenario, 32
 SaveScenarioAs, 32
 SearchObjectByForeignKey, 32
 SelectToolbox, 33
 SetShowAllUsers, 33
 Sleep, 34
 SplitLine, 34
 StatFileGetXrange, 34
 StatFileResetXrange, 35
 StatFileSetXrange, 35
 SummaryGrid, 22
 validLDF, 24
 validRMS, 25
 validSHC, 26
 validSIM, 26
GoToCell
 ComTablereport, 490
GrpPage, 584
 DoAutoScale, 585
 DoAutoScaleX, 585
 DoAutoScaleY, 585
 GetOrInsertCurvePlot, 585
 GetOrInsertDiscreteBarPlot, 586
 GetOrInsertXYPlot, 586
 GetPlot, 586
 RemovePage, 587
 SetAutoScaleModeX, 587
 SetAutoScaleModeY, 587
 SetLayoutMode, 588
 SetResults, 588
 SetScaleTypeX, 588
 SetScaleTypeY, 588
 SetScaleX, 589
 SetScaleY, 589
 Show, 589

HasCell
 ComTablereport, 490
HasColumnAutoFilter
 ComTablereport, 490
HasCreatedCalBus
 ElmTerm, 266
HasExternalReferences
 IntPrj, 658
HasGnrlMod
 ElmBbone, 182
HasParameter
 CimModel, 577
 CimObject, 582
HasResults
 General Object Methods, 150
HasResultsForDirectionalBackup
 ComCoordreport, 397
 HasResultsForNonDirectionalBackup
 ComCoordreport, 397
HasResultsForOverreach
 ComCoordreport, 398
HasResultsForZone
 ComCoordreport, 398
HasRoutes
 ElmLne, 209
HasRoutesOrSec
 ElmLne, 209
HttpGet
 Global Functions, 22

Import
 IntDocument, 615
 IntGrfgroup, 629
 IntGrflayer, 631
 IntlIcon, 634
ImportAndConvert
 ComCimdbimp, 386
ImportDz
 Global Functions, 23
ImportFromVec
 IntGrfgroup, 630
 IntGrflayer, 631
ImportSnapshot
 Global Functions, 24
IndexOf
 IntDplvec, 624
Info
 ElmStactrl, 248
 Output Window Functions, 118
Init
 ElmRes, 241
 IntMat, 639
 IntVec, 684
InitialiseWriting
 ElmRes, 241
InitQuickAccess
 ComUcteexp, 526
InitTmp
 StaExtbrkmea, 302
 StaExtcmdmea, 307
 StaExtdatmea, 312
 StaExtfmea, 317
 StaExtfuelmea, 322
 StaExttimea, 327
 StaExtpfmea, 332
 StaExtpmea, 337
 StaExtqmea, 342
 StaExtsmea, 346
 StaExttapmea, 352
 StaExtv3mea, 357
 StaExtvmea, 362
Inom

General Object Methods, 142
input
 Dialogue Boxes Functions, 50
Insert
 IntDplmap, 621
 IntDplvec, 625
InsertPlot
 SetVipage, 566
IntAddonvars, 590
 AddDouble, 590
 AddDoubleMatrix, 590
 AddDoubleVector, 591
 AddInteger, 591
 AddIntegerVector, 592
 AddObject, 592
 AddObjectVector, 593
 AddString, 593
 RemoveParameter, 594
IntCase, 594
 Activate, 594
 ApplyNetworkState, 594
 ApplyStudyTime, 595
 Consolidate, 595
 Deactivate, 595
 SetStudyTime, 595
IntComtrade, 596
 ConvertToASCIIFormat, 596
 ConvertToBinaryFormat, 596
 FindColumn, 597
 FindMaxInColumn, 597
 FindMinInColumn, 598
 GetAnalogueDescriptions, 599
 GetDescription, 599
 GetDigitalDescriptions, 599
 GetNumberOfAnalogueSignalDescriptions, 600
 GetNumberOfColumns, 600
 GetNumberOfDigitalSignalDescriptions, 600
 GetNumberOfRows, 600
 GetObjectValue, 601
 GetSignalHeader, 601
 GetUnit, 601
 GetValue, 602
 GetVariable, 602
 Load, 603
 NCol, 600
 NRow, 600
 Release, 604
 SizeX, 600
 SizeY, 600
 SortAccordingToColumn, 604
IntComtradeset, 605
 FindColumn, 606
 FindMaxInColumn, 606
 FindMinInColumn, 607
 GetAnalogueDescriptions, 608
 GetDescription, 608
 GetDigitalDescriptions, 608
 GetNumberOfAnalogueSignalDescriptions, 609
 GetNumberOfColumns, 609
 GetNumberOfDigitalSignalDescriptions, 609
 GetNumberOfRows, 609
 GetObjectValue, 609
 GetSignalHeader, 610
 GetUnit, 610
 GetValue, 611
 GetVariable, 611
 Load, 612
 NCol, 609
 NRow, 609
 Release, 613
 SizeX, 609
 SizeY, 609
 SortAccordingToColumn, 613
IntDataset, 614
 AddRef, 614
 All, 614
 Clear, 615
 GetAll, 615
IntDocument, 615
 Export, 615
 Import, 615
 Reset, 616
 View, 616
IntDplmap, 616
 Clear, 616
 Contains, 617
 First, 617
 GetValue, 618
 Insert, 621
 Next, 621
 Remove, 622
 Size, 622
 Update, 622
IntDplvec, 623
 Clear, 623
 Get, 623
 IndexOf, 624
 Insert, 625
 Remove, 625
 Size, 625
 Sort, 626
IntEvt, 626
 CreateCBEvents, 626
 RemoveSwitchEvents, 627
IntExtaccess, 627
 CheckUrl, 627
IntGate, 628
 AddTrigger, 628
IntGrf, 628
 MoveToLayer, 628

IntGrfgroup, 628
 ClearData, 628
 Export, 629
 ExportToVec, 629
 Import, 629
 ImportFromVec, 630
IntGrflayer, 630
 ClearData, 630
 Export, 630
 ExportToVec, 631
 Import, 631
 ImportFromVec, 631
IntGrfnet, 632
 Close, 632
 SetLayerVisibility, 632
 SetSymbolComponentVisibility, 632
 Show, 633
IntlIcon, 633
 Export, 634
 Import, 634
IntLibrary, 634
 Activate, 634
 Deactivate, 635
IntMat, 635
 CollBl, 635, 636
 Get, 635
 GetColumnLabel, 636
 GetColumnLabelIndex, 637
 GetNumberOfColumns, 637
 GetNumberOfRows, 637
 GetRowLabel, 638
 GetRowLabelIndex, 638
 Init, 639
 Invert, 639
 Multiply, 640
 NCol, 637
 NRow, 637
 Resize, 640
 RowLbl, 638, 640
 Save, 641
 Set, 641
 SetColumnLabel, 641
 SetRowLabel, 642
 SizeX, 637
 SizeY, 637
 SortToColum, 642
 SortToColumn, 642
IntMon, 643
 AddVar, 643
 AddVars, 643
 ClearVars, 644
 GetVar, 644
 NVars, 645
 PrintAllVal, 645
 PrintVal, 645
 RemoveVar, 646
IntOutage, 646
 Apply, 647
 ApplyAll, 647
 Check, 647
 CheckAll, 647
 IsInStudyTime, 648
 IsInStudytime, 648
 ResetAll, 648
IntPlannedout, 649
 SetRecurrence, 649
IntPlot, 649
 SetAdaptY, 649
 SetAutoScaleY, 650
 SetScaleY, 651
IntPrj, 652
 Activate, 652
 AddProjectToCombined, 652
 AddProjectToRemoteDatabase, 653
 Archive, 653
 BeginDataExtensionModification, 653
 CanAddProjectToRemoteDatabase, 653
 CanSubscribeProjectReadOnly, 654
 CanSubscribeProjectReadWrite, 654
 ClearInvalidReferences, 654
 CopyDataExtensionFrom, 654
 CreateVersion, 655
 Deactivate, 656
 EndDataExtensionModification, 656
 GetDerivedProjects, 656
 GetExternalReferences, 656
 GetGeoCoordinateSystem, 657
 GetLatestVersion, 657
 GetVersions, 657
 HasExternalReferences, 658
 LoadData, 658
 MergeToBaseProject, 658
 Migrate, 659
 NormaliseCombined, 659
 PackExternalReferences, 659
 Purge, 660
 RemoveProjectFromCombined, 660
 Restore, 660
 SetGeoCoordinateSystem, 660
 SubscribeProjectReadOnly, 661
 SubscribeProjectReadWrite, 661
 TransformGeoCoordinates, 661
 UnsubscribeProject, 662
 UpdateStatistics, 662
 UpdateToDefaultStructure, 662
 UpdateToMostRecentBaseVersion, 662
IntPrjfolder, 663
 GetProjectFolderType, 663
 IsProjectFolderType, 665
IntQlim, 665
 GetQlim, 665
IntRas, 666

AddEvent, 666
 AddTrigger, 666
 IsValid, 667
IntRunarrange, 667
 GetSwitchStatus, 667
IntScenario, 668
 Activate, 668
 Apply, 668
 ApplySelective, 669
 Deactivate, 669
 DiscardChanges, 669
 GetObjects, 669
 GetOperationValue, 670
 ReleaseMemory, 670
 Save, 670
 SetOperationValue, 671
IntScensched, 671
 Activate, 671
 Deactivate, 671
 DeleteRow, 672
 GetScenario, 672
 GetStartTime, 672
 SearchScenario, 672
IntScheme, 673
 Activate, 673
 Consolidate, 673
 Deactivate, 673
 GetActiveScheduler, 674
 NewStage, 674
IntScheduler, 674
 Activate, 674
 Deactivate, 675
 Update, 675
IntStage, 675
 Activate, 675
 CreateStageObject, 676
 EnableDiffMode, 676
 GetScheme, 677
 GetVariation, 677
 IsExcluded, 677
 PrintModifications, 677
 ReadValue, 678
 WriteValue, 678
IntSubset, 678
 Apply, 679
 ApplySelective, 679
 Clear, 679
 GetConfiguration, 679
 GetObjects, 680
IntThrating, 680
 GetCriticalSectionPhase, 680
 GetRating, 681
IntlUrl, 681
 View, 681
IntlUser, 682
 Purge, 682
 SetPassword, 682
 TerminateSession, 682
IntlUserman, 682
 CreateGroup, 683
 CreateUser, 683
 GetGroups, 683
 GetUsers, 683
 UpdateGroups, 684
IntlVec, 684
 Get, 684
 Init, 684
 Max, 685
 Mean, 685
 Min, 685
 Resize, 685
 Save, 686
 Set, 686
 Size, 686
 Sort, 686
IntlVecobj, 687
 Get, 688
 Resize, 688
 Save, 688
 Search, 688
 Set, 688
 Size, 689
IntlVersion, 689
 CreateDerivedProject, 689
 GetDerivedProjects, 690
 GetHistoricalProject, 690
 Rollback, 690
IntlViewbookmark, 691
 JumpTo, 691
 UpdateFromCurrentView, 691
Invert
 IntMat, 639
InvertMatrix
 Mathematical Functions, 71
IsAC
 ComLdf, 421
IsAdditionalResultsFlagSetForCommand
 ComTasks, 510
IsAutomaticCalculationResetEnabled
 Environment Functions, 58
IsBalanced
 ComLdf, 421
IsBreaker
 ElmCoup, 189
IsCable
 ElmLne, 210
 ElmLnesec, 212
 TypLne, 713
IsCalcRelevant
 General Object Methods, 151
IsClass
 General Object Methods, 152

IsClosed
 ElmCoup, 189
 ElmGndswt, 202
 StaCubic, 300
 StaSwitch, 366
IsColumnScrollable
 ComTablereport, 491
IsColumnSortable
 ComTablereport, 491
IsCommandIgnored
 ComTasks, 510
IsConnected
 ElmGenstat, 200
 ElmPvsys, 218
 ElmSym, 259
 StaCubic, 300
IsDC
 ComLdf, 421
IsDeleted
 General Object Methods, 152
IsEarthed
 General Object Methods, 153
IsElectrEquivalent
 ElmTerm, 266
IsEncrypted
 ComDpl, 410
 TypQdsl, 714
IsEnergized
 General Object Methods, 153
IsEquivalent
 ElmTerm, 267
IsExcluded
 IntStage, 677
IsFinalEchoOnEnabled
 Environment Functions, 59
IsFrozen
 SetDesktop, 542
IsHidden
 General Object Methods, 154
IsIn
 Set Routines, 771
IsInFeeder
 General Object Methods, 154
IsInStudyTime
 IntOutage, 648
IsInStudytime
 IntOutage, 648
IsInternalNodeInStation
 ElmTerm, 268
IsInternalNodeInSubStation
 ElmTerm, 268
IsLastIterationFeasible
 ComTransfer, 519
IsLdfValid
 Global Functions, 24
IsNetCoupling
 ElmLne, 210
IsNetworkDataFolder
 General Object Methods, 155
IsNode
 General Object Methods, 156
IsObjectActive
 General Object Methods, 156
IsObjectModifiedByVariation
 General Object Methods, 156
Isolate
 General Object Methods, 157
IsOpen
 ElmCoup, 189
 ElmGndswt, 202
 StaSwitch, 367
IsOpened
 SetDesktop, 542
IsOutOfService
 General Object Methods, 157
IsPQ
 ElmAsm, 175
IsProjectFolderType
 IntPrjfolder, 665
IsQuadBooster
 ElmTr2, 271
 ElmTr3, 276
 ElmTr4, 280
IsReducible
 General Object Methods, 158
IsRelevant
 General Object Methods, 151
IsRmsValid
 Global Functions, 25
IsRowVisible
 ComTablereport, 491
IsScenarioAttribute
 Global Functions, 25
IsShcValid
 Global Functions, 26
IsShortCircuited
 General Object Methods, 158
IsSimValid
 Global Functions, 26
IsSplitting
 ElmBoundary, 184
IsStarted
 ElmRelay, 221
IsStatusBitSet
 StaExtbrkmea, 302
 StaExtcmdmea, 307
 StaExtdatmea, 313
 StaExtfmea, 317
 StaExtfuelmea, 322
 StaExttimea, 327
 StaExtpfmea, 332
 StaExtptmea, 337

StaExtqmea, 342
 StaExtsmea, 347
 StaExttapmea, 352
 StaExtv3mea, 357
 StaExtvmea, 362
 IsStatusBitSetTmp
 StaExtbrkmea, 303
 StaExtcmdmea, 308
 StaExtdatmea, 313
 StaExtfmea, 318
 StaExtfuelmea, 322
 StaExttimea, 327
 StaExtpfmea, 332
 StaExtppmea, 337
 StaExtqmea, 342
 StaExtsmea, 347
 StaExttapmea, 352
 StaExtv3mea, 357
 StaExtvmea, 362
 IsStudyCaselnored
 ComTasks, 511
 IsValid
 IntRas, 667
 JumpTo
 IntViewbookmark, 691
 JumpToLastStep
 ComContingency, 395
 In
 Mathematical Functions, 73
 Inm
 General Object Methods, 158
 Load
 ElmRes, 242
 IntComtrade, 603
 IntComtradeset, 612
 LoadData
 IntPrj, 658
 LoadFile
 ElmFile, 195
 LoadMicroSCADAFile
 ComLink, 422
 LoadProfile
 Global Functions, 26
 LoadResData
 Global Functions, 27
 LoadSimulationState
 ComSim, 457
 LoadSnapshot
 ComSim, 457
 log
 Mathematical Functions, 73
 LossCostAtBusTech
 ComCapo, 382
 Mark

SetLevelvis, 550
 MarkInGraphics
 General Object Methods, 159
 Global Functions, 27
 Set Routines, 771
 MarkRegions
 ComSimoutage, 462
 Mathematical Functions, 64
 abs, 65
 acos, 65
 asin, 65
 atan, 65
 atan2, 65
 BinaryAnd, 66
 BinaryOr, 66
 CalcWeibullPar, 67
 ceil, 68
 cos, 68
 cosh, 68
 exp, 69
 floor, 69
 frac, 69
 fRand, 70
 GetRandomNumber, 69
 GetRandomNumberEx, 70
 InvertMatrix, 71
 In, 73
 log, 73
 MatrixInvert, 71
 max, 74
 min, 74
 modulo, 74
 pi, 74
 pow, 75
 Random, 69
 RndExp, 75
 RndGetMethod, 75
 RndGetSeed, 76
 RndNormal, 76
 RndSetup, 76
 RndUnifInt, 78
 RndUnifReal, 79
 RndWeibull, 79
 round, 80
 SetRandomSeed, 80
 SetRandSeed, 80
 sin, 80
 sinh, 80
 sqr, 81
 sqrt, 81
 tan, 81
 tanh, 81
 time, 82
 trunc, 82
 twopi, 82
 MatrixInvert

Mathematical Functions, 71
Max
 IntVec, 685
max
 Mathematical Functions, 74
MaxZoneNumberFor
 ComCoordreport, 398
mbprintf
 Output Window Functions, 118
mbschg
 Multibyte Encoded String Functions, 114
mbscmp
 Multibyte Encoded String Functions, 114
mbscpy
 Multibyte Encoded String Functions, 114
mbslen
 Multibyte Encoded String Functions, 115
mbsprintf
 Multibyte Encoded String Functions, 115
mbsscanf
 Multibyte Encoded String Functions, 115
mbsstr
 Multibyte Encoded String Functions, 115
mbstok
 Multibyte Encoded String Functions, 116
mdbClose
 MS Access Functions, 83
mdbExecuteSqlQuery
 MS Access Functions, 83
mdbExecuteSqlStatement
 MS Access Functions, 83
mdbFetchResult
 MS Access Functions, 84
mdbGetResultColumnCount
 MS Access Functions, 84
mdbGetResultColumnName
 MS Access Functions, 85
mdbGetResultColumnType
 MS Access Functions, 85
mdbGetResultColumnNameValue
 MS Access Functions, 86
mdbOpen
 MS Access Functions, 86
mdbSetDebug
 MS Access Functions, 90
Mean
 IntVec, 685
MeasureLength
 ElmLne, 211
Merge
 ComMerge, 428
MergeToBaseProject
 IntPrj, 658
MessageBox
 Dialogue Boxes Functions, 51
Migrate
 IntPrj, 659
MigratePage
 SetVipage, 566
Min
 IntVec, 685
min
 Mathematical Functions, 74
ModuleExists
 ComAddon, 379
modulo
 Mathematical Functions, 74
Move
 General Object Methods, 160
MoveToLayer
 IntGrf, 628
MS Access Functions, 82
 mdbClose, 83
 mdbExecuteSqlQuery, 83
 mdbExecuteSqlStatement, 83
 mdbFetchResult, 84
 mdbGetResultColumnCount, 84
 mdbGetResultColumnName, 85
 mdbGetResultColumnType, 85
 mdbGetResultColumnNameValue, 86
 mdbOpen, 86
 mdbSetDebug, 90
MS Excel Functions, 91
 xlActivateWorksheet, 92
 xlAddWorksheet, 92
 xlCloseWorkbook, 92
 xlDeleteWorksheet, 93
 xlGetActiveWorksheetIndex, 93
 xlGetDateSeparator, 94
 xlGetDecimalSeparator, 94
 xlGetThousandsSeparator, 94
 xlGetValue, 94
 xlGetWorksheetCount, 95
 xlGetWorksheetName, 95
 xlNewWorkbook, 96
 xlOpenWorkbook, 96
 xlResetTextStyle, 96
 xlRunMacro, 97
 xlSaveWorkbook, 98
 xlSaveWorkbookAs, 98
 xlSetBorder, 99
 xlSetColumnWidth, 100
 xlSetDebug, 100
 xlSetFillColor, 101
 xlSetFontName, 102
 xlFontSize, 102
 xlSetHorizontalAlignment, 103
 xlSetNumberFormat, 104
 xlSetPrintTitleRows, 104
 xlSetRowHeight, 105
 xlSetTextColor, 105
 xlSetTextStyle, 106

xlSetValue, 107
 xlSetValues, 107
 xlSetVerticalAlignment, 108
 xlSetVisible, 109
 xlSetWorksheetName, 109
 xlSetWrapText, 110
 xlStart, 110
 xlTerminate, 113
 MS Office Functions, 82
 Multibyte Encoded String Functions, 114
 mbschg, 114
 mbscmp, 114
 mbscpy, 114
 mbslen, 115
 mbsprintf, 115
 mbsscanf, 115
 mbstrstr, 115
 mbstok, 116
 tombchar, 116
 tombcharcodepoint, 116
 tombslower, 116
 tombsupper, 117
 Multiply
 IntMat, 640
 NCol
 ComTablereport, 486
 ElmRes, 236
 IntComtrade, 600
 IntComtradeset, 609
 IntMat, 637
 Network Elements Methods, 169
 NewStage
 IntScheme, 674
 Next
 IntDplmap, 621
 Set Routines, 771
 NextFilt
 Set Routines, 772
 NextStepAvailable
 ComShctrace, 456
 NoFinalUpdate
 Environment Functions, 59
 NormaliseCombined
 IntPrj, 659
 NRow
 ComTablereport, 486
 ElmRes, 236
 IntComtrade, 600
 IntComtradeset, 609
 IntMat, 637
 NTap
 ElmTr2, 272
 ElmTr3, 276
 ElmTr4, 281
 ElmVoltreg, 287
 NVars
 IntMon, 645
 Obj
 Set Routines, 772
 Object Methods, 132
 Open
 ElmCoup, 190
 ElmGndswt, 202
 StaSwitch, 367
 Other Objects Methods, 572
 Output Window Functions, 117
 ClearOutput, 117
 ClearOutputWindow, 117
 Error, 117
 fWrite, 119
 Info, 118
 mbprintf, 118
 printf, 119
 SetLineFeed, 120
 SetOutputWindowState, 120
 Warn, 121
 Write, 121
 OutputFlexibleData
 Global Functions, 28
 Set Routines, 772
 OverwriteRA
 ElmSubstat, 254
 OvlAlleviate
 ComRel3, 446
 Pack
 BlkDef, 573
 PackAsMacro
 BlkDef, 573
 PackExternalReferences
 IntPrj, 659
 ParseDateLT
 Date/Time Functions, 47
 ParseDateUTC
 Date/Time Functions, 48
 PasteCopy
 General Object Methods, 160
 pi
 Mathematical Functions, 74
 PltDataseries, 691
 AddCurve, 691
 AddXYCurve, 692
 ClearCurves, 693
 GetDataSource, 693
 GetIntCalcres, 693
 PltLinebarplot, 693
 DoAutoScale, 693
 GetAxisX, 694
 GetAxisY, 694
 GetDataSeries, 694

GetLegend, 695
 GetTitleObject, 695
 SetAutoScaleModeX, 695
 SetAutoScaleModeY, 696
 SetAxisSharingLevelX, 696
 SetAxisSharingLevelY, 696
 SetScaleTypeX, 696
 SetScaleTypeY, 697
 SetScaleX, 697
 SetScaleY, 697
 PostCommand
 Global Functions, 28
 pow
 Mathematical Functions, 75
 PrepForUntouchedDelete
 Global Functions, 29
 PrintAllVal
 IntMon, 645
 PrintCheckResults
 ComLdf, 421
 PrintComparisonReport
 ComMerge, 428
 printf
 Output Window Functions, 119
 PrintModifications
 ComMerge, 428
 IntSstage, 677
 PrintVal
 IntMon, 645
 Purge
 IntPrj, 660
 IntUser, 682
 SetTboxconfig, 561
 PurgeUnusedObjects
 General Object Methods, 161
 QuickAccessAvailable
 ComUcteexp, 526
 Random
 Mathematical Functions, 69
 ReadValue
 IntSstage, 678
 Rebuild
 Global Functions, 29
 ReceiveData
 ComLink, 423
 ComOpc, 435
 Reconnect
 ElmGenstat, 200
 ElmPvsys, 218
 ElmSym, 260
 ElmXnet, 290
 ReductionInMemory
 ComRed, 444
 Refresh
 VisMagndiffplt, 729
 VisOcplot, 730
 Release
 ElmRes, 242
 IntComtrade, 604
 IntComtradeset, 613
 ReleaseMemory
 IntScenario, 670
 ReleaseResData
 Global Functions, 29
 ReloadProfile
 Global Functions, 29
 RelZpol, 698
 AssumeCompensationFactor, 698
 AssumeReRl, 698
 AssumeXeXI, 698
 Remove
 IntDplmap, 622
 IntDplvec, 625
 Set Routines, 772
 RemoveAllConfigurations
 SetDataext, 538
 RemoveAllRas
 ComSimoutage, 462
 RemoveBreaker
 StaCubic, 301
 RemoveCmdsForStudyCaseRow
 ComTasks, 511
 RemoveCommand
 ComTasks, 512
 RemoveConfiguration
 SetDataext, 538
 RemoveContingencies
 ComSimoutage, 462
 RemoveEvents
 ComContingency, 395
 ComOutage, 437
 ComRel3, 446
 RemoveOutages
 ComRel3, 446
 RemovePage
 GrpPage, 587
 SetDesktop, 542
 RemoveParameter
 CimModel, 577
 CimObject, 582
 IntAddonvars, 594
 RemoveProjectFromCombined
 IntPrj, 660
 RemoveRas
 ComSimoutage, 462
 RemoveRatio
 TypCtcore, 712
 RemoveRatioByIndex
 TypCtcore, 712
 RemoveRow

- ComTableReport, 492
- RemoveStudyCase
 - ComTasks, 512
- RemoveStudyCases
 - ComTasks, 513
- RemoveSwitchEvents
 - IntEvt, 627
- RemoveVar
 - IntMon, 646
- ReplaceNonAsciiCharacters
 - General Object Methods, 161
- Report
 - ComDllManager, 406
- ReportNonAsciiCharacters
 - General Object Methods, 162
- ReportUnusedObjects
 - General Object Methods, 162
- Reset
 - ComMerge, 428
 - ComSimoutage, 463
 - IntDocument, 616
 - SetLevelvis, 550
- ResetAll
 - IntOutage, 648
- ResetCalculation
 - Global Functions, 29
- ResetDerating
 - ElmGenstat, 201
 - ElmPvsys, 219
 - ElmSym, 260
- ResetQuickAccess
 - ComUcteexp, 527
- ResetRA
 - ElmSubstat, 254
- ResetReductionInMemory
 - ComRed, 444
- ResetStatusBit
 - StaExtbrkmea, 303
 - StaExtcmdmea, 308
 - StaExtdatmea, 313
 - StaExtfmea, 318
 - StaExtfuelmea, 323
 - StaExtmea, 328
 - StaExtppfmea, 333
 - StaExtppmea, 338
 - StaExtqmea, 343
 - StaExtstsmea, 347
 - StaExttapmea, 352
 - StaExtv3mea, 358
 - StaExtvmea, 363
- ResetStatusBitTmp
 - StaExtbrkmea, 303
 - StaExtcmdmea, 308
 - StaExtdatmea, 313
 - StaExtfmea, 318
 - StaExtfuelmea, 323
- StaExtmea, 328
- StaExtppfmea, 333
- StaExtppmea, 338
- StaExtqmea, 343
- StaExtstsmea, 347
- StaExttapmea, 353
- StaExtv3mea, 358
- StaExtvmea, 363
- ResetThirdPartyModule
 - BlkDef, 573
 - ComDpl, 411
 - TypQdsl, 714
- ResFirstValidObject
 - Global Functions, 30
- ResFirstValidObjectVar
 - Global Functions, 30
- ResFirstValidVar
 - Global Functions, 30
- ResGetMax
 - Global Functions, 30
- ResGetMin
 - Global Functions, 30
- ResIndex
 - Global Functions, 31
- Resize
 - ElmBoundary, 185
 - IntMat, 640
 - IntVec, 685
 - IntVecobj, 688
- ResNextValidObject
 - Global Functions, 31
- ResNextValidObjectVar
 - Global Functions, 31
- ResNextValidVar
 - Global Functions, 31
- ResNval
 - Global Functions, 31
- ResNvars
 - Global Functions, 31
- ResSortToVar
 - Global Functions, 32
- Restore
 - IntPrj, 660
- ResultForDirectionalBackupVariable
 - ComCoordreport, 399
- ResultForNonDirectionalBackupVariable
 - ComCoordreport, 399
- ResultForOverreachVariable
 - ComCoordreport, 400
- ResultForZoneVariable
 - ComCoordreport, 400
- RndExp
 - Mathematical Functions, 75
- RndGetMethod
 - Mathematical Functions, 75
- RndGetSeed

Mathematical Functions, 76
RndNormal
 Mathematical Functions, 76
RndSetup
 Mathematical Functions, 76
RndUnifInt
 Mathematical Functions, 78
RndUnifReal
 Mathematical Functions, 79
RndWeibull
 Mathematical Functions, 79
Rollback
 IntVersion, 690
round
 Mathematical Functions, 80
RowLbl
 IntMat, 638, 640
Save
 IntMat, 641
 IntScenario, 670
 IntVec, 686
 IntVecobj, 688
SaveAsRA
 ElmSubstat, 254
SaveAsScenario
 Global Functions, 32
SaveFile
 ElmFile, 196
SaveScenarioAs
 Global Functions, 32
SaveSimulationState
 ComSim, 457
SaveSnapshot
 ComSim, 457
ScnFreq, 698
 GetLimit, 699
 GetNumberOfViolations, 699
 GetValue, 699
 GetVariable, 699
 GetViolatedElement, 700
 GetViolationTime, 700
ScnFrt, 701
 GetLimit, 701
 GetNumberOfViolations, 701
 GetValue, 701
 GetVariable, 702
 GetViolatedElement, 702
 GetViolationTime, 702
ScnSpeed, 703
 GetLimit, 703
 GetNumberOfViolations, 703
 GetValue, 703
 GetVariable, 704
 GetViolatedElement, 704
 GetViolationTime, 704
ScnSync, 705
 GetLimit, 705
 GetNumberOfViolations, 705
 GetValue, 705
 GetVariable, 706
 GetViolatedElement, 706
 GetViolationTime, 706
ScnVar, 707
 GetLimit, 707
 GetNumberOfViolations, 707
 GetValue, 707
 GetVariable, 708
 GetViolatedElement, 708
 GetViolationTime, 708
ScnVolt, 709
 GetLimit, 709
 GetNumberOfViolations, 709
 GetValue, 709
 GetVariable, 710
 GetViolatedElement, 710
 GetViolationTime, 710
Search
 IntVecobj, 688
SearchObject
 General Object Methods, 162
SearchObjectByForeignKey
 Global Functions, 32
SearchScenario
 IntScensched, 672
SelectToolbox
 Global Functions, 33
SendData
 ComLink, 423
 ComOpc, 435
SentDataStatus
 ComLink, 423
Set Routines, 768
 Add, 768
 Clear, 769
 Count, 769
 First, 769
 FirstFilt, 770
 IsIn, 771
 MarkInGraphics, 771
 Next, 771
 NextFilt, 772
 Obj, 772
 OutputFlexibleData, 772
 Remove, 772
 ShowModalBrowser, 773
 ShowModalSelectBrowser, 773
 ShowModelessBrowser, 773
 SortToClass, 773
 SortToName, 774
 SortToVar, 774
Set

IntMat, 641
 IntVec, 686
 IntVecobj, 688
 SetActiveModule
 ComAddon, 380
 SetAdaptX
 SetDesktop, 542
 SetVipage, 567
 VisPath, 732
 VisPlot, 741
 VisPlot2, 754
 SetAdaptY
 IntPlot, 649
 VisPath, 733
 VisPlot, 742
 VisPlot2, 755
 SetAdditionalResultsFlagForCommand
 ComTasks, 513
 SetAsDefault
 ElmRes, 243
 SetAssociationValue
 CimModel, 578
 CimObject, 582, 583
 SetAttributeEnumeration
 CimModel, 578, 579
 CimObject, 583
 SetAttributeValue
 CimModel, 579
 CimObject, 584
 SetAuthorityUri
 ComGridtocim, 416
 SetAutoAssignmentForAll
 ComMerge, 428
 SetAutomaticCalculationResetEnabled
 Environment Functions, 59
 SetAutoScaleModeX
 GrpPage, 587
 PltLinebarplot, 695
 SetAutoScaleModeY
 GrpPage, 587
 PltLinebarplot, 696
 SetAutoScaleX
 SetDesktop, 543
 SetVipage, 568
 VisHrm, 725
 VisPlot, 742
 VisPlot2, 756
 SetAutoScaleY
 IntPlot, 650
 VisHrm, 726
 VisPlot, 743
 VisPlot2, 756
 SetAxisSharingLevelX
 PltLinebarplot, 696
 SetAxisSharingLevelY
 PltLinebarplot, 696
 SetBarLimits
 ComTablereport, 492
 SetBatchMode
 ComUcte, 520
 SetBoundaries
 ComGridtocim, 416
 SetCellAccess
 ComTablereport, 493
 SetCellEdit
 ComTablereport, 493
 SetCellValueToBar
 ComTablereport, 494
 SetCellValueToCheckbox
 ComTablereport, 495
 SetCellValueToDate
 ComTablereport, 496
 SetCellValue.ToDouble
 ComTablereport, 497
 SetCellValue.ToInt
 ComTablereport, 498
 SetCellValueToObject
 ComTablereport, 499
 SetCellValue.ToString
 ComTablereport, 500
 SetCluster, 528
 CalcCluster, 528
 GetNumberOfClusters, 529
 SetColouring
 SetColscheme, 531
 SetColscheme, 530
 CreateFilter, 530
 SetColouring, 531
 SetFilter, 536
 SetColumnHeader
 ComTablereport, 501
 SetColumnLabel
 IntMat, 641
 SetConsistencyCheck
 Environment Functions, 59
 SetCrvDesc
 VisHrm, 726
 VisPlot, 743
 VisPlot2, 757
 SetCrvDescX
 VisXyplot, 766
 SetCrvDescY
 VisXyplot, 767
 SetCurveValue
 ComTablereport, 501
 SetDataext, 536
 AddConfiguration, 536
 GetConfiguration, 537
 GetConfigurations, 537
 RemoveAllConfigurations, 538
 RemoveConfiguration, 538
 SetDefScaleX

VisHrm, [727](#)
 VisPlot, [744](#)
 VisPlot2, [758](#)
SetDefScaleY
 VisHrm, [727](#)
 VisPlot, [744](#)
 VisPlot2, [758](#)
SetDesktop, [539](#)
 AddPage, [539](#)
 Close, [540](#)
 DoAutoScaleX, [540](#)
 Freeze, [540](#)
 GetActivePage, [540](#)
 GetCanvasSize, [540](#)
 GetPage, [541](#)
 IsFrozen, [542](#)
 IsOpened, [542](#)
 RemovePage, [542](#)
 SetAdaptX, [542](#)
 SetAutoScaleX, [543](#)
 SetResults, [543](#)
 SetScaleX, [544](#)
 SetXVar, [545](#)
 Show, [545](#)
 Unfreeze, [546](#)
 WriteWMF, [546](#)
 ZoomAll, [547](#)
SetDetailed
 ElmLne, [211](#)
SetDialogSize
 ComTablereport, [502](#)
SetDiffMode
 Environment Functions, [60](#)
SetDisplayedButtons
 SetTboxconfig, [561](#)
SetDistrstate, [547](#)
 CalcCluster, [547](#)
SetElms
 StoMaint, [711](#)
SetEnableUserBreak
 Environment Functions, [63](#)
SetExportHtml
 ComTablereport, [502](#)
SetExportXls
 ComTablereport, [502](#)
SetExternalObject
 ComDpl, [411](#)
 ComPython, [442](#)
SetFileName
 ComSvgexport, [464](#)
 ComSvginport, [464](#)
SetFilt, [548](#)
 Get, [548](#)
SetFilter
 SetColscheme, [536](#)
SetFinalEchoOnEnabled
 Environment Functions, [61](#)
SetGeoCoordinateSystem
 IntPrj, [660](#)
SetGraphicUpdate
 Environment Functions, [61](#)
SetGridSelection
 ComUcteexp, [527](#)
SetGridsToExport
 ComGridtocim, [416](#)
SetGuiUpdateEnabled
 Environment Functions, [62](#)
SetIgnoreFlagForCommand
 ComTasks, [515](#)
SetIgnoreFlagForStudyCase
 ComTasks, [516](#)
SetImpedance
 ElmRelay, [221](#)
SetInputParameterDouble
 ComDpl, [411](#)
 ComPython, [443](#)
SetInputParameterInt
 ComDpl, [412](#)
 ComPython, [443](#)
SetInputParameterString
 ComDpl, [412](#)
 ComPython, [443](#)
SetLayerVisibility
 IntGrfnet, [632](#)
SetLayoutMode
 GrpPage, [588](#)
SetLevelvis, [548](#)
 AdaptWidth, [548](#)
 Align, [548](#)
 ChangeFont, [548](#)
 ChangeFrameAndWidth, [549](#)
 ChangeLayer, [549](#)
 ChangeRefPoints, [549](#)
 ChangeWidthVisibilityAndColour, [549](#)
 Mark, [550](#)
 Reset, [550](#)
SetLimits
 ComSimoutage, [463](#)
SetLineFeed
 Output Window Functions, [120](#)
SetListFilterSelection
 ComTablereport, [503](#)
SetLoadScaleAbsolute
 ElmZone, [293](#)
SetMaxI
 ElmRelay, [222](#)
SetMaxlearth
 ElmRelay, [223](#)
SetMeaValue

StaExtbrkmea, 303, 304
 StaExtcmdmea, 308
 StaExtdatmea, 313
 StaExtfmea, 318
 StaExtfuelmea, 323
 StaExttimea, 328
 StaExtpfmea, 333
 StaExtpmea, 338
 StaExtqmea, 343
 StaExttsmea, 348
 StaExttapmea, 353
 StaExtv3mea, 358
 StaExtvmea, 363
 SetMicroSCADASatus
 ComLink, 424
 SetMinI
 ElmRelay, 223
 SetMinlearth
 ElmRelay, 223
 SetNumberFormatForPlot
 ComTablereport, 503
 SetNumSlave
 SetParalman, 551
 SetObj
 ElmRes, 243
 SetObject
 ComSvgexport, 464
 ComSvgimport, 464
 SetObjects
 ComSvgexport, 464
 SetObjectsToCompare
 ComMerge, 429
 SetObjs
 ComOutage, 438
 SetOldDistributeLoadMode
 ComLdf, 421
 SetOPCReceiveQuality
 ComLink, 424
 SetOperationValue
 IntScenario, 671
 SetOutOfService
 ElmRelay, 223
 SetOutputWindowState
 Output Window Functions, 120
 SetParalman, 550
 GetNumSlave, 550
 SetNumSlave, 551
 SetTransfType, 551
 SetPassword
 IntUser, 682
 SetPath, 551
 AllBreakers, 551
 AllClosedBreakers, 552
 AllOpenBreakers, 552
 AllProtectionDevices, 552
 Create, 552
 GetAll, 553
 GetBranches, 553
 GetBuses, 554
 GetPathFolder, 554
 SetPrimaryTap
 StaCt, 294
 SetProgressBarUpdatesEnabled
 Environment Functions, 62
 SetRA
 ElmSubstat, 255
 SetRandomSeed
 Mathematical Functions, 80
 SetRandSeed
 Mathematical Functions, 80
 SetRecurrence
 IntPlannedout, 649
 SetRefresh
 ComTablereport, 504
 SetRescheduleFlag
 Environment Functions, 62
 SetResults
 GrpPage, 588
 SetDesktop, 543
 SetVipage, 568
 SetResultsFolder
 ComTasks, 517
 SetResultString
 ComDpl, 413
 SetRowLabel
 IntMat, 642
 SetScaleTypeX
 GrpPage, 588
 PltLinebarplot, 696
 SetScaleTypeY
 GrpPage, 588
 PltLinebarplot, 697
 SetScaleX
 GrpPage, 589
 PltLinebarplot, 697
 SetDesktop, 544
 SetVipage, 569
 VisPath, 733
 VisPlot, 745
 VisPlot2, 759
 SetScaleY
 GrpPage, 589
 IntPlot, 651
 PltLinebarplot, 697
 VisBdia, 717
 VisPath, 734
 VisPlot, 746
 VisPlot2, 760
 SetSelect, 554
 AddRef, 554
 All, 555
 AllAsm, 555

AllBars, 556
 AllBreakers, 556
 AllClosedBreakers, 556
 AllElm, 557
 AllLines, 557
 AllLoads, 558
 AllOpenBreakers, 558
 AllSym, 558
 AllTypLne, 559
 Clear, 559
 GetAll, 560
 SetShowAllUsers
 Global Functions, 33
 SetSize
 General Object Methods, 163
 SetSorting
 ComTablereport, 504
 SetStatus
 StaExtbrkmea, 304
 StaExtcmdmea, 309
 StaExtdatmea, 314
 StaExtfmea, 319
 StaExtfuelmea, 323
 StaExttimea, 328
 StaExtpfmea, 333
 StaExtpmea, 338
 StaExtqmea, 343
 StaExtsmea, 348
 StaExttapmea, 353
 StaExtv3mea, 358
 StaExtvmea, 363
 SetStatusBit
 StaExtbrkmea, 305
 StaExtcmdmea, 309
 StaExtdatmea, 314
 StaExtfmea, 319
 StaExtfuelmea, 324
 StaExttimea, 329
 StaExtpfmea, 334
 StaExtpmea, 339
 StaExtqmea, 344
 StaExtsmea, 349
 StaExttapmea, 354
 StaExtv3mea, 359
 StaExtvmea, 364
 SetStatusBitTmp
 StaExtbrkmea, 305
 StaExtcmdmea, 310
 StaExtdatmea, 315
 StaExtfmea, 320
 StaExtfuelmea, 324
 StaExttimea, 329
 StaExtpfmea, 334
 StaExtpmea, 339
 StaExtqmea, 344
 StaExtsmea, 349
 StaExttapmea, 354
 StaExtv3mea, 359
 StaExtvmea, 364
 SetStatusText
 ComTablereport, 504
 SetStatusTmp
 StaExtbrkmea, 305
 StaExtcmdmea, 310
 StaExtdatmea, 315
 StaExtfmea, 320
 StaExtfuelmea, 325
 StaExttimea, 330
 StaExtpfmea, 335
 StaExtpmea, 340
 StaExtqmea, 345
 StaExtsmea, 349
 StaExttapmea, 354
 StaExtv3mea, 360
 StaExtvmea, 365
 SetStudyTime
 IntCase, 595
 SetStyle
 SetVipage, 570
 SetSubElmResKey
 ElmRes, 244
 SetSwitchShcEventMode
 ComLink, 424
 SetSymbolComponentVisibility
 IntGrfnet, 632
 SetTboxconfig, 560
 Check, 560
 GetAvailableButtons, 560
 GetDisplayedButtons, 561
 Purge, 561
 SetDisplayedButtons, 561
 SetTextAxisDistForPlot
 ComTablereport, 504
 SetThirdPartyModule
 BlkDef, 574
 ComDpl, 413
 TypQdsl, 714
 SetTicksForPlot
 ComTablereport, 505
 SetTile
 SetVipage, 570
 SetTime, 561
 Date, 562
 ElmRelay, 224
 SetTime, 562
 SetTimeUTC, 562
 Time, 563
 SetTimeUTC
 SetTime, 562
 Settings Methods, 528
 SetTitle
 ComTablereport, 505

SetTransfType
 SetParalman, 551
SetUser, 563
 GetNumProcesses, 563
ShowModalSelectFileDialog
 Dialogue Boxes Functions, 56
ShowModalSelectTree
 General Object Methods, 165
ShowModelessBrowser
 Dialogue Boxes Functions, 57
 Set Routines, 773
ShowY2
 VisPlot2, 762
sin
 Mathematical Functions, 80
sinh
 Mathematical Functions, 80
Size
 IntDplmap, 622
 IntDplvec, 625
 IntVec, 686
 IntVecobj, 689
SizeX
 ComTablereport, 486
 ElmRes, 236
 IntComtrade, 600
 IntComtradeset, 609
 IntMat, 637
SizeY
 ComTablereport, 486
 ElmRes, 236
 IntComtrade, 600
 IntComtradeset, 609
 IntMat, 637
Sleep
 Global Functions, 34
slotupd
 ElmComp, 187
 ElmRelay, 224
SlotUpdate
 ElmComp, 187
 ElmRelay, 224
snm
 General Object Methods, 166
Sort
 IntDplvec, 626
 IntVec, 686
SortAccordingToColumn
 ElmRes, 244
 IntComtrade, 604
 IntComtradeset, 613
SortToClass
 Set Routines, 773
SortToColum
 IntMat, 642
SortToColumn
 IntMat, 642
SortToName
 Set Routines, 774
SortToVar

Set Routines, 774
SplitLine
 Global Functions, 34
sprintf
 String Functions, 124
sqr
 Mathematical Functions, 81
sqrt
 Mathematical Functions, 81
sscanf
 String Functions, 125
sscanfsep
 String Functions, 125
StaCt, 293
 SetPrimaryTap, 294
StaCubic, 294
 AddBreaker, 294
 GetAll, 295
 GetBranch, 296
 GetConnectedMajorNodes, 296
 GetConnections, 297
 GetNearestBusbars, 297
 GetPathToNearestBusbar, 298
 GetRemoteBorderCubicles, 299
 GetSwitch, 300
 IsClosed, 300
 IsConnected, 300
 RemoveBreaker, 301
StaExtbrkmea, 301
 CopyExtMeaStatusToStatusTmp, 301
 GetMeaValue, 302
 GetStatus, 302
 GetStatusTmp, 302
 InitTmp, 302
 IsStatusBitSet, 302
 IsStatusBitSetTmp, 303
 ResetStatusBit, 303
 ResetStatusBitTmp, 303
 SetMeaValue, 303, 304
 SetStatus, 304
 SetStatusBit, 305
 SetStatusBitTmp, 305
 SetStatusTmp, 305
 UpdateControl, 305
 UpdateCtrl, 306
StaExtcmdmea, 306
 CopyExtMeaStatusToStatusTmp, 306
 GetMeaValue, 307
 GetStatus, 307
 GetStatusTmp, 307
 InitTmp, 307
 IsStatusBitSet, 307
 IsStatusBitSetTmp, 308
 ResetStatusBit, 308
 ResetStatusBitTmp, 308
 SetMeaValue, 308
StaExtdatmea, 311
 CopyExtMeaStatusToStatusTmp, 311
 CreateEvent, 311
 GetMeaValue, 312
 GetStatus, 312
 GetStatusTmp, 312
 InitTmp, 312
 IsStatusBitSet, 313
 IsStatusBitSetTmp, 313
 ResetStatusBit, 313
 ResetStatusBitTmp, 313
 SetMeaValue, 313
 SetStatus, 314
 SetStatusBit, 314
 SetStatusBitTmp, 315
 SetStatusTmp, 315
 UpdateControl, 315
 UpdateCtrl, 316
StaExtfmea, 316
 CopyExtMeaStatusToStatusTmp, 316
 GetMeaValue, 316
 GetStatus, 317
 GetStatusTmp, 317
 InitTmp, 317
 IsStatusBitSet, 317
 IsStatusBitSetTmp, 318
 ResetStatusBit, 318
 ResetStatusBitTmp, 318
 SetMeaValue, 318
 SetStatus, 319
 SetStatusBit, 319
 SetStatusBitTmp, 320
 SetStatusTmp, 320
 UpdateControl, 320
 UpdateCtrl, 320
StaExtfuelmea, 321
 CopyExtMeaStatusToStatusTmp, 321
 GetMeaValue, 321
 GetStatus, 322
 GetStatusTmp, 322
 InitTmp, 322
 IsStatusBitSet, 322
 IsStatusBitSetTmp, 322
 ResetStatusBit, 323
 ResetStatusBitTmp, 323
 SetMeaValue, 323
 SetStatus, 323
 SetStatusBit, 324
 SetStatusBitTmp, 324
 SetStatusTmp, 325

- UpdateControl, 325
 - UpdateCtrl, 325
 - StaExtmea, 326
 - CopyExtMeaStatusToStatusTmp, 326
 - GetMeaValue, 326
 - GetStatus, 327
 - GetStatusTmp, 327
 - InitTmp, 327
 - IsStatusBitSet, 327
 - IsStatusBitSetTmp, 327
 - ResetStatusBit, 328
 - ResetStatusBitTmp, 328
 - SetMeaValue, 328
 - SetStatus, 328
 - SetStatusBit, 329
 - SetStatusBitTmp, 329
 - SetStatusTmp, 330
 - UpdateControl, 330
 - UpdateCtrl, 330
 - StaExtpfmea, 331
 - CopyExtMeaStatusToStatusTmp, 331
 - GetMeaValue, 331
 - GetStatus, 331
 - GetStatusTmp, 332
 - InitTmp, 332
 - IsStatusBitSet, 332
 - IsStatusBitSetTmp, 332
 - ResetStatusBit, 333
 - ResetStatusBitTmp, 333
 - SetMeaValue, 333
 - SetStatus, 333
 - SetStatusBit, 334
 - SetStatusBitTmp, 334
 - SetStatusTmp, 335
 - UpdateControl, 335
 - UpdateCtrl, 335
 - StaExtpmea, 336
 - CopyExtMeaStatusToStatusTmp, 336
 - GetMeaValue, 336
 - GetStatus, 336
 - GetStatusTmp, 337
 - InitTmp, 337
 - IsStatusBitSet, 337
 - IsStatusBitSetTmp, 337
 - ResetStatusBit, 338
 - ResetStatusBitTmp, 338
 - SetMeaValue, 338
 - SetStatus, 338
 - SetStatusBit, 339
 - SetStatusBitTmp, 339
 - SetStatusTmp, 340
 - UpdateControl, 340
 - UpdateCtrl, 340
 - StaExtqmea, 341
 - CopyExtMeaStatusToStatusTmp, 341
 - GetMeaValue, 341
 - GetStatus, 341
 - GetStatusTmp, 342
 - InitTmp, 342
 - IsStatusBitSet, 342
 - IsStatusBitSetTmp, 342
 - ResetStatusBit, 343
 - ResetStatusBitTmp, 343
 - SetMeaValue, 343
 - SetStatus, 343
 - SetStatusBit, 344
 - SetStatusBitTmp, 344
 - SetStatusTmp, 345
 - UpdateControl, 345
 - UpdateCtrl, 345
- StaExtsmea, 346
 - CopyExtMeaStatusToStatusTmp, 346
 - GetStatus, 346
 - GetStatusTmp, 346
 - InitTmp, 346
 - IsStatusBitSet, 347
 - IsStatusBitSetTmp, 347
 - ResetStatusBit, 347
 - ResetStatusBitTmp, 347
 - SetMeaValue, 348
 - SetStatus, 348
 - SetStatusBit, 349
 - SetStatusBitTmp, 349
 - SetStatusTmp, 349
 - UpdateControl, 350
 - UpdateCtrl, 350
- StaExttapmea, 350
 - CopyExtMeaStatusToStatusTmp, 351
 - GetMeaValue, 351
 - GetStatus, 351
 - GetStatusTmp, 352
 - InitTmp, 352
 - IsStatusBitSet, 352
 - IsStatusBitSetTmp, 352
 - ResetStatusBit, 352
 - ResetStatusBitTmp, 353
 - SetMeaValue, 353
 - SetStatus, 353
 - SetStatusBit, 354
 - SetStatusBitTmp, 354
 - SetStatusTmp, 354
 - UpdateControl, 355
 - UpdateCtrl, 355
- StaExtv3mea, 356
 - CopyExtMeaStatusToStatusTmp, 356
 - GetMeaValue, 356
 - GetStatus, 356
 - GetStatusTmp, 357
 - InitTmp, 357
 - IsStatusBitSet, 357
 - IsStatusBitSetTmp, 357
 - ResetStatusBit, 358

- ResetStatusBitTmp, 358
SetMeaValue, 358
SetStatus, 358
SetStatusBit, 359
SetStatusBitTmp, 359
SetStatusTmp, 360
UpdateControl, 360
UpdateCtrl, 360
StaExtvmea, 361
CopyExtMeaStatusToStatusTmp, 361
GetMeaValue, 361
GetStatus, 361
GetStatusTmp, 362
InitTmp, 362
IsStatusBitSet, 362
IsStatusBitSetTmp, 362
ResetStatusBit, 363
ResetStatusBitTmp, 363
SetMeaValue, 363
SetStatus, 363
SetStatusBit, 364
SetStatusBitTmp, 364
SetStatusTmp, 365
UpdateControl, 365
UpdateCtrl, 365
Start
ComDiff, 406
StartTrace
ComContingency, 396
ComOutage, 438
StaSwitch, 366
Close, 366
IsClosed, 366
IsOpen, 367
Open, 367
StatFileGetXrange
Global Functions, 34
StatFileResetXrange
Global Functions, 35
StatFileSetXrange
Global Functions, 35
Station Elements Methods, 293
StoMaint, 711
SetElms, 711
Stop
ComDiff, 406
StopTrace
ComContingency, 396
ComOutage, 438
strchg
String Functions, 126
strcmp
String Functions, 127
strcpy
String Functions, 128
strftime
Date/Time Functions, 49
String Functions, 122
sprint, 124
sscanf, 125
sscanfsep, 125
strchg, 126
strcmp, 127
strcpy, 128
strlen, 128
strstr, 129
strtok, 129
tochar, 130
tocharcodepoint, 130
tolower, 131
ToStr, 124
toupper, 131
strlen
String Functions, 128
strstr
String Functions, 129
strtok
String Functions, 129
SubscribeProjectReadOnly
IntPrj, 661
SubscribeProjectReadWrite
IntPrj, 661
SummaryGrid
Global Functions, 22
SwitchOff
General Object Methods, 166
SwitchOn
General Object Methods, 167
tan
Mathematical Functions, 81
tanh
Mathematical Functions, 81
TerminateSession
IntUser, 682
Time
SetTime, 563
time
Mathematical Functions, 82
tochar
String Functions, 130
tocharcodepoint
String Functions, 130
tolower
String Functions, 131
tombchar
Multibyte Encoded String Functions, 116
tombcharcodepoint
Multibyte Encoded String Functions, 116
tombslower
Multibyte Encoded String Functions, 116
tombsupper

Multibyte Encoded String Functions, 117
TopologyForDirectionalBackupVariable
 ComCoordreport, 401
TopologyForNonDirectionalBackupVariable
 ComCoordreport, 402
TopologyForOverreachVariable
 ComCoordreport, 402
TopologyForZoneVariable
 ComCoordreport, 402
ToStr
 String Functions, 124
TotalLossCost
 ComCapo, 384
toupper
 String Functions, 131
TransferDirectionalBackupResultsTo
 ComCoordreport, 403
TransferNonDirectionalBackupResultsTo
 ComCoordreport, 403
TransferOverreachResultsTo
 ComCoordreport, 404
TransferResultsTo
 ComCoordreport, 405
TransferZoneResultsTo
 ComCoordreport, 405
TransformGeoCoordinates
 IntPrj, 661
trunc
 Mathematical Functions, 82
twopi
 Mathematical Functions, 82
TypAsmo, 711
 CalcElParams, 711
TypCtcore, 712
 AddRatio, 712
 RemoveRatio, 712
 RemoveRatioByIndex, 712
TypLne, 713
 IsCable, 713
TypQdsl, 713
 Encrypt, 713
 IsEncrypted, 714
 ResetThirdPartyModule, 714
 SetThirdPartyModule, 714
TypTr2, 715
 GetZeroSequenceHVLVT, 715
Unfreeze
 SetDesktop, 546
unm
 General Object Methods, 167
UnsubscribeProject
 IntPrj, 662
Update
 ChaVecfile, 575
 ComSimoutage, 463
ElmBmu, 182
ElmBoundary, 185
ElmBranch, 185
ElmCabsys, 187
IntDplmap, 622
IntScheduler, 675
UpdateControl
 StaExtbrkmea, 305
 StaExtcmdmea, 310
 StaExtdatmea, 315
 StaExtfmea, 320
 StaExtfuelmea, 325
 StaExtmea, 330
 StaExtpfmea, 335
 StaExtpmea, 340
 StaExtqmea, 345
 StaExtsmea, 350
 StaExttapmea, 355
 StaExtv3mea, 360
 StaExtvmea, 365
UpdateCtrl
 StaExtbrkmea, 306
 StaExtcmdmea, 311
 StaExtdatmea, 316
 StaExtfmea, 320
 StaExtfuelmea, 325
 StaExtmea, 330
 StaExtpfmea, 335
 StaExtpmea, 340
 StaExtqmea, 345
 StaExtsmea, 350
 StaExttapmea, 355
 StaExtv3mea, 360
 StaExtvmea, 365
UpdateFromCurrentView
 IntViewbookmark, 691
UpdateGroups
 IntUserman, 684
UpdateStatistics
 IntPrj, 662
UpdateSubstationTerminals
 ElmTerm, 268
UpdateTableReports
 Dialogue Boxes Functions, 57
UpdateTablesByCalcPeriod
 ComTececo, 518
UpdateToDefaultStructure
 IntPrj, 662
UpdateToMostRecentBaseVersion
 IntPrj, 662
ValidateConstraints
 ComRel3, 447
validLDF
 Global Functions, 24
validRMS

Global Functions, 25
validSHC
 Global Functions, 26
validSIM
 Global Functions, 26
VarExists
 General Object Methods, 168
View
 IntDocument, 616
 IntUrl, 681
VisBdia, 716
 AddObjs, 716
 AddResObjs, 716
 Clear, 717
 SetScaleY, 717
 SetXVariable, 718
 SetYVariable, 719
VisDraw, 719
 AddRelay, 720
 AddRelays, 720
 CentreOrigin, 720
 Clear, 720
 DoAutoSizeOnAll, 720
 DoAutoSizeOnCharacteristics, 721
 DoAutoSizeOnImpedances, 721
 DoAutoSizeX, 721
 DoAutoSizeY, 721
VisHrm, 722
 Clear, 722
 DoAutoSizeX, 722
 DoAutoSizeY, 723
 GetDataSource, 723
 GetScaleObjX, 724
 GetScaleObjY, 724
 SetAutoSizeX, 725
 SetAutoSizeY, 726
 SetCrvDesc, 726
 SetDefScaleX, 727
 SetDefScaleY, 727
VisMagndiffplt, 728
 AddRelay, 728
 AddRelays, 728
 Clear, 728
 DoAutoSizeX, 728
 DoAutoSizeY, 729
 Refresh, 729
VisOcplot, 729
 AddRelay, 729
 AddRelays, 730
 Clear, 730
 DoAutoSizeX, 730
 DoAutoSizeY, 730
 Refresh, 730
VisPath, 731
 Clear, 731
 DoAutoSizeX, 731
 DoAutoSizeY, 732
 SetAdaptX, 732
 SetAdaptY, 733
 SetScaleX, 733
 SetScaleY, 734
VisPcompdifffplt, 735
 AddRelay, 735
 AddRelays, 735
 CentreOrigin, 735
 Clear, 735
 DoAutoSizeX, 736
 DoAutoSizeY, 736
VisPlot2, 748
 AddResVars, 749
 AddVars, 750
 Clear, 750
 DoAutoSizeX, 751
 DoAutoSizeY, 751
 DoAutoSizeY2, 752
 GetDataSource, 752
 GetScaleObjX, 753
 GetScaleObjY, 754
 SetAdaptX, 754
 SetAdaptY, 755
 SetAutoSizeX, 756
 SetAutoSizeY, 756
 SetCrvDesc, 757
 SetDefScaleX, 758
 SetDefScaleY, 758
 SetScaleX, 759
 SetScaleY, 760
 SetXVar, 761
 ShowY2, 762
VisPlot, 736
 AddResVars, 736
 AddVars, 737
 Clear, 738
 DoAutoSizeX, 738
 DoAutoSizeY, 739
 GetDataSource, 739
 GetIntCalcres, 740
 GetScaleObjX, 740
 GetScaleObjY, 740
 SetAdaptX, 741
 SetAdaptY, 742
 SetAutoSizeX, 742
 SetAutoSizeY, 743
 SetCrvDesc, 743
 SetDefScaleX, 744
 SetDefScaleY, 744
 SetScaleX, 745
 SetScaleY, 746
 SetXVar, 747
VisPlottz, 762
 AddRelay, 762
 AddRelays, 763

Clear, 763
 DoAutoSizeX, 763
 DoAutoSizeY, 763
 VisVec, 764
 CentreOrigin, 764
 VisVecres, 764
 CentreOrigin, 764
 VisXyplot, 764
 Clear, 764
 DoAutoSizeX, 765
 DoAutoSizeY, 765
 GetDataSource, 766
 SetCrvDescX, 766
 SetCrvDescY, 767
 Warn
 Output Window Functions, 121
 WereModificationsFound
 ComMerge, 429
 Write
 ElmRes, 245
 Output Window Functions, 121
 WriteDraw
 ElmRes, 245
 WriteValue
 IntStage, 678
 WriteWMF
 SetDesktop, 546

 xlActivateWorksheet
 MS Excel Functions, 92
 xlAddWorksheet
 MS Excel Functions, 92
 xlCloseWorkbook
 MS Excel Functions, 92
 xlDeleteWorksheet
 MS Excel Functions, 93
 xlGetActiveWorksheetIndex
 MS Excel Functions, 93
 xlGetDateSeparator
 MS Excel Functions, 94
 xlGetDecimalSeparator
 MS Excel Functions, 94
 xlGetThousandsSeparator
 MS Excel Functions, 94
 xlGetValue
 MS Excel Functions, 94
 xlGetWorksheetCount
 MS Excel Functions, 95
 xlGetWorksheetName
 MS Excel Functions, 95
 xlNewWorkbook
 MS Excel Functions, 96
 xlOpenWorkbook
 MS Excel Functions, 96
 xlResetTextStyle

 MS Excel Functions, 96
 xlRunMacro
 MS Excel Functions, 97
 xlSaveWorkbook
 MS Excel Functions, 98
 xlSaveWorkbookAs
 MS Excel Functions, 98
 xlSetBorder
 MS Excel Functions, 99
 xlSetColumnWidth
 MS Excel Functions, 100
 xlSetDebug
 MS Excel Functions, 100
 xlSetFillColor
 MS Excel Functions, 101
 xlSetFontName
 MS Excel Functions, 102
 xlFontSize
 MS Excel Functions, 102
 xlSetHorizontalAlignment
 MS Excel Functions, 103
 xlSetNumberFormat
 MS Excel Functions, 104
 xlSetPrintTitleRows
 MS Excel Functions, 104
 xlSetRowHeight
 MS Excel Functions, 105
 xlSetTextColor
 MS Excel Functions, 105
 xlSetTextStyle
 MS Excel Functions, 106
 xlSetValue
 MS Excel Functions, 107
 xlSetValues
 MS Excel Functions, 107
 xlSetVerticalAlignment
 MS Excel Functions, 108
 xlSetVisible
 MS Excel Functions, 109
 xlSetWorksheetName
 MS Excel Functions, 109
 xlSetWrapText
 MS Excel Functions, 110
 xlStart
 MS Excel Functions, 110
 xlTerminate
 MS Excel Functions, 113

 ZeroDerivative
 ComInc, 419
 ZoomAll
 SetDesktop, 547