

# Auxiliary File Format for Simulator 17

Last Updated: December 10, 2013



**PowerWorld**  
Corporation

PowerWorld Corporation

2001 South First St

Champaign, IL 61820

(217) 384-6330

<http://www.powerworld.com>

[info@powerworld.com](mailto:info@powerworld.com)

# Table of Contents

<b>Introduction</b> .....	<b>1</b>
<b>SCRIPT Section</b> .....	<b>2</b>
<b>Using Filters in Script Commands</b> .....	<b>2</b>
<b>General Actions</b> .....	<b>3</b>
Generic Data Actions .....	3
CopyFile("oldfilename", "newfilename");.....	3
CreateData(objecttype, [fieldlist], [valuelist]); .....	3
Delete(objecttype, filter);.....	3
DeleteFile("filename");.....	3
DeleteIncludingContents(objecttype, filter);.....	4
EnterDistMasterPassword(Password);.....	4
ImportData("filename", FileType, CreateIfNotFound);.....	4
LoadAux("filename", CreateIfNotFound); .....	4
LoadData("filename", DataName, CreateIfNotFound);.....	4
LoadScript("filename", ScriptName, CreateIfNotFound);.....	4
RenameFile("oldfilename", "newfilename"); .....	4
SaveData("filename", filetype, objecttype, [fieldlist], [subdatalist], filter, [SortFieldList]);.....	5
SaveDataUsingExportFormat("filename", filetype, "FormatName", ModelToUse); .....	5
SaveDataWithExtra("filename", filetype, objecttype, [fieldlist], [subdatalist], filter, [SortFieldList], [Header_List], [Header_Value_List]); .....	6
SaveObjectFields("filename", objecttype, [fieldlist]); .....	6
SelectAll(objecttype, filter);.....	6
SendtoExcel(objecttype, [fieldlist], filter, UseColumnHeaders, "workbookname", "worksheetname");.....	7
SetCurrentDirectory("filedirectory", CreateIfNotFound); .....	7
SetData(objecttype, [fieldlist], [valuelist], filter);.....	7
StopAuxFile;.....	7
UnSelectAll(objecttype, filter);.....	7
WriteTextToFile("filename", "text..."); .....	8
Case Related Actions .....	9
AutoInsertTieLineTransactions; .....	10
CalculateRXBGFromLengthConfigCondType(filter); .....	10
CaseDescriptionClear; .....	10
CaseDescriptionSet("text...", Append);.....	10
ChangeSystemMVABase(NewBase);.....	10
ConditionVoltagesOnRadialBuses(VoltageThreshold, AngleThreshold, filter);.....	10
DetermineBranchesThatCreateIslands(Filter, StoreBuses, "filename", SetSelectedOnLines, FileType); .....	11
DeterminePathDistance([start], BranchDistMeas, BranchFilter, BusField); .....	11
DetermineShortestPath([start], [end], BranchDistanceMeasure, BranchFilter, Filename);.....	12
DirectionsAutoInsert(Source, Sink, DeleteExisting, UseAreaZoneFilters, Start, Increment);.....	12
EnterMode(mode or submode); .....	12
ExitProgram;.....	12
GenForceLDC_RCC(filter); .....	12
InjectionGroupsAutoInsert; .....	13
LogAdd("string..."); .....	13
LogAddDateTime("label", includedate, includetime, includemilliseconds);.....	13
LogClear; .....	13
LogSave("filename", AppendFile);.....	13
NewCase;.....	13
OpenCase("filename", OpenFileType,[LoadTransactions,StarBus,MSLine,VarLimDead,PostCTGAGC]);.....	13
SaveCase("filename", SaveFileType, [PostCTGAGC, UseAreaZone]); .....	14
SaveJacobian("JacFileName", "JIDFileName", FileType, JacForm).....	14
SaveYbusInMatlabFormat("filename", IncludeVoltages); .....	14
Scale(scaletype, basedon, [parameters], scalemarker);.....	14
SetGenPMaxFromReactiveCapabilityCurve(filter); .....	15
SetParticipationFactors(Method, ConstantValue, Object);.....	15
SetScheduledVoltageForABus([bus identifier], voltage);.....	15

Online Actions .....	16
CloseOnline("OnlineName");.....	16
ExportOnlineAsShapeFile("filename", "OnlineName", "ShapeFileExportDescriptionName", UseLonLat, PointLocation) ..	16
LoadAXD("filename", "OnlineName", CreateIfNotFound) .....	16
OpenOnline("filename", "view", FullScreen, ShowFull, LinkMethod);.....	16
SaveOnline("filename", "OnlineName", SaveFileType);.....	17
<b>Edit Mode Actions.....</b>	<b>18</b>
Case Related Actions .....	18
AppendCase("filename", OpenFileType);.....	18
AppendCase("filename", OpenFileType, [StarBus]);.....	18
AppendCase("filename", OpenFileType, [MSLine, VarLimDead, PostCTGAGC]);.....	19
Combine([elementA], [elementB]);.....	19
DeleteExternalSystem;.....	19
Equivalence;.....	19
InterfacesAutoInsert(Type, DeleteExisting, UseFilters, "Prefix", Limits);.....	19
MergeBuses([element], Filter);.....	19
MergeLineTerminals(Filter); .....	20
Move([elementA], [destination parameters], HowMuch);.....	20
Renumber3WXFormerStarBuses("filename");.....	20
RenumberMSLineDummyBuses("filename"); .....	21
SaveExternalSystem("Filename", SaveFileType, WithTies);.....	21
SplitBus([element], NewBusNumber, InsertBusTieLine, LineOpen); .....	21
TapTransmissionLine([element], PosAlongLine, NewBusNumber, ShuntModel, TreatAsMSLine); .....	21
<b>Run Mode Actions.....</b>	<b>22</b>
Animate(DoAnimate); .....	22
CalculatePTDF([transactor seller], [transactor buyer], LinearMethod);.....	23
CalculatePTDFMultipleDirections(StoreForBranches, StoreForInterfaces, LinearMethod);.....	23
CalculateLODF([BRANCH nearbusnum farbusnum ckt], LinearMethod); .....	23
CalculateLODFMatrix(WhichOnes, filterProcess, filterMonitor, MonitorOnlyClosed, LinearMethod);.....	24
CalculateTLR([flow element], direction, [transactor], LinearMethod);.....	24
CalculateTLRMultipleElement(TypeElement, WhichElement, direction, [transactor], LinearMethod);.....	25
CalculateVoltSense([BUS num]);.....	25
CalculateFlowSense([flow element], FlowType); .....	25
CalculateLossSense(FunctionType);.....	25
CalculateVoltToTransferSense([transactor seller], [transactor buyer], TransferType, TurnOffAVR); .....	26
CalculateVoltSelfSense(filter); .....	26
SetSensitivitiesAtOutOfServiceToClosest;.....	26
ZeroOutMismatches;.....	26
Power Flow Related Actions.....	27
DiffFlowClearBase; .....	27
DiffFlowKeyType(KeyType); .....	27
DiffFlowMode(diffmode);.....	27
DiffFlowSetAsBase; .....	27
DiffFlowRefresh;.....	27
DoCTGAction([contingency action]); .....	27
ResetToFlatStart (FlatVoltagesAngles, ShuntsToMax, LTCsToMiddle, PSAnglesToMiddle); .....	28
SolvePowerFlow (SolMethod, "filename1", "filename2", CreateIfNotFound1, CreateIfNotFound2); .....	28
Contingency Related Actions.....	29
CTGSolveAll(DoDistributed);.....	29
CTGSolve("ContingencyName");.....	29
CTGSetAsReference;.....	29
CTGRestoreReference; .....	29
CTGWriteResultsAndOptions("filename", [opt1, opt2, opt3, ..., opt9], KeyField, UseDATASection);.....	29
CTGProduceReport("filename");.....	30
CTGAutoInsert; .....	30
CTGConvertAllToDeviceCTG(KeepOriginalIfEmpty); .....	30
CTGCalculateOTDF([transactor seller], [transactor buyer], LinearMethod); .....	30
CTGWriteFilePTI("filename", BusFormat, TruncateCTGLabels);.....	30
CTGJoinActiveCTGs(InsertSolvePowerFlow, DeleteExisting, JoinWithSelf, "filename");.....	30
Fault Related Actions.....	31

Fault([Bus num, faulttype, R, X]);	31
Fault([BRANCH nearbusnum farbusnum ckt], faultlocation, faulttype, R, X]);	31
ATC (Available Transfer Capability) Related Actions	32
ATCDetermine([transactor seller], [transactor buyer], ApplyTransfer, DoDistributed);	32
ATCRestoreInitialState;	32
ATCIncreaseTransferBy(amount);	32
ATCTakeMeToScenario(RL, G, I);	32
ATCDetermineATCFor(RL, G, I, ApplyTransfer);	32
ATCWriteResultsAndOptions("filename", AppendFile);	32
ATCWriteToExcel("worksheetname");	32
ATCWriteToText("filename", filetype);	33
ATCSetAsReference;	33
GIC (Geomagnetically Induced Current) Related Actions	34
GICCalculate(MaxField, Direction, SolvePF);	34
GICClear;	34
GICTimeVaryingCalculate(TheTime,SolvePF);	34
GICTimeVaryingAddNewTime(NewTime);	34
GICTimeVaryingDeleteAllTimes;	34
GICWriteOptions("FileName", KeyField);	34
ITP (Integred Topology Processing) Related Actions	35
CloseWithBreakers(objecttype, "filtername" or [object identifier], OnlyEnergizeSpecifiedObjects);	35
SaveConsolidatedCase("filename", filetype, [BusFormat, TruncateCtgLabels, AddCommentsForObjectLabels]);	36
OPF (Optimal Power Flow) and SCOPF Related Actions	37
SolvePrimalLP("filename1", "filename2", CreateIfNotFound1, CreateIfNotFound2);	37
InitializeLP("filename1", "filename2", CreateIfNotFound1, CreateIfNotFound2);	37
SolveSinglePrimalLPOuterLoop("filename1", "filename2", CreateIfNotFound1, CreateIfNotFound2);	37
SolveFullSCOPF (BCMethod, "filename1", "filename2", CreateIfNotFound1, CreateIfNotFound2);	38
OPFWriteResultsAndOptions("filename");	38
PV Related Actions	39
PVSetSourceAndSink([elementSource], [elementSink]);	39
PVRun;	39
PVRun([elementSource], [elementSink]);	39
PVClear;	39
PVStartOver;	39
PVDestroy;	39
PVWriteResultsAndOptions("filename", AppendFile);	39
RefineModel(objecttype, filter, Action, Tolerance);	40
PVQVTrackSingleBusPerSuperBus;	40
QV Related Actions	41
QVRun("filename", InErrorMakeBaseSolvable);	41
QVWriteResultsAndOptions("filename");	41
RefineModel(objecttype, filter, Action, Tolerance);	41
QVSelectSingleBusPerSuperBus;	41
TS (Transient Stability) Related Actions	42
TSSolveAll;	42
TSSolve("ContingencyName", [StartTime, StopTime, StepSize]);	42
TSWriteOptions("FileName", [SaveDynamicModel, SaveStabilityOptions, SaveStabilityEvents, SaveResultsEvents, SavePlotDefinitions], KeyField);	42
TSSaveTwoBusEquivalent ("AuxFileName", [BUS]);	42
TSCalculateSMIBEigenValues;	42
TSLoadGE("FileName", GENCCYN);	42
TSLoadPTI("FileName", "MCREfilename", "MTRLOfilename", "GNETfilename", "BASEGENfilename");	42
TSLoadBPA("FileName");	43
TSGetResults("FileName", Single/Separate, [Contingencies], [Plots, ObjectFields], StartTime, EndTime);	43
TSAutoCorrect;	43
<b>DATA Section</b>	<b>44</b>
<b>DATA Argument List</b>	<b>44</b>
<b>Field Naming</b>	<b>45</b>
<b>Key Fields</b>	<b>45</b>

<b>Data List</b> .....	<b>46</b>
<b>Using Labels for Identification</b> .....	<b>46</b>
Saving Auxiliary Files Using Labels .....	47
Loading Auxiliary Files SUBDATA Sections Using Labels .....	47
Special Use of Labels in SUBDATA .....	47
<b>SubData Sections</b> .....	<b>49</b>
ATC_Options .....	50
RLScenarioName .....	50
GScenarioName .....	50
IScenarioName .....	50
ATCMemo .....	50
ATCExtraMonitor .....	50
ATCFlowValue .....	50
ATCScenario .....	50
TransferLimiter .....	50
ATCExtraMonitor .....	51
AUXFileExportFormatData .....	51
DataBlockDescription .....	51
AUXFileExportFormatDisplay .....	52
DataBlockDescription .....	52
BGCalculatedField .....	52
Condition .....	52
Bus .....	52
MWMarginalCostValues .....	52
MvarMarginalCostValues .....	52
LPOPFMarginalControls .....	52
BusViewFormOptions .....	52
BusViewBusField .....	52
BusViewFarBusField .....	52
BusViewGenField .....	52
BusViewLineField .....	52
BusViewLoadField .....	52
BusViewShuntField .....	52
ColorMap .....	53
ColorPoint .....	53
Contingency .....	54
CTGElementAppend .....	54
CTGElement .....	54
LimitViol .....	58
Sim_Solution_Options .....	59
WhatOccurredDuringContingency .....	59
ContingencyMonitoringException .....	59
CTG_Options .....	59
Sim_Solution_Options .....	59
CTGElementBlock .....	59
CTGElement .....	59
CTGElementAppend .....	59
CustomColors .....	60
CustomColors .....	60
CustomCaseInfo .....	60
ColumnInfo .....	60
DataGrid .....	60
ColumnInfo .....	60
DynamicFormatting .....	60
DynamicFormattingContextObject .....	60
LineThicknessLookupMap .....	61
LineColorLookupMap .....	61
FillColorLookupMap .....	61
FontColorLookupMap .....	61
FontSizeLookupMap .....	61

BlinkColorLookupMap .....	61
XoutColorLookupMap .....	61
FlowColorLookupMap .....	61
SecondaryFlowColorLookupMap .....	61
Filter .....	62
Condition .....	62
Gen .....	63
BidCurve .....	63
ReactiveCapability .....	63
GeoDataViewStyle .....	63
TotalAreaValueMap .....	63
RotationRateValueMap .....	63
RotationAngleValueMap .....	64
LineThicknessValueMap .....	64
GlobalContingencyActions .....	64
CTGElementAppend .....	64
CTGElement .....	64
HintDefValues .....	64
HintObject .....	64
InjectionGroup .....	65
PartPoint .....	65
Interface .....	66
InterfaceElement .....	66
KMLExportFormat .....	66
DataBlockDescription .....	66
LimitSet .....	67
LimitCost .....	67
Load .....	67
BidCurve .....	67
LPVariable .....	67
LPVariableCostSegment .....	67
ModelCondition .....	67
Condition .....	67
ModelExpression .....	68
LookupTable .....	68
ModelFilter .....	68
ModelCondition .....	68
MTDCRecord .....	69
MTDCBus .....	69
MTDCConverter .....	69
MTDCTransmissionLine .....	70
MultiSectionLine .....	71
Bus .....	71
BusRenumber .....	72
Nomogram .....	72
InterfaceElementA .....	72
InterfaceElementB .....	72
NomogramBreakPoint .....	72
NomogramInterface .....	72
InterfaceElement .....	72
Owner .....	73
Bus .....	73
Load .....	73
Gen .....	73
Branch .....	73
PostPowerFlowActions .....	73
CTGElementAppend .....	73
CTGElement .....	74
PWCaseInformation .....	74
PWCaseHeader .....	74
PWFormOptions .....	74

PieSizeColorOptions .....	74
PWLPOPFACTGViol .....	75
OPFControlSense .....	75
OPFBusSenseP .....	75
OPFBusSenseQ .....	75
PWLPTabRow .....	75
LPBasisMatrix .....	75
PWPVResultListContainer .....	75
PWPVResultObject .....	75
LimitViol .....	76
PVBUSInadequateVoltages .....	76
PWQVResultListContainer .....	76
PWPVResultObject .....	76
QVCurve .....	76
QVPoints .....	76
QVCurve_Options .....	77
Sim_Solution_Options .....	77
SelectByCriteriaSet .....	77
SelectByCriteriaSetType .....	77
Area .....	78
Zone .....	78
ScreenLayer .....	78
ShapefileExportDescription .....	78
StudyMWTransactions .....	78
ImportExportBidCurve .....	78
SuperArea .....	79
SuperAreaArea .....	79
TSSchedule .....	79
SchedPoint .....	79
UserDefinedDataGrid .....	79
ColumnInfo .....	79
<b>SCRIPT Section for Display Auxiliary File .....</b>	<b>80</b>
<b>AXD Actions .....</b>	<b>81</b>
AutoInsertBorders; .....	81
AutoInsertBuses(LocationSource, MapProjection, AutoInsertBranches, InsertIfNotAlreadyShown, "filename", FileCoordinates); .....	81
AutoInsertLoads(MinkV, InsertTextFields); .....	81
AutoInsertSwitchedShunts(MinkV, InsertTextFields); .....	81
AutoInsertLines(MinkV, InsertTextFields, InsertEquivObjects, InsertZBRPieCharts, InsertMSLines, ZBRImpedance, NoStubsZBRs, SingleCBZRs); .....	82
AutoInsertLineFlowObjects(MinkV, InsertOnlyIfNotAlreadyShown, LineLocation, Size, FieldDigits, FieldDecimals, TextPosition, ShowMW, ShowMvar, ShowMVA, ShowUnits, ShowComplex); .....	82
AutoInsertSubStations(LocationSource, MapProjection, AutoInsertBranches, InsertIfNotAlreadyShown, "filename", FileCoordinates); .....	82
AutoInsertLineFlowPieCharts(MinkV, InsertOnlyIfNotAlreadyShown, InsertMSLines, Size); .....	82
AutoInsertInterfaces(InsertPieCharts, PieChartSize); .....	82
ResetStubLocations(ZBRImpedance, NoStubsZBRs); .....	82
FixFlowArrowLineEnds("OnelineName", "LayerName"); .....	83
General Script Commands .....	83
<b>DATA Section for Display Auxiliary File .....</b>	<b>84</b>
<b>DATA Argument List .....</b>	<b>84</b>
<b>Key Fields .....</b>	<b>85</b>
<b>Data List .....</b>	<b>85</b>
<b>Special Data Sections .....</b>	<b>85</b>
GeographyDisplayOptions .....	85
Picture .....	86
PWFormOptions .....	86

View.....	86
<b>SubData Sections.....</b>	<b>87</b>
ColorMap.....	87
CustomColors.....	87
DisplayDCTransmissionLine.....	87
DisplayInterface.....	87
DisplayMultiSectionLine.....	87
DisplaySeriesCapacitor.....	87
DisplayTransformer.....	87
DisplayTransmissionLine.....	87
Line.....	87
Line.....	87
DynamicFormatting.....	88
Filter.....	88
GeoDataViewStyle.....	88
PieChartGaugeStyle.....	88
ColorMap.....	88
PWFormOptions.....	88
SelectByCriteriaSet.....	88
UserDefinedDataGrid.....	88
View.....	89
ScreenLayer.....	89



# Introduction

PowerWorld has incorporated the ability to import data to/from data sources other than power flow models into PowerWorld Simulator. The text file interface for exchanging data, as well as for executing a batch script command, is represented by the auxiliary files. The script language and auxiliary data formats are incorporated together. This format is described in this document.

Script/Data files are called data auxiliary files in Simulator and typically have the file extension .AUX. These files mostly contain information about power system elements and options for running the various tools within Simulator. They do not contain any information about the individual display objects contained on a one-line diagram. There are separate files called display auxiliary files that are available for importing display data to/from Simulator in a text format. These files are distinguished from the data auxiliary files by using the extension .AXD. The format for these two types of files is similar, but different object types are supported by each and require that the files be read separately.

Both file types will be generically referred to as auxiliary files. An auxiliary file may be comprised of one or more DATA or SCRIPT sections. A DATA section provides specific data for a specific type of object. A SCRIPT section provides a list of script actions for Simulator to perform. These sections have the following format:

```
SCRIPT ScriptName1
{
script_statement_1;
.
script_statement_n;
}

DATA DataName1(object_type, [list_of_fields], file_type_specifier, create_if_not_found)
{
data_list_1
.
data_list_n
}

DATA DataName2(object_type, [list_of_fields], file_type_specifier, create_if_not_found)
{
data_list_1
.
data_list_n
}

SCRIPT ScriptName2
{
script_statement_1;
.
script_statement_n;
}
```

Note that the keywords SCRIPT or DATA must occur at the start of a text file line. Auxiliary files may contain more than one DATA and/or SCRIPT section. These sections always begin with the keyword DATA or SCRIPT. DATA sections are followed by an argument list enclosed in ( ). The actual data or script commands are then contained within curly braces { }. Strings are enclosed in straight quotes – note that smart quotes will not work (this might be encountered when copy/pasting script commands from another program). The Script commands available in Simulator 17 are described in the next main section. The DATA sections are then described after this. There are separate sections for describing the DATA sections for the data auxiliary files and the display auxiliary file.

# SCRIPT Section

```
SCRIPT ScriptName
{
script_statement_1;
.
script_statement_n;
}
```

Scripts may optionally contain a ScriptName. This enables you to call a particular SCRIPT by using the LoadScript action (see General Actions). After the optional name, the SCRIPT section begins with a left curly brace and ends with a right curly brace. Inside of this, script statements can be given. In general, a script statement has the following format

```
Keyword(arg1, arg2, ...);
```

- Statement starts with a keyword.
- The keyword is followed by an argument list which is encompassed in parentheses ( ).
- The arguments are separated by commas.
- If a single argument is a list of things, this list is encompassed by braces [ ].
- Statements end with a semicolon.
- Statements may take up several lines of the text file.
- You may put more than one statement on a single text line.

Those familiar with using Simulator will know that there is a RUN and EDIT mode in Simulator. Some features in Simulator are only available in one mode or the other. This functionality will be preserved in the script language. In earlier versions of the software, certain functionality was organized by the "submode" feature. While existing scripts designed to work with submodes will still function as before, moving between submodes is no longer necessary.

Various script commands require that you be in RUN or EDIT mode. If a script requires this, then the script will automatically change modes.

## Using Filters in Script Commands

Many script commands allow the specification of a filename. Only those objects meeting this filter will be selected for the specified action. This filename can be the name of an advanced filter. Advanced filters belonging to a different objecttype can also be used depending on the objecttype in use. For example, if filtering generator objects a bus filter can also be used. When using an advanced filter that belongs to a different objecttype the format of the filter is "<Objecttype> filename" instead of just specifying the filename itself.

The filename can also be the name of a device filter. A device filter allows you to specify a particular object for filtering instead of a class of object. For example you might want to return all buses that belong to a particular substation. You can specify the device filter for the particular substation and then apply this to the bus objects. The format of a device filter is "<DEVICE> objecttype 'key1' 'key2' 'key3'".

# General Actions

## Generic Data Actions

Available to you regardless of the mode

ChangeData	(objecttype, [fieldlist], [valuelist], filter); (NOT AVAILABLE YET)
CopyFile	("oldfilename", "newfilename");
CreateData	(objecttype, [fieldlist], [valuelist]);
Delete	(objecttype, filter);
DeleteFile	("filename");
DeleteIncludingContents	(objecttype, filter);
EnterDistMasterPassword	(Password);
ImportData	("filename", FileType, CreateIfNotFound);
LoadAux	("filename", CreateIfNotFound);
LoadData	("filename", DataName, CreateIfNotFound);
LoadScript	("filename", ScriptName, CreateIfNotFound);
RenameFile	("oldfilename", "newfilename");
SaveData	("filename", filetype, objecttype, [fieldlist], [subdatalist], filter, [SortFieldList]);
SaveDataUsingExportFormat	("filename", filetype, "FormatName", ModelToUse);
SaveDataWithExtra	("filename", filetype, objecttype, [fieldlist], [subdatalist], filter, [SortFieldList], [Header_List], [Header_Value_List]);
SaveObjectFields	("filename", objecttype, [fieldlist]);
SelectAll	(objecttype, filter);
SendToExcel	(objecttype, [fieldlist], filter, UseColumnHeaders, "workbookname", "worksheetname");
SetCurrentDirectory	("filedirectory", CreateIfNotFound);
SetData	(objecttype, [fieldlist], [valuelist], filter);
StopAuxFile;	
UnSelectAll	(objecttype, filter);
WriteTextToFile	("filename", "text...");

### **CopyFile("oldfilename", "newfilename");**

Use this action to copy a file from within a script.

"oldfilename" : The present file name.  
"newfilename" : The new file name desired.

### **CreateData(objecttype, [fieldlist], [valuelist]);**

Use this action to create particular objects. Note that the key fields for the objecttype must be specified.

objecttype : The objecttype being set.  
[fieldlist] : A list of fields that you want to save.  
[valuelist] : A list of values to set the respective fields to.

### **Delete(objecttype, filter);**

Use this delete objects of a particular type. A filter may optionally be specified to only delete object which meet a filter.

objecttype : The objecttype being selected.  
filter : There are four options for the filter:  
Delete(objecttype); – No filter means to delete all objects of this type.  
Delete(objecttype, "filtername"); – See the Using Filters in Script Commands section for more information on specifying the filtername.  
Delete(objecttype, AREAZONE); – means delete those meeting the area/zone filters.  
Delete(objecttype, SELECTED); – means delete if Selected? = YES

### **DeleteFile("filename");**

Use this action to delete a file from within a script.

"filename" : The file name to delete.

### **DeleteIncludingContents(objecttype, filter);**

Use this to delete objects of a particular type and other objects that these contain. Currently, only multi-section lines (objecttype = MultiSectionLine) can be used with this command. The branches and dummy buses that belong to multi-section lines will also be deleted along with the multi-section lines. A filter may optionally be specified to only delete objects that meet a filter. The syntax is identical to the Delete(objecttype, filter); action above.

### **EnterDistMasterPassword(Password);**

Use this action to enter the master password used to unlock distributed machine login credentials.

Password : Password that must be specified to unlock the credentials.

### **ImportData("filename", FileType, CreateIfNotFound);**

Use this action to import data in various file formats that are not native to Simulator.

"filename" : Name of the file to import

FileType : Parameter that specifies the format of the data this is being read. Currently there is only one file format supported.

CROW : Describes system outages in a format that is created by the Equinox Control Room Operations Window application. This is used with the Scheduled Actions add-on tool.

CreateIfNotFound : Optional parameter that is NO by default. Set this to YES to create objects defined in the data if they do not already exist.

### **LoadAux("filename", CreateIfNotFound);**

Use this action to load another auxiliary file from within a script.

"filename" : The filename of the auxiliary file being loaded.

CreateIfNotFound : Set to YES or NO. YES means that objects which cannot be found will be created while reading in DATA sections from filename. If this parameter is not specified, NO is assumed.

### **LoadData("filename", DataName, CreateIfNotFound);**

Use this action to load a named Script Section from another auxiliary file. This will open the auxiliary file denoted by "filename", but will only execute the script section specified.

"filename" : The filename of the auxiliary file being loaded.

DataName : The specific ScriptName from the auxiliary file which should be loaded.

CreateIfNotFound : Set to YES or NO. YES means that objects which cannot be found will be created while reading in DATA sections from filename. If this parameter is not specified, NO is assumed.

### **LoadScript("filename", ScriptName, CreateIfNotFound);**

Use this action to load a named Script Section from another auxiliary file. This will open the auxiliary file denoted by "filename", but will only execute the script section specified.

"filename" : The filename of the auxiliary file being loaded.

ScriptName : The specific ScriptName from the auxiliary file which should be loaded.

CreateIfNotFound : Set to YES or NO. YES means that objects which cannot be found will be created while reading in SCRIPT sections from filename. If this parameter is not specified, NO is assumed.

### **RenameFile("oldfilename", "newfilename");**

Use this action to rename a file from within a script.

"oldfilename" : The present file name.

"newfilename" : The new file name desired.

### **SaveData("filename", filetype, objecttype, [fieldlist], [subdatalist], filter, [SortFieldList]);**

Use this action to save data in a custom defined format. The filter and [SortFieldList] are optional.

- "filename" : The file path and name to save.
- filetype : There are several options for the filetype
  - AUXCSV : save as a comma-delimited auxiliary data file.
  - AUX : save as a space-delimited auxiliary data file.
  - CSV : save as a normal CSV file without the AUX file syntax. The first few lines of the text file will represent the object name and field names.
  - CSVColHeader : save as a normal CSV file without the AUX file syntax. The first few lines of the text file will represent the object name and field names.
- objecttype : The objecttype being saved.
- [fieldlist] : A list of fields that you want to save. For numeric fields, the number of digits and the number of decimal places (digits to right of decimal) can be specified by using the following format for the field, variablename:location:digits:rod. **The keyword ALL can be used instead of the location number of a field to return all fields with the same variablename, e.g. MultBusTLRSens:ALL. The keyword ALL can be used instead of a list of fields to return all fields for a particular objecttype.**
- [subdatalist] : A list of the subdata objecttypes to save with each object record.
- filter : There are four options for the filter:
  - SaveData(...);- No filter specified means to save all objects of this type.
  - SaveData(..., "filtername"); - See the Using Filters in Script Commands section for more information on specifying the filtername.
  - SaveData(..., AREAZONE); - AREAZONE means save those that meet the area/zone/owner filters.
  - SaveData(..., SELECTED); - SELECTED means save those that are selected.
- [SortFieldList] : This allows the specification of a sort order in which the data will be saved. The format is: [variablename1:+:0, variablename2:-:1] where
  - variablename: is the name of the field to sort by. There is no limit to how many fields can be specified for sorting. For fields that require a location other than zero, variablename can be in the format fieldname:location.
  - + or - : for the second parameter indicates sort ascending for + and sort descending for -. This parameter must be specified.
  - 0 or 1 : for the third parameter 0 means case insensitive and do not use absolute value, 1 mean case sensitive or use absolute value. This parameter is optional.

### **SaveDataUsingExportFormat("filename", filetype, "FormatName", ModelToUse);**

Use this action to save data in a user-defined format that has previously been defined.

- "filename" : The file to save the data to
- filetype : There are several options for the filetype
  - AUXCSV : save as a comma-delimited auxiliary data file.
  - AUX : save as a space-delimited auxiliary data file.
  - CSV : save as a normal CSV file without the AUX file syntax. The first few lines of the text file will represent the object name and field names.
- FormatName : The name of the Object Export Format Description to use.
- ModelToUse : Optional parameter that indicates the model to use.
  - FULL : Full-topology model. This is the default if the parameter is omitted.
  - CONSOLIDATED : Consolidated planning-type model. This option will only work with the Topology Processing add-on.

**SaveDataWithExtra("filename", filetype, objecttype, [fieldlist], [subdatalist], filter, [SortFieldList], [Header\_List], [Header\_Value\_List]);**

Use this action to save data in a custom defined format. User-specified fields and field values can also be specified in the output. Optional parameters are filter, [SortFieldList], [Header\_List], and [Header\_Value\_List]. The syntax is identical to the SaveData() command with the following exceptions.

- Filetype : There are several options for the filetype
  - CSV : save as a normal CSV file without the AUX file syntax. The first few lines of the text file will represent the object name and field names.
  - CSVNOHEADER: save as a normal CSV text file, without the AUX file formatting. The object name and field names are NOT included. This option is useful when appending data of the same object type and field list into a common file.
  - CSVCOLHEADER: save as a normal CSV without the AUX syntax and with the first row showing column headers you would see in a case information display
- [Header\_List] : Data cannot be saved using AUX or AUXCSV filetypes with this command. This allows the specification of user-defined fields that will appear in the output. Headers should be specified as a list of comma delimited strings. A string should be enclosed in double quotes if the string contains a comma. Header strings cannot be blank.
- [Header\_Value\_List] : Allows the specification of the values that should be assigned to the user-defined fields specified by Header\_List. Specifying the values is optional. If specified, there must be as many values specified as there are headers. If not specified, all values are blank. Each object will use the same specified value for the specified field. To use different values for different objects and save these in the same file, make use of the CSVNOHEADER file format and filtering. [Special keywords can be entered that will be replaced with appropriate text. @BUILDDATE, @DATETIME, @DATE, @TIME, and @VERSION will be replaced with the appropriate values.](#)

**SaveObjectFields("filename", objecttype, [fieldlist]);**

Use this action to save a list of fields available for the specified objecttype to a CSV file. Format of the file is *variablename, field, col header, description*.

- "filename" : The file path and name to save.
- objecttype : The type of object for which fields should be saved.
- [fieldlist] : List of fields for which information will be saved. [The keyword ALL can be used instead of the location number of a field to return all fields with the same variablename, e.g. MultBusTLRSens:ALL. The keyword ALL can be used instead of a list of fields to return all fields for a particular objecttype.](#)

**SelectAll(objecttype, filter);**

Use this to set the selected property of objects of a particular type to true. A filter may optionally be specified to only set this property for objects which meet a filter.

- objecttype : The objecttype being selected.
- filter : There are three options for the filter:
  - SelectAll(objecttype); – No filter means to select all objects of this type.
  - SelectAll(objecttype, "filtername"); – See the Using Filters in Script Commands section for more information on specifying the filtername.
  - SelectAll(objecttype, AREAZONE); – means select those meeting the area/zone filters

### **SendToExcel(objecttype, [fieldlist], filter, UseColumnHeaders, "workbookname", "worksheetname");**

Use this action to mimic the behavior of the Send to Excel option found within a case information display.

- objecttype : The type of object for which fields should be saved.
- [fieldlist] : List of fields for which information will be saved. **The keyword ALL can be used instead of the location number of a field to return all fields with the same variable name, e.g. MultBusTLRSens:ALL. The keyword ALL can be used instead of a list of fields to return all fields for a particular objecttype.**
- filter : There are four options for the filter:
  - SendToExcel(...); – No filter specified means to save all objects of this type.
  - SendToExcel (..., "filtername"); – See the Using Filters in Script Commands section for more information on specifying the filtername.
  - SendToExcel (... , AREAZONE); – AREAZONE means save those that meet the area/zone/owner filters.
  - SendToExcel (... , SELECTED); – SELECTED means save those that are selected.
- UseColumnHeaders : Set to YES or NO. YES signifies that the first row shows the Column Header, NO signifies that variable names are used.
- "workbookname" : **Path and name of the workbook to save or modify. If no path is specified, the workbook will be saved or opened from the current directory. If the workbook already exists, it will be modified with a new worksheet, or if the worksheet is specified and already exists, the worksheet will be overwritten.**
- "worksheetname" : Optional parameter to specify the worksheet name to save. If blank, a new worksheet will be created, if a value is specified it will overwrite the data in any existing worksheet of that name.

### **SetCurrentDirectory("filedirectory", CreateIfNotFound);**

Use this action to set the current work directory.

- "filedirectory" : The path of the work directory.
- CreateIfNotFound : Set to YES or NO. YES means that if the directory path cannot be found, the directory will be created. If this parameter is not specified, NO is assumed.

### **SetData(objecttype, [fieldlist], [valuelist], filter);**

Use this action to set fields for particular objects. If a filter is specified, then it will set the respective fields for all objects which meet this filter. Otherwise, if no filter is specified, then the keyfields must be included in the field list so that the object can be found.

- objecttype : The objecttype being set.
- [fieldlist] : A list of fields that you want to save.
- [valuelist] : A list of values to set the respective fields to .
- filter : There are four options for the filter:
  - SetData(...); – No filter specified sets data only for the object described by the [fieldlist] and [valuelist] parameters.
  - SetData(..., ALL); – sets data for all objects.
  - SetData(..., "filtername"); – See the Using Filters in Script Commands section for more information on specifying the filtername.
  - SetData(..., AREAZONE); – sets data for all objects that meet the area/zone filters.
  - SetData(..., SELECTED); – sets data for all objects with Selected? = YES

### **StopAuxFile;**

Use this action to treat the remainder of the file after the command as a big comment. This includes any script commands inside the present SCRIPT block, as well as all remaining SCRIPT or DATA blocks.

### **UnSelectAll(objecttype, filter);**

Same as SelectAll, but this action sets the selected properties to false.

**WriteTextToFile("filename", "text...");**

Use this action to write text to a file. If the specified file already exists, the text will be appended to the file. Otherwise, it creates the file and writes the text to the file.

- "filename" : The file path and name to save.
- "text..." : The text to be written to the file. [Special keywords can be entered that will be replaced with appropriate text. @BUILDDATE, @DATETIME, @DATE, @TIME, and @VERSION will be replaced with the appropriate values.](#)



## Case Related Actions

Available to you regardless of the mode

```

AutoInsertTieLineTransactions;
CalculateRXBGFromLengthConfigCondType(filter);
                                unspecified
                                "filename"
                                SELECTED
                                AREAZONE

CaseDescriptionClear;
CaseDescriptionSet      ("text...",      Append);
                                YES or NO

ChangeSystemMVABase      (NewBase);
ConditionVoltagesOnRadialBuses(VoltageTreshold, AngleThreshold, filter);
DetermineBranchesThatCreateIslands(Filter, StoreBuses,"filename", SetSelectedOnLines, FileType);
                                ALL          YES/NO          YES/NO          AUX
                                SELECTED
                                AREAZONE          CSV
                                "FilterName"

DeterminePathDistance      ([start],      BranchDistMease, BranchFilter, BusField);
                                [AREA num]      X          ALL          variableName
                                [ZONE num]      Z          Selected
                                [SUPERAREA "name"] Length          Closed
                                [INJECTIONGROUP "name"] Nodes          "Filename"
                                [BUS num]      "Variablename"

DetermineShortestPath      ([start],      [end], BranchDistMeas,BranchFilter,Filename);
                                [AREA num]      same X          ALL
                                [ZONE num]      as Z          Selected
                                [SUPERAREA "name"] for Length          Closed
                                [INJECTIONGROUP "name"] start Nodes          "Filename"

DirectionsAutoInsert      (Source, Sink, DeleteExisting, UseDisplayFilters, Start, Increment);
                                AREA AREA YES          YES          value value
                                ZONE ZONE NO          NO
                                INJECTIONGROUP
                                SLACK

EnterMode      (mode);
                                EDIT
                                RUN

ExitProgram;
GenForceLDC_RCC      (filter);
                                unspecified
                                "filename"
                                SELECTED
                                AREAZONE

InjectionGroupsAutoInsert;
LogAdd      ("string...");
LogAddDateTime      ("label", includedate, includetime, includemilliseconds);
LogClear;
LogSave      ("filename", AppendFile);
                                YES or NO

NewCase;
OpenCase      ("filename"); // assumes to open as PWB
OpenCase      ("filename", openfiletype );
                                PWB, GE, PTI, CF
OpenCase      ("filename", openfiletype, [LoadTransactions, StarBus]);
                                PTI          YES, NO, DEFAULT NEAR, MAX, number
OpenCase      ("filename", openfiletype, [MSLine, VarLimDead, PostCTGAGC]);
                                GE          MAINTAIN number YES
                                EQUIVALENCE          NO

SaveCase      ("filename"); // assumes save as PWB
SaveCase      ("filename", savefiletype,
                                PWB,PWB5,PWB6,PWB7,PWB8,PWB9,PWB10,PWB11,PWB12
                                PWB13,PWB14,PWB15,PWB16,PWB17
                                PTI,PTI23,PTI24,PTI25,PTI26,PTI27,PTI28,PTI29,PTI30
                                PTI31,PTI32,PTI33
                                GE,GE14,GE15,GE17,GE18,CF
                                [PostCTGAGC, UseAreaZone]);
                                YES/NO          YES/NO

```

SaveJacobian	("JacFileName", "JIDFileName",	FileType,	JacForm)
		M or TXT	R or P
		EXPM	R
SaveYbusInMatlabFormat	("filename", IncludeVoltages);		
Scale	(scaletype, basedon, [parameters],	ScaleMarker);	
	LOAD MW [P, Q]	BUS	
	GEN FACTOR [P] means constant pf	AREA	
	INJECTIONGROUP	ZONE	
SetGenPMaxFromReactiveCapabilityCurve(filter);	Selected		
	AREAZONE		
	FilterName		
		OWNER	
SetParticipationFactors	(Method, ConstantValue, Object);		
	MAXMWRAT [Area num]		
	RESERVE [Zone num]		
	CONSTANT SYSTEM		
		AREAZONE or DISPLAYFILTERS	
	[BUS num]	"Variablename"	
SetScheduledVoltageForABus([bus identifier], voltage);			

### AutoInsertTieLineTransactions;

Use this action to delete all existing MW transactions and set the unspecified MW interchange for each area to zero. It then automatically creates a MW transaction between each pair of connected areas with a MW transaction exactly equal to the sum of the tie-line flows.

### CalculateRXBGFromLengthConfigCondType(filter);

Use this action to go through branches in the power system and automatically recalculate the per unit R, X, G, and B values using the TransLineCalc tool. The branches Conductor Type, Tower Configuration, and Line Length will be passed to the TransLineCalc tool and new R, X, G and B values will be calculated. This is only available if you have installed the TransLineCalc tool.

- filter : This parameter is used to specify which branches are checked.
- ALL : means all branches will be checked
  - SELECTED : means only branches whose Selected field = YES will be checked
  - AREAZONE : means only branches that meet the area/zone/owner filters will be checked
  - "FilterName" : means only branches that meet the specified filter will be checked. See the Using Filters in Script Commands section for more information on specifying the filtername.

### CaseDescriptionClear;

Use this action to clear the case description of the presently open case.

### CaseDescriptionSet("text...", Append);

Use this action to set or append text to the case description.

- "text..." : Specify the text to set/append to the case description.
- Append : YES – will append the text specified to the existing case description. NO – will replace the case description.

### ChangeSystemMVABase(NewBase);

Use this action to change the system MVA base to the specified value and update all internal data structures to store values on the new base.

- NewBase : New power system base in MVA.

### ConditionVoltagesOnRadialBuses(VoltageThreshold, AngleThreshold, filter);

The goal of this script command is to find groups of radial buses that may have bad initial voltage estimates and to get a better voltage estimate of these buses based on assuming that the voltages on buses outside of the radial group are good. These radial groups are determined by finding branches that would cause islands AND the voltage difference (absolute value) across the branch is greater than the VoltageThreshold OR the angle difference (absolute value) across the branch is greater than the AngleThreshold, thus identifying parts of the system that have large voltage or angle differences.

- VoltageThreshold : Per-unit voltage difference (absolute value) that determines if a branch can be considered when determining groups of radial buses.

- AngleThreshold : Angle difference in degrees (absolute value) that determines if a branch can be considered when determining groups of radial buses.
- filter : This is an optional parameter is used to specify which branches are checked. If omitted all branches are considered.
  - ALL : All branches will be checked
  - SELECTED : Only branches whose Selected field = YES will be checked
  - AREAZONE : Only branches that meet the area/zone/owner filters will be checked
  - "FilterName" : Only branches that meet the specified filter will be checked. See Using Filters in Script Commands section for more information on specifying the filename.

**DetermineBranchesThatCreateIslands(Filter, StoreBuses, "filename", SetSelectedOnLines, FileType);**

Use this action to determine the branches whose outage results in island formation. Note that setting the Selected field will overwrite the Selected fields.

- Filter : This parameter is used to specify which branches are checked.
  - ALL : means all branches will be checked
  - SELECTED : means only branches whose Selected field = YES will be checked
  - AREAZONE : means only branches that meet the area/zone/owner filters will be checked
  - "FilterName" : means only branches that meet the specified filter will be checked. See the Using Filters in Script Commands section for more information on specifying the filename.
- StoreBuses : YES to store the buses in the island to the output file
- "filename" : file to which the results will be written. The format of the file is based on the auxiliary file format. Each branch that was checked will be followed by the list of buses that are islanded. The branch and bus information will be written in appropriate auxiliary file DATA format.
- SetSelectedOnLines : YES to set the SELECTED field to YES for branches that create islands
- FileType : Optional parameter used to specify the format of the file. This is AUX by default.
  - AUX : The saved file is based on an auxiliary file data format. Each branch that causes an island appears in the file in the auxiliary file data format followed by a auxiliary file bus data section containing all of the buses that are islanded by the preceeding branch.
  - CSV : The saved file is a comma-delimited text file. Each unique bus/branch pair appears on a single line. A unique bus/branch pair is determined by a bus that is islanded and a particular branch that causes it to be islanded. A header appears in the file specifying the fields used to identify the branch and bus in each record.

**DeterminePathDistance([start], BranchDistMeas, BranchFilter, BusField);**

Use this action to calculate a distance measure at each bus in the entire model. The distance measure will represent how far each bus is from the starting group specified. The distance measure can be related to impedance, geographical distance, or simply the number of nodes.

- [start] : is an the starting place which is enclosed in brackets. The starting place may be either an Area, Zone, SuperArea, Substation, Injection Group, or Bus. Format of string is
  - [Area Num], [Area "Name"], or [Area "label"]
  - [Zone Num], [Zone "Name"], or [Zone "label"]
  - [SuperArea "Name"] or [SuperArea "label"]
  - [Substation Num] or [Substation "label"]
  - [InjectionGroup "Name"] or [InjectionGroup "label"]
- BranchDistMeas : is either X, Z, Length, Nodes, or a variable name for a branch.
  - X : means use the series reactance,
  - Z : means use  $\sqrt{X^2 + R^2}$ ,
  - Length : means us the Length field, and
  - Nodes : means treat each branch as a length of one.
  - "Variablename" : Otherwise use any Branch object variable name.

- BranchFilter : is either All, Selected, Closed or the name of a branch Advanced Filter. This parameter is used to specify which branch can be traversed at all.
- All : means all branches can be traversed
  - Selected : means only branches whose Selected field = YES can be traversed
  - Closed : means only branches that are CLOSED can be traversed.
  - "FilterName" : See the Using Filters in Script Commands section for more information on specifying the filename.
- BusField : is the variablename of a Bus field. This field is populated with the minimum distance from the Start Place to that bus. All buses in the start group will have a distance measure of zero. Buses which cannot be reached from the start group will have a distance measure of -1.

#### **DetermineShortestPath([start], [end], BranchDistanceMeasure, BranchFilter, Filename);**

Use this action to calculate the shortest path between a starting group and an ending group. The results will be written to a textfile specified by filename. In the text file, the first bus listed will be in the end grouping and the last bus listed will be the start grouping. The result text file will have a line for each bus passed. Each line will contain three entries delimited by a space: "Number DistanceMeasure Name".

- [start] : same as the starting place for the DeterminePathDistance script command
- [end] : same as the starting place for the DeterminePathDistance script command
- BranchDistanceMeasure: same as for DeterminePathDistance script command
- BranchFilter : same as for DeterminePathDistance script command
- Filename : is a filename (may need to be enclosed in quotes) to which the results will be written.

#### **DirectionsAutoInsert(Source, Sink, DeleteExisting, UseAreaZoneFilters, Start, Increment);**

Use this action to auto-insert directions to the case

- Source : AREA, ZONE, or INJECTION GROUP – specifies what to use as source
- Sink : AREA, ZONE, INJECTION GROUP, or SLACK – specifies what to use as sink.
- DeleteExisting : YES – to delete existing direction; NO to not do that.
- UseAreaZoneFilters : YES – to filter Area/Zones by filter.
- Start : The starting number for the new directions added.
- Increment : The increment for subsequent directions.

#### **EnterMode(mode or submode);**

This action will save part of the power system to a "filename". It will save only those buses whose property pwEquiv must be set true.

- SubMode : The submode to enter. A parameter stating what submode to put the program in. Options available are CASE, POWERFLOW, CONTINGENCY, ATC, FAULT, PV, QV. One may also put in RUN or EDIT which will place the program in the POWERFLOW or CASE respectively.

#### **ExitProgram;**

Immediately exits the program with no prompts.

#### **GenForceLDC\_RCC(filter);**

Use this action to force generators in the case onto line drop / reactive current compensation. **The present voltage at the point at which the generator is controlling based on the line drop/reactive current compensation impedance is calculated, and the setpoint of the generator is set to this value. If the absolute value of the line drop/reactive current compensation impedance is less than or equal to  $2 \times 10^{-6} \times \text{MVA Base}$ , the generator will regulate its terminal bus and the setpoint voltage is set to the present value of the terminal bus voltage. For a typical case with an MVA Base of 100 MVA, this value is 0.0002.**

- filter : There are four options for the filter:
  - GenForceLDC\_RCC; – No filter specified means to set all generators.
  - GenForceLDC\_RCC("filename"); – See the Using Filters in Script Commands section for more information on specifying the filename.
  - GenForceLDC\_RCC(AREAZONE); – AREAZONE means to set generators that meet the area/zone filters.
  - GenForceLDC\_RCC(SELECTED); – means set generators if Selected?=YES

### **InjectionGroupsAutoInsert;**

Use this action to automatically insert injection groups according to the options specified in the "IG\_AutoInsert\_Options" object. The settings available with this object represent what is seen on the Auto Insert Injection Groups Dialog.

### **LogAdd("string...");**

Use this action to add a personal message to the MessageLog.  
"string..." : The string that will appear as a message in the log.

### **LogAddDateTime("label", includedate, includetime, includemilliseconds);**

Use this action to add the date and time to the message log  
"label" : A string which will appear at the start of the line containing the date/time.  
includedate : YES – Include the data or NO to not include.  
includetime : YES – Include the time or NO to not include.  
includemilliseconds : YES – Include the milliseconds or NO to not include.

### **LogClear;**

Use this action to clear the Message Log.

### **LogSave("filename", AppendFile);**

This action saves the contents of the Message Log to "filename".  
"filename" : The file name to save the information to.  
AppendFile : Set to YES or NO. YES means that the contents of the log will be appended to "filename". NO means that "filename" will be overwritten.

### **NewCase;**

This action clears out the existing case and open a new case from scratch.

### **OpenCase("filename", OpenFileType,[LoadTransactions,StarBus,MSLine,VarLimDead,PostCTGAGC]);**

This action will open a case stored in "filename" of the type OpenFileType. Different sets of optional parameters apply for the PTI and GE file formats. The LoadTransactions and Star bus parameters are available for writing to RAW files. MSLine, VarLimDead, and PostCTGAGC are for writing EPC files.

"filename" : The file to be opened.  
OpenFileType : An optional parameter saying what the format of the file being opened is. If none is specified, then PWB will be assumed. It may be one of the following strings  
PWB, PTI, PTI23, PTI24, PTI25, PTI26, PTI27, PTI28, PTI29, PTI30, PTI31, PTI32, PTI33  
GE (means GE18), GE14, GE15, GE17, GE18, CF  
AUX, UCTE, AREVAHDB  
LoadTransactions : valid for PTI RAW format only  
YES -- load transactions when opening case.  
NO -- do not load transactions when opening case.  
DEFAULT – follow default behavior.  
StarBus : valid for PTI RAW format only  
NEAR -- star buses are numbered starting after the near bus number  
MAX -- star buses are numbered starting with the maximum bus number  
VALUE -- star bus numbering will start at value  
MSLine : valid for GE EPC format only  
MAINTAIN – maintain multi-section lines  
EQUIVALENCE – equivalence mult-section lines  
VarLimDead : valid for GE EPC format only  
Number -- set the GE var limit deadband  
PostCTGAGC : valid for GE EPC format only  
set to YES to populate the generator field Post-CTG Prevent Response based on the EPC file's generator base load flag.

### **SaveCase("filename", SaveFileType, [PostCTGAGC, UseAreaZone]);**

This action will save the case to "filename" in the format SaveFileType.

- "filename" : The file name to save the information to. Special keywords can be entered as part of the filename. Simulator will replace these special keywords with appropriate information as part of the filename when creating the filename to save. @DATETIME will replace the keyword with the actual date and time in the format yyyyymmdd\_hhnnss-hhmm with the UTC offset included on the end of the time. The other special keywords @DATE, @TIME, @BUILDDATE, and @VERSION can also be used.
- SaveFileType : An optional parameter saying the format of the file to be saved. If none is specified, then PWB will be assumed. It may be one of the following strings PWB (means PWB17), PWB5, PWB6, PWB7, PWB8, PWB9, PWB10, PWB11, PWB12, PWB13, PWB14, PWB15, PWB16, PWB17 PTI (means PTI32), PTI23, PTI24, PTI25, PTI26, PTI27, PTI28, PTI29, PTI30, PTI31, PTI32, PTI33 GE (means GE18), GE14, GE15, GE17, GE18, CF AUX, AUXSECOND, AUXLABEL, UCTE
- PostCTGAGC : An optional parameter, only valid for GE EPC format. YES sets the base load flag in the EPC file based on the Post-Contingency Prevent AGC Response setting. If preventing post-contingency AGC, the base load flag is set to 1. If not preventing post-contingency AGC or this option is set to NO, the base load flag is set to 0.
- UseAreaZone : An optional parameter, only valid for GE EPC format. YES limits the entries in the EPC file based on the area/zone/owner filter (NO by default)

### **SaveJacobian("JacFileName", "JIDFileName", FileType, JacForm)**

Use this action to save the Jacobian Matrix to a text file or a file formatted for use with Matlab

- "JacFileName" : File to save the Jacobian to.
- "JIDFileName" : File to save a description of what each row and column of the Jacobian represents.
- FileType : M – Matlab form.  
TXT – Text file.  
EXPM – Save Jacobian in Exponential form (ex. 1.2E-2) in Matlab form.
- JacForm : R – Rectangular coordinates Jacobian.  
P – Polar coordinates Jacobian.

### **SaveYbusInMatlabFormat("filename", IncludeVoltages);**

Use this action to save the YBus to a file formatted for use with Matlab

- "filename" : File to save the YBus to.
- IncludeVoltages : YES – Includes the per unit bus voltages in the file; NO does not include.

### **Scale(scaletype, basedon, [parameters], scalemarker);**

Use this action to scale the load and generation in the system.

- scaletype : The objecttype begin scaled. Must be either LOAD, GEN, INJECTIONGROUP, or BUSSHUNT.
- basedon : MW – parameters are given in MW, MVAR units.  
FACTOR – parameters a factor to multiple the present values by.
- [parameters] : These parameters have different meanings depending on ScaleType.  
LOAD : [MW, MVAR] or [MW]. If you want to scale load using constant power factor, then do not specifying a MVAR value.  
GEN : [MW]  
INJECTIONGROUP : [MW, MVAR] or [MW] . If you want to scale load using constant power factor, then do not specifying a MVAR value.  
BUSSHUNT : [GMW, BCAPMVAR, BREAMVAR]. The first values scales G shunt values, the second value scales positive (capacitive) B shunt values, and the third value scales negative (reactive) B shunt values

- scalemarker : This value specifies whether to look at an element's bus, area or zone to determine whether it should be scaled.
- BUS : Means that elements will be scaled according to the Scale property of the element's terminal bus.
  - AREA : Means that elements will be scaled according to the Scale property of the element's Area. Note that it is possible for the area of a load, generator, or switched shunt to be different than the terminal bus's area.
  - ZONE : Means that elements will be scaled according to the Scale property of the element's Zone. Note that it is possible for the zone of a load, generator, or switched shunt to be different than the terminal bus's zone.
  - OWNER : Means that elements will be scaled according to the Scale property of the element's Owner. Note that it is possible for the Owner of a load, generator, or switched shunt to be different than the terminal bus's Owner.

### **SetGenPMaxFromReactiveCapabilityCurve(filter);**

Use this action to change the Maximum MW output of generators which use a capability curve, equal to the second-to-last MW point in the capability curve if the last Max Mvar point on the capability curve is smaller than 0.001 Mvar. If the present MW output is higher than this new Max MW value, then Max MW is set to the present MW output..

- filter : optional parameter which is either Selected, Closed or the name of a branch Advanced Filter. This parameter is used to specify which generators are processed. If blank, all generators are processed.
- Selected : means only generators whose Selected field = YES can be processed
  - AREAZONE : means process those generators that meet the area/zone/owner filters.
  - "FilterName" : See the Using Filters in Script Commands section for more information on specifying the filename.

### **SetParticipationFactors(Method, ConstantValue, Object);**

Use this action to modify the generator participation factors in the case

- Method : The formula used to calculate the participation factors for each generator. It may be one of the following strings:  
 MAXMWRAT – base factors on the maximum MW ratings.  
 RESERVE – base factors on the (Max MW rating – Present MW).  
 CONSTANT – set factors to a constant value.
- ConstantValue : The value used if CONSTANT method is specified. If CONSTANT method is not specified, enter 0 (zero).
- Object : Specify which generators to set the participation factor for.  
 [Area Num], [Area "name"], [Area "label"]  
 [Zone Num], [Zone "name"], [Zone "label"]  
 SYSTEM  
 AREAZONE or DISPLAYFILTERS

### **SetScheduledVoltageForABus([bus identifier], voltage);**

Use this action to set the stored scheduled voltage, vsched, for a bus according to how this is defined in the EPC format. This value is not used by Simulator but is stored for purposes of writing out to an EPC file. The setpoint voltages for generators and switched shunts regulating the specified bus are also set to the new voltage. The regulation range for switched shunts is modified for the new setpoint voltage according to how this is defined in the EPC format: vband = (VHigh-VLow)/2 with newVHigh = voltage+vband and new VLow = voltage-vband.

- [bus identifier] : specifies bus  
 voltage : the new voltage

## Online Actions

Available to you regardless of the mode

```
CloseOnline          ("OnlineName");
ExportOnlineAsShapeFile
    ("filename", "OnlineName", "shapefileExportDescriptionName", UseLonLat, PointLocation);
                                                                NO or YES
LoadAXD              ("filename", "OnlineName", CreateIfNotFound);
OpenOnline           ("filename", "view", FullScreen, ShowFull, LinkMethod);
SaveOnline           ("filename", "OnlineName", SaveFileType);
```

### CloseOnline("OnlineName");

Use this action to close an open online diagram without saving it. If the name is omitted, the last focused online diagram will be closed.

" OnlineName" : The name of the online diagram to close.

### ExportOnlineAsShapeFile("filename", "OnlineName", "ShapeFileExportDescriptionName", UseLonLat, PointLocation)

Use this action to save an open online diagram to a shapefile.

"filename" : The file name of the shapefile to save.

" OnlineName" : The name of the online diagram to save to a shapefile. The online diagram must be open. Use the OpenOnline script command if necessary to open the appropriate online.

"ShapeFileExportDescriptionName" : Name of the ShapeFile Export Description to use when saving the shapefile.

UseLonLat : Set to YES or NO. YES means that the coordinates of objects on the online diagram will be saved using longitude,latitude. This will only be true if a valid map projection is in use with the online diagram. Otherwise, the coordinates will be saved in x,y. If this parameter is set to NO, the coordinates will be saved in x,y. If this parameter is not specified, YES is assumed.

PointLocation : Determines where points are specified – object centers, or the upper left corner. Specify “center” to define points as the shape centers, or “ul” to define them as the upper left corner of the shapes. If not specified, upper left is assumed.

### LoadAXD("filename", "OnlineName", CreateIfNotFound)

Use this action to apply a display auxiliary file to an open online diagram.

"filename" : The file name of the display auxiliary file to load.

"OnlineName" : The name of the online diagram to which to apply the display auxiliary file. If the online is not already open and SimAuto or Retriever is being used, the OpenOnline script command can be used to open the appropriate online. If the specified online is not open, a new one will be created with the given name if using SimAuto or Retriever. If using Simulator, the online must already be open or this script command will not work.

CreateIfNotFound : Set to YES or NO. YES means that objects which cannot be found will be created while reading in DATA sections from filename. If this parameter is not specified, NO is assumed.

### OpenOnline("filename", "view", FullScreen, ShowFull, LinkMethod);

Use this action to open a online diagram. This can only be used in SimAuto or Retriever. When using SimAuto, this action cannot be used to actually view a online. This script can be used in SimAuto to associate onelines with a PWB file. Any online that is opened using the script command and while the case is saved will opened in the GUI once the case is reopened.

"filename" : The file name of the online diagram to open.

"view" : The view name that should be opened. Pass an empty string to denote no specific view.

FullScreen : Set to YES or NO. YES means that the online diagram will be open in full screen mode. If this parameter is not specified, then NO is assumed.

ShowFull : Optional parameter. Set to YES to open the online and apply the Show Full option. Set to NO to open the online and leave the online as sis. Default is NO if not specified.

LinkMethod : Optional Parameter that controls online linking. LABELS, NAMENOMKV, and NUMBER will link using the respective key fields.



**SaveOnline("filename", "OnlineName", SaveFileType);**

Use this action to save an open online diagram to file

- "filename" : The path and file name of the file to save. If a full path is not specified, then the file is saved to the current directory.
- "OnlineName" : Name of the open online to save.
- SaveFileType : Type of file to save. Valid options are AXD, PWB, PWB5, PWB6, PWB7, PWB8, PWB9, PWB10, PWB11, PWB12, PWB13, PWB14, PWB15, PWB16, and PWB17. If omitted, PWB, which is the most recent version, will be assumed. Note the use of "PWB" instead of "PWD" is not a typo. The version of the PWD file corresponding to the PWB version will be used.

## Edit Mode Actions

### Case Related Actions

The following script commands are available during Edit mode

```
AppendCase ("filename", OpenFileType);
AppendCase ("filename", OpenFileType, [StarBus]);
AppendCase ("filename", OpenFileType, [MSLine, VarLimDead, PostCTGAGC]);
Combine ([elementA], [elementB]);
DeleteExternalSystem;
Equivalence;
InterfacesAutoInsert (Type, DeleteExisting, UseFilters, Prefix, Limits);
MergeBuses ([element], Filter);
MergeLineTerminals (Filter);
Move ([elementA], [destination parameter], HowMuch);
Renummer3WXFormerStarBuses("filename");
RenummerMSLineDummyBuses("filename");
SaveExternalSystem ("filename", Savefiletype, withties);
SplitBus ([element], NewBusNumber, InsertBusTieLine, LineOpen);
TapTransmissionLine ([element], PosAlongLine, NewBusNumber, ShuntModel, TreatAsMSLine);
```

**AppendCase("filename", OpenFileType);**

**AppendCase("filename", OpenFileType, [StarBus]);**

### **AppendCase("filename", OpenFileType, [MSLine, VarLimDead, PostCTGAGC]);**

Use this action to append a case to the currently open case. Optional parameters are StarBus, MSLine, VarLimDead, and PostCTGAGC. StarBus is an optional parameter that can be used when reading RAW files that can be used to control the numbering of star buses.

"Filename"	:	File name of the case to be appended
OpenFileType	:	PWB – case file is a PowerWorld Binary file GE – case file is a GE .epc file PTI – case file is a PTI .raw file IEEECF – case file is an IEEE common data format file
StarBus	:	Only used for PTI RAW format, with the following options: NEAR – star buses are numbered starting after the near bus number MAX – star buses are numbered starting with the maximum bus number Value – star bus numbering will start at value
MSLine	:	Only used for the GE EPC format, with the following options: MAINTAIN – maintain multisection lines EQUIVALENCE – equivalence multisection lines
VarLimDead	:	Only used for the GE EPC format NUMBER – set the GE var limit deadband
PostCTGAGC	:	Only used for the GE EPC format. Set to YES to populate the generator field Post-CTG Prevent Response based on the EPC file's generator base load flag.

### **Combine([elementA], [elementB]);**

NOTE: THIS ACTION IS ONLY AVAILABLE FOR GENERATORS

Use this action to combine two generators, two loads, or two transmission line. Note that elementA and elementB must be of the same object type. You cannot combine a BRANCH and a LOAD.

[elementA]	:	The object that should be moved. See the format for [elementA] in the Move() script command for information on the formatting of this string.
[elementB]	:	The object that element A should be combined with. Same format as for elementA.

### **DeleteExternalSystem;**

This action will delete part of the power system. It will delete those buses whose property Equiv must is set true.

### **Equivalence;**

This action will equivalence a power system. All options regarding equivalencing are handled by the Equiv\_Options objecttype. Use the SetData() action, or a DATA section to set these options prior to using the Equivalence() action. Also, remember that the property Equiv must be set true for each bus that you want to equivalence.

### **InterfacesAutoInsert(Type, DeleteExisting, UseFilters, "Prefix", Limits);**

Use this action to auto-insert interfaces

Type	:	AREA – insert area-to-area tieline interfaces. ZONE – insert zone-to-zone tieline interfaces.
DeleteExisting	:	YES – to delete existing interfaces; NO – to leave existing interfaces alone.
UseFilters	:	YES – to user Area/Zone Filters; NO – to insert for entire case.
"Prefix"	:	Enter a string which will be a prefix on the interface names.
Limits	:	ZEROS – to make all limits zero. AUTO – limits will be set to the sum of the branch limits. [lima, limb, limc, limd, ...] – Enter 8 limits enclosed in brackets, separated by commas. This will set the limits as specified.

### **MergeBuses([element], Filter);**

Use this action to merge buses

Element	:	Enter the description of the bus that the other buses will be merged into.
Filter	:	Enter the number of the new bus to be created. MergeBuses([element]); – No filter means to merge all buses into one. MergeBuses([element], "filename"); – See the Using Filters in Script Commands section for more information on specifying the filename. MergeBuses([element], AREAZONE); – AREAZONE means to merge those that meet area/zone filters. MergeBuses([element], SELECTED); – means merge those if Selected?=YES

### MergeLineTerminals(Filter);

Use this action to merge line terminals. This action can be used to remove a line by merging the terminal buses of that line into a single bus. The only parameter of the script command is a filter parameter, which must be populated with either the name of an advanced filter (with the name in quotation marks) or the text SELECTED (with no quotation marks). If an advanced filter is given, then Simulator will find all branches that meet the advanced filter definition and will individually merge the line terminals of each line one at a time.

Filter : "filename" – See the Using Filters in Script Commands section for more information on specifying the filename.  
SELECTED – select objects that have the selected property set to true

### Move([elementA], [destination parameters], HowMuch);

Use this action to move a generator, load, transmission line, or switched shunt.

[elementA] : The object that should be moved. Must be one of the following formats:

[GEN busnum id], [GEN "name\_nomkv" id],  
[GEN "buslabel" id], [GEN "label"]  
[LOAD busnum id] , [LOAD "name\_nomkv" id],  
[LOAD "buslabel" id], [LOAD "label"],  
[BRANCH busnum1 busnum2 ckt],  
[BRANCH "name\_kv1" "name\_kv2" ckt],  
[BRANCH "buslabel1" "buslabel2" ckt], [BRANCH "label"]  
[SHUNT busnum id], [SHUNT "name\_nomkv" id],  
[SHUNT "buslabel" id], [SHUNT "label"]  
[MULTISECTIONLINE busnum1 busnum2 ckt],  
[MULTISECTIONLINE "name\_kv1" "name\_kv2" ckt],  
[MULTISECTIONLINE "buslabel1" "buslabel2" ckt]  
[MULTISECTIONLINE "label"]

[destination parameters] : These parameters have different meanings depending on object type of the element. Must use bus numbers here

GEN : [busnum id]  
LOAD : [busnum id]  
BRANCH : [busnum1 busnum2 id]  
SHUNT : [busnum id]

HowMuch : The amount of the element to move. A value of 1 indicates that 100% should be moved. This parameter is only valid for generators and loads. It is ignored for lines and switched shunts.

### Renumber3WXFormerStarBuses("filename");

Use this action to renumber star buses based on user-specified values.

"filename" : The name of the file containing the renumbering

The file may be comma or space delimited. The contents of the file should be formatted using the format below.

*Primary bus, secondary bus, tertiary bus, circuit, new starbus number, new starbus name*

*or*

*pribusname\_nomkV, secbusname\_nomkV, terbusname\_nomkV, circuit ID, newstarbusnum, newstarbusname*

Either the bus number or the name\_nominalkV identifier may be used to identify the buses. Each bus may be identified using either method even for the same transformer. Lines starting with two slashes (//) will be ignored. The next two lines are sample file contents using different methods to identify buses.

```
11037, 11038, 11199, "1", 11202, "Ki star"  
"WESTWING_500.00" "WESTWNGW_230.00" "WESTWG 4_34.50" "2" 99823 "KI STAR 3"
```

### RenumberMSLineDummyBuses("filename");

Use this action to renumber dummy buses or a multisection line based on user-specified values.

"filename" : The name of the file containing the renumbering

The file may be comma or space delimited. Buses may be identified using bus numbers or using the BusName\_NominalkV combination. The file format is below:

```
from bus, to bus, circuit //identify multi-section line
dummybusnumber1, dummybusname1
dummybusnumber2,dummybusname2
```

where the dummy bus numbers and names give the numbers and names that will be assigned for the dummy buses of a multi-section line. An example of the file contents is below:

```
40039 , 40141, 1 // ALFALFA 230 N BONNVL 230 #1
49997, "ALFN B11"
40062 , 40699, 2 // ASHE R1 500 MARION 500 #2
49990, ASHMAR21
49989, ASHMAR22
49988, ASHMAR23
```

### SaveExternalSystem("Filename", SaveFileType, WithTies);

This action will save part of the power system to a "filename". It will save only those buses whose property Equiv must is set true.

filename : The file name to save the information to.  
SaveFileType : An optional parameter saying the format of the file to be saved. If none is specified, then PWB will be assumed. May be one of the following strings PWB, PWB5, PWB6, PWB7, PWB8, PWB9, PWB10, PWB11, PWB12, PWB13, PWB14, PWB15, PWB16, PWB17 PTI (means PTI32), PTI23, PTI24, PTI25, PTI26, PTI27, PTI28, PTI29, PTI30, PTI31, PTI32, PTI33 GE (means GE18), GE14, GE15, GE17, GE18, CF, AUX  
WithTies : An optional parameter. The user must specify the file type explicitly in order to use the WithTies parameter. Allows saving a transmission line that ties a bus marked with Equiv as false and one marked true. This must be a string which starts with the letter Y, otherwise NO will be assumed.

### SplitBus([element], NewBusNumber, InsertBusTieLine, LineOpen);

Use this action to split buses

Element : Enter the description of which bus to split by enclosing in brackets the word bus and an identifier. The format looks as follows.  
[BUS num]  
[BUS "name\_nomkv"]  
[BUS "buslabel"]  
NewBusNumber : enter the number of the new bus to be created  
InsertBusTieLine : YES – insert a low impedance tie line between the buses; NO – to not do that.  
LineOpen : YES – to make the inserted bus tie open; NO – to make the tie closed.

### TapTransmissionLine([element], PosAlongLine, NewBusNumber, ShuntModel, TreatAsMSLine);

Use this action to tap a transmission line by adding in a new bus and splitting the line in two.

Element : A description of the branch being tapped. Enclose description in brackets  
[BRANCH busnum1 busnum2 ckt]  
[BRANCH "name\_kv1" "name\_kv2" ckt]  
[BRANCH "buslabel1" "buslabel2" ckt]  
[BRANCH "label"]  
PosAlongLine : The percent distance along the branch at which the line will be tapped.  
NewBusNumber : The number of the new bus created at the tap point.  
ShuntModel : How should the shunt charging capacitance values be handled for the split lines.  
LINESHUNTS – Line shunts will be create (keeps exact power flow model).  
CAPACITANCE – Convert shunt values capacitance in the PI model.  
TreatAsMSLine : YES – the two new lines created will be made part of a multi-section line or.  
NO – the two lines will not be made multi-section lines.

## Run Mode Actions

The following script commands are available during Run Mode.

```

Animate (DoAnimate);
CalculatePTDF ([transactor seller], [transactor buyer], LinearMethod);
              [AREA num] [AREA num] AC
              [ZONE num] [ZONE num] DC
              [SUPERAREA "name"] [SUPERAREA name] DCPS
              [INJECTIONGROUP "name"] [INJECTIONGROUP name]
              [BUS num] [BUS num]
              [SLACK] [SLACK]

CalculatePTDFMultipleDirections (StoreValuesForBranches,StoreValuesForInterfaces,LinearMethod);
                                YES YES AC
                                NO NO DC
                                                                DCPS

CalculateLODF ([BRANCH nearbusnum farbusnum ckt], LinearMethod);
CalculateLODFMatrix(WhichOnes, filterProcess, filterMonitor, MonitorOnlyClosed,LinearMethod);
                   OUTAGES ALL ALL YES DC
                   CLOSURES SELECTED SELECTED NO DCPS
                   AREAZONE AREAZONE
                   "filtername" "filtername"
                               SAME

CalculateTLR ([flowelement], direction, [transactor], LinearMethod);
              [INTERFACE "name"] BUYER same as above for PTDFs
              [BRANCH nearbusnum farbusnum ckt] SELLER

CalculateTLRMultipleElement( TypeElement, WhichElement, direction, [transactor],LinearMethod);
                             BRANCH SELECTED BUYER same as above for PTDFs
                             INTERFACE OVERLOAD SELLER
                             BOTH CTGOVERLOAD

CalculateVoltSense ([BUS num]);

CalculateFlowSense ([flowelement], FlowType);
                   [INTERFACE "name"] MW
                   [BRANCH busnum1 busnum2 ckt] MVAR
                                                MVA

CalculateLossSense (FunctionType);
                   NONE
                   ISLAND
                   AREA
                   AREASA
                   SELECTED

CalculateVoltToTransferSense
([transactor seller], [transactor buyer], TransferType, TurnOffAVR);
  same as above for PTDFs [P, Q or PQ] YES
                                                                NO

CalculateVoltSelfSense (filter);
                       unspecified
                       "filtername"
                       SELECTED
                       AREAZONE

SetSensitivitiesAtOutOfServiceToClosest;
ZeroOutMismatches;

```

### Animate(DoAnimate);

Use this action to animate all the open online diagrams.

DoAnimate : Set to YES or NO. YES means to start the animation of the open online diagrams, while NO means that the animation will be paused.

### **CalculatePTDF([transactor seller], [transactor buyer], LinearMethod);**

Use this action to calculate the PTDF values between a seller and a buyer. You may optionally specify the linear calculation method. Note that the buyer and seller must not be same thing. If no Linear Method is specified, Lossless DC will be used.

- [transactor seller] : The seller (or source) of power. There are six possible settings:
  - [AREA num], [AREA "name"], [AREA "label"]
  - [ZONE num], [ZONE "name"], [ZONE "label"]
  - [SUPERAREA "name"], [SUPERAREA "label"]
  - [INJECTIONGROUP "name"], [INJECTIONGROUP "label"]
  - [BUS num], [BUS "name\_nomkv"], [BUS "label"]
  - [SLACK]
- [transactor buyer] : The buyer (or sink) of power. There are six possible settings which are the same as for the seller.
- LinearMethod : The linear method to be used for the PTDF calculation. The options are:
  - AC – for calculation including losses
  - DC – for lossless DC
  - DCPC – for lossless DC that takes into account phase shifter operation

### **CalculatePTDFMultipleDirections(StoreForBranches, StoreForInterfaces, LinearMethod);**

Use this action to calculate the PTDF values between all the directions specified in the case. You may optionally specify the linear calculation method. If no Linear Method is specified, Lossless DC will be used.

- StoreForBranches : Specify YES to store the values calculated for each branch.
- StoreForInterfaces : Specify YES to store the values calculated for each interface.
- LinearMethod : the linear method to be used for the PTDF calculation. The options are:
  - AC – for calculation including losses.
  - DC – for lossless DC.
  - DCPC – for lossless DC that takes into account phase shifter operation.

### **CalculateLODF([BRANCH nearbusnum farbusnum ckt], LinearMethod);**

Use this action to calculate the Line Outage Distribution Factors (or the Line Closure Distribution Factors) for a particular branch. If the branch is presently closed, then the LODF values will be calculated, otherwise the LCDF values will be calculated. You may optionally specify the linear calculation method as well. If no Linear Method is specified, Lossless DC will be used.

- [BRANCH nearbusnum farbusnum ckt]: the branch whose status is being changed. Can also use strings
  - [BRANCH "nearbusname\_kv" "farbusname\_kv" ckt]
  - [BRANCH "nearbuslabel" "farbuslabel" ckt]
  - [BRANCH "label"]
- LinearMethod : The linear method to be used for the LODF calculation. The options are:
  - DC – for lossless DC.
  - DCPC – for lossless DC that takes into account phase shifter operation.Note: AC is NOT an option for the LODF calculation.

### **CalculateLODFMatrix(WhichOnes, filterProcess, filterMonitor, MonitorOnlyClosed, LinearMethod);**

Use this action to calculate the Line Outage Distribution Factors (or the Line Closure Distribution Factors) for a particular branch. If the branch is presently closed, then the LODF values will be calculated, otherwise the LCDF values will be calculated. You may optionally specify the linear calculation method as well. If no Linear Method is specified, Lossless DC will be used.

- WhichOnes : Specify the type of sensitivities to be calculated.  
OUTAGES – Outage sensitivities will be calculated for those branches meeting the filterProcess.  
CLOSURES – Closure sensitivities will be calculated for those branches meeting the filterProcess.
- filterProcess : Specify a filter for the branches for which the outages or closures will be implemented.  
ALL – All AC transmission lines.  
SELECTED – Only those branches whose Selected field is YES.  
AREAZONE – Only those branches meeting the area/zone filter.  
"filename" – See the Using Filters in Script Commands section for more information on specifying the filename.
- filterMonitor : Specify a filter for the branches for which the impact of the outages or closures will be determined.  
ALL – All AC transmission lines.  
SELECTED – Only those branches whose Selected field is YES.  
AREAZONE – Only those branches meeting the area/zone filter.  
"filename" – See the Using Filters in Script Commands section for more information on specifying the filename.  
SAME – Same as set of branches to process as specified by filterProcess.
- MonitorOnlyClosed : Set to YES to monitor only those branches that are closed. Set to NO to monitor branches regardless of their status.
- LinearMethod : The linear method to be used for the LODF calculation.  
DC – for lossless DC.  
DCPC – for lossless DC that takes into account phase shifter operation.  
Note: AC is NOT an option for the LODF calculation.

### **CalculateTLR([flow element], direction, [transactor], LinearMethod);**

Use this action to calculate the TLR values a particular flow element (transmission line or interface). You also specify one end of the potential transfer direction. You may optionally specify the linear calculation method. If no Linear Method is specified, Lossless DC will be used.

- [flow element] : This is the flow element we are interested in. Choices are:  
[INTERFACE "name"]  
[INTERFACE "label"]  
[BRANCH nearbusnum farbusnum ckt]  
[BRANCH "nearbusname\_kv" "farbusname\_kv" ckt]  
[BRANCH "nearbuslabel" "farbuslabel" ckt]  
[BRANCH "label"]
- direction : The type of the transactor. Either BUYER or SELLER.
- [transactor buyer] : The transactor of power. There are six possible settings:  
AREA num], [AREA "name"], [AREA "label"]  
[ZONE num], [ZONE "name"], [ZONE "label"]  
[SUPERAREA "name"], [SUPERAREA "label"]  
[INJECTIONGROUP "name"], [INJECTIONGROUP "label"]  
[BUS num], [BUS "name\_nomkv"], [BUS "label"]  
[SLACK]
- LinearMethod : The linear method to be used for the calculation. The options are:  
AC – for calculation including losses  
DC – for lossless DC  
DCPC – for lossless DC that takes into account phase shifter operation



### **CalculateTLRMultipleElement(TypeElement,WhichElement,direction,[transactor],LinearMethod);**

Use this action to calculate the TLR values a multiple elements similar to as is done on the TLR multiple elements dialog. You also specify one end of the potential transfer direction. You may optionally specify the linear calculation method. If no Linear Method is specified, Lossless DC will be used.

- TypeElement : May be either INTERFACE, BRANCH, or BOTH  
WhichElement : There are three choices which represent which elements of the TypeElement specified will have TLR calculations performed.
- SELECTED : Only branches or interfaces with their Selected Field = YES will be used.
  - OVERLOAD : Only branches that are presently overloaded using their normal ratings will be used
  - CTGOVERLOAD : You must have first run the contingency analysis. A branch or interface is included in the calculation if it has been overloaded during at least one contingency.
- Direction : the type of the transactor. Either BUYER or SELLER.  
[transactor buyer] : the transactor of power. There are six possible settings.  
[AREA num], [AREA "name"], [AREA "label"]  
[ZONE num], [ZONE "name"], [ZONE "label"]  
[SUPERAREA "name"], [SUPERAREA "label"]  
[INJECTIONGROUP "name"], [INJECTIONGROUP "label"]  
[BUS num], [BUS "name\_nomkv"], [BUS "label"]  
[SLACK]
- LinearMethod: The linear method to be used for the calculation. The options are:  
AC: for calculation including losses.  
DC: for lossless DC.  
DCPC: for lossless DC that takes into account phase shifter operation.

### **CalculateVoltSense([BUS num]);**

This calculates the sensitivity of a particular buses voltage to real and reactive power injections at all buses in the system. (Note: this assumes that the power is injected at a given bus and taken out at the slack bus).

- [BUS num] : the bus to calculate sensitivities for.

### **CalculateFlowSense([flow element], FlowType);**

This calculates the sensitivity of the MW, MVAR, or MVA flow of a line or interface to an real and reactive power injections at all buses in the system. (Note: this assumes that the power is injected at a given bus and taken out at the slack bus).

- [flow element] : This is the flow element we are interested in. Choices are:  
[INTERFACE "name"]  
[INTERFACE "label"]  
[BRANCH busnum1 bus num2 ckt]  
[BRANCH "name\_kv1" "name\_kv2" ckt]  
[BRANCH "buslabel1" "buslabel2" ckt]  
[BRANCH "label"]
- FlowType : The type of flow to calculate this for. Either MW, MVAR, or MVA.

### **CalculateLossSense(FunctionType);**

This calculates the loss sensitivity at each bus for an injection of power at the bus. The parameter FunctionType determines which losses are referenced.

- FunctionType : This is the losses for which sensitivities are calculated.
- NONE : all loss sensitivities will be set to zero
  - ISLAND : all loss sensitivities are referenced to the total loss in the island
  - AREA : For each bus it calculates how the losses in the bus' area will change (Note: this means that sensitivities at buses in two different areas cannot be directly compared because they are referenced to different losses)
  - AREASA : same as Each Area, but if a Super Area exists it will use this instead (Note: this means that sensitivities at buses in two different areas cannot be directly compared because they are referenced to different losses)
  - SELECTED : Calculates how the losses in the areas selected on the Loss Sensitivity Form will change

### **CalculateVoltToTransferSense([transactor seller], [transactor buyer], TransferType, TurnOffAVR);**

This calculates the sensitivity of bus voltage to a real or reactive power transfer between a seller and a buyer. The sensitivity is calculated for all buses in the system.

- [transactor seller] : This is the seller (or source) of power. There are six possible settings:  
[AREA num], [AREA "name"], [AREA "label"]  
[ZONE num], [ZONE "name"], [ZONE "label"]  
[SUPERAREA "name"], [SUPERAREA "label"]  
[INJECTIONGROUP "name"], [INJECTIONGROUP "label"]  
[BUS num], [BUS "name\_nomkv"], [BUS "label"]  
[SLACK]
- [transactor buyer] : This is the buyer (or sink) of power. There are six possible settings, which are the same as for the seller.
- TransferType : The type of power transfer. The options are:  
P : real power transfer  
Q : reactive power transfer  
PQ : both real and reactive power transfer. (Note: Real and reactive power transfers are calculated independently, but both are calculated.)
- TurnOffAVR : Set to YES or NO. Set to YES to turn off AVR control for generators participating in the transfer. Set to NO to leave the AVR control unchanged for generators participating in the transfer.

### **CalculateVoltSelfSense(filter);**

This calculates the sensitivity of a particular bus' voltage to real and reactive power injections at the same bus. (Note: This assumes that the power is injected at a given bus and taken out at the slack bus.)

- filter : There are four options for the filter:  
CalculateVoltSelfSense; – No filter specified means that sensitivities will be calculated for all buses in the system.  
CalculateVoltSelfSense("filename"); – See the Using Filters in Script Commands section for more information on specifying the filename.  
CalculateVoltSelfSense(AREAZONE); – AREAZONE means to set generators that meet the area/zone filters.  
CalculateVoltSelfSense(SELECTED); – means to set gens if Selected=YES

### **SetSensitivitiesAtOutOfServiceToClosest;**

This will take the P Sensitivity and Q Sensitivity values calculated using CalculateTLR, CalculateFlowSense, or CalculateVoltSense actions and then populate the respective values at out-of-service buses so that they are equal to the value at the closest in service bus. The "distance" to the in-service buses will be measured by the number of nodes. If an out-of-service bus is equally close to a set of buses, then the average of that set of buses will be used.

### **ZeroOutMismatches;**

With this script command, the bus shunts are changed at each bus that has a mismatch greater than the MVA convergence tolerance so that the mismatch at that bus is forced to zero.

## Power Flow Related Actions

```
DiffFlowClearBase;
DiffFlowKeyType (KeyType);
                PRIMARY
                SECONDARY
                LABEL
DiffFlowMode (diffmode);
             PRESENT
             BASE
             DIFFERENCE

DiffFlowSetAsBase;
DiffFlowRefresh;
DoCTGAction ([contingency action]);
ResetToFlatStart (FlatVoltagesAngles, ShuntsToMax, LTCsToMiddle, PSAnglesToMiddle);
                TRUE YES YES YES
                FALSE NO NO NO
SolvePowerFlow (SolMethod, "filename1", "filename2", CreateIfNotFound1, CreateIfNotFound2);
                RECTNEWTON "file.aux" "file.aux" YES YES
                POLARNEWTON STOP STOP NO NO
                GAUSSSEIDEL
                FASTDECOUPLED
                DC
                ROBUST
```

### DiffFlowClearBase;

Call this action to clear the base case for the difference flows abilities of Simulator.

### DiffFlowKeyType(KeyType);

Use this action to change the key type that should be used when comparing fields when using the difference flows abilities of Simulator.

KeyType : String that starts with 'P' changes key field type to PRIMARY.  
String that starts with 'S' changes key field type to SECONDARY.  
String that starts with 'L' changes key field type to LABEL.

### DiffFlowMode(diffmode);

Call this action to change the mode for the difference flows abilities of Simulator.

diffmode : String that starts with 'P' changes it to PRESENT.  
String that starts with 'B' changes it to BASE.  
String that starts with 'D' changes it to DIFFERENCE.

### DiffFlowSetAsBase;

Call this action to set the present case as the base case for the difference flows abilities of Simulator.

### DiffFlowRefresh;

Call this action to refresh the linking between the base case and the present case. This should be used before saving data that identifies objects as being added or removed, especially if any topological differences have been made that affect the comparison.

### DoCTGAction([contingency action]);

Call this action to use the formats seen in the CTGElement subdata record for Contingency Data. Note that all actions are supported, except COMPENSATION sections are not allowed.

### **ResetToFlatStart (FlatVoltagesAngles, ShuntsToMax, LTCsToMiddle, PSAnglesToMiddle);**

Use this action to initialize the Power Flow Solution to a "flat start." The parameters are all optional and specify a conditional response depending on whether the solution is successfully found. If parameters are not passed then default values will be used.

- FlatVoltagesAngles : Set to YES or NO. YES means setting all the voltage magnitudes and generator setpoint voltages to 1.0 per unit and all the voltage angles to zero. Default Value = YES.
- ShuntsToMax : Set to YES or NO. YES means to increase Switched Shunts Mvar half way to maximum. Default Value = NO.
- LTCsToMiddle : Set to YES or NO. YES means setting the LTC Transformer Taps to middle of range. Default Value = NO.
- PSAnglesToMiddle : Set to YES or NO. YES means setting Phase Shifter angles to middle of range. Default Value = NO.

### **SolvePowerFlow (SolMethod, "filename1", "filename2", CreateIfNotFound1, CreateIfNotFound2);**

Call this action to perform a single power flow solution. The parameters are all optional and specify a conditional response depending on whether the solution is successfully found. If parameters are not passed then default values will be used.

- SolMethod : The solution method to be used for the Power Flow calculation. The options are:
  - RECTNEWT : for Rectangular Newton-Raphson.
  - POLARNEWTON : for Polar Newton-Raphson.
  - GAUSSSEIDEL : for Gauss-Seidel.
  - FASTDEC : for Fast Decoupled.
  - DC : for DC power flow calculationDefault Value = RECTNEWT.
- "filename1" : The filename of the auxiliary file to be loaded if there is a successful solution. You may also specify STOP, which means that all AUX file execution should stop under the condition. Default Value = "".
- "filename2" : The filename of the auxiliary file to be loaded if there is a NOT successful solution. You may also specify STOP, which means that all AUX file execution should stop under the condition. Default Value = "".
- CreateIfNotFound1 : Set to YES or NO. YES means that objects which cannot be found will be created while reading in DATA sections of filename1. Default Value = NO.
- CreateIfNotFound2 : Set to YES or NO. YES means that objects which cannot be found will be created while reading in DATA sections of filename2. Default Value = NO.

## Contingency Related Actions

```

CTGSolveAll (DoDistributed);
CTGSolve ("ContingencyName");
CTGSetAsReference;
CTGRestoreReference;
CTGWriteResultsAndOptions ("filename");
CTGProduceReport ("filename");
CTGAutoInsert;
CTGConvertAllToDeviceCTG (KeepOriginalIfEmpty);
CTGCalculateOTDF ([transactor seller], [transactor buyer], LinearMethod);
[AREA num] [AREA num] AC
[ZONE num] [ZONE num] AC
[SUPERAREA "name"] [AUPERAREA "name"] DCPS
[INJECTIONGROUP "name"] [INJECTIONGROUP "name"] AC
[BUS num] [BUS num] AC
[SLACK] [SLACK]
CTGWriteFilePTI ("filename", BusFormat, TruncateCTGLabels);
CTGJoinActiveCTGs (InsertSolvePowerFlow, DeleteExisting, JoinWithSelf, "filename");

```

### CTGSolveAll(DoDistributed);

Call this action to solve all the contingencies which are not marked skip. The DoDistributed is a YES/NO field that can be used if the distributed contingency analysis add-on is installed. If the DoDistributed flag is set to YES, then the distributed methods will be used to solve the contingency analysis. Note that the distributed analysis requires the proper configuration and security settings to work.

### CTGSolve("ContingencyName");

Call this action solve a particular contingency. The contingency is denoted by the "Contingency Name".

### CTGSetAsReference;

Call this action to set the present system state as the reference for contingency analysis.

### CTGRestoreReference;

Call this action to reset the system state to the reference state for contingency analysis.

### CTGWriteResultsAndOptions("filename", [opt1, opt2, opt3, ..., opt9], KeyField, UseDATASection);

Writes out all information related to contingency analysis as an auxiliary file. This includes Contingency Definitions, Limit Monitoring Settings, Contingency Results, Solution Options, CTG Options as well as any Model Criteria that are used by the Contingency Definitions.

Each entry in the Option Settings parameter, [opt1, opt2, ..., opt9], is either a YES or NO entry corresponding to the following options. If not specified or blank, the default entry given for each will be used.

- Opt1 – Save unlinked contingency actions, default = NO
- Opt2 – Save Contingency Options, default = YES
- Opt3 – Save Limit Monitoring Settings, default = NO
- Opt4 – Save General Power Flow Solution Options, default = YES
- Opt5 – Save List Display Settings, default = NO
- Opt6 – Save Contingency Results, default = YES
- Opt7 – Save Inactive Violations, default = YES
- Opt8 – Save Interface Definitions, default = NO
- Opt9 – Save Injection Group Definitions, default = NO
- Opt10 – Suppress Gen and Load options when writing out Options, default = NO

KeyField indicates the identifier that should be used for the data. Valid entries are PRIMARY, SECONDARY, or LABEL. The default setting is PRIMARY. PRIMARY will save using bus numbers and other primary key fields. SECONDARY will save using bus name and nominal kV and other secondary fields. LABEL will save using device labels. If no labels are specified then the primary key field will be used.

UseDATASection – set this to YES or NO. If YES, data that by default is specified using SUBDATA sections will instead be specified using DATA sections. For example, the actions that define a contingency by default are specified using a SUBDATA section. If choosing to use the DATA section instead, each action will be specified in a DATA record belonging to the ContingencyElement objecttype.

### **CTGProduceReport("filename");**

Produces a text-based contingency analysis report using the settings defined in CTG\_Options.

### **CTGAutoInsert;**

This action will auto insert contingencies for you case. Prior to calling this action, all options for this action must be specified in the Ctg\_AutoInsert\_Options object using SetData() or DATA sections.

### **CTGConvertAllToDeviceCTG(KeepOriginalIfEmpty);**

This command is intended for use with full topology models, where breakers and disconnects are defined in addition to generators, loads, transmission lines, and so on. This function would have no affect on a traditional planning model representation, which has no breakers or disconnects explicitly defined.

The purpose of the function is to allow the user to take a contingency set that is defined with outages of breakers and disconnects in a full topology model and convert them to outages of the traditional planning model elements, such as generators, loads, transmission lines, etc. This would be used in conjunction with the ability to save a full topology model as a consolidated model. A consolidated model reduces the full model down to a traditional planning model by examining the breaker and disconnect statuses, and reducing the system down by consolidating breakers and disconnects that are in service. The resulting model is a smaller model with the traditional planning elements represented, but breakers and disconnects have been removed and nodes aggregated into bus representations. This function will also take the breaker and disconnect statuses and convert contingencies defined with the breakers and disconnects and convert them into contingencies of the planning model devices affected by opening the original breakers and disconnects. Thus you could create a contingency set that is defined for the consolidated model, and can be run on the consolidated model with the same results as if the original contingency set is run on the full topology model.

The parameter *KeepOriginalIfEmpty* is a YES or NO option to retain or not the original contingency definitions for any contingencies that do not end up isolating any devices. This is an optional parameter that is NO by default if it is not specified.

Note that the contingency set generated depends on the statuses of the breakers and disconnects, and that the contingencies created will be different for different statuses of breakers and disconnects in the full topology model.

### **CTGCalculateOTDF([transactor seller], [transactor buyer], LinearMethod);**

This action first performs the same action as done by the CalculatePTDF([transactor seller], [transactor buyer], LinearMethod) call. It then goes through all the violations found by the contingency analysis tool and determines the OTDF values for the various contingency/violation pairs.

### **CTGWriteFilePTI("filename", BusFormat, TruncateCTGLabels);**

Write contingencies to a PTI format file.

"filename"	:	The name of the text file to write out.
BusFormat	:	Number – Identify buses using numbers Name8 -- Identify buses using BusName_NomkV strings truncated to 8 characters Name12 – Identify buses using BusName_NomkV strings truncated to 12 characters
TruncateCTGLabels	:	Set to YES or NO. YES means that the contingency labels will be truncated after 12 characters.

### **CTGJoinActiveCTGs(InsertSolvePowerFlow, DeleteExisting, JoinWithSelf, "filename");**

This command creates new contingencies that are a join of the current contingency list and a list read in from an auxiliary file or the current list itself. Contingencies with their Skip field set to YES will not be included in the join.

InsertSolvePowerFlow	:	Set to YES or NO. YES means to insert the solve power flow solution action between the joined contingency actions.
DeleteExisting	:	Set to YES or NO. YES means to delete the existing contingencies and only keep the joined contingencies.
JoinWithSelf	:	Set to YES or NO. YES means that the current contingency list will be joined with itself instead of contingencies specified in a file. If set to YES, the "filename" parameter does not have to be specified.
"filename"	:	Name of auxiliary file containing contingencies to join with the current contingency list. This does not have to be specified if JoinWithSelf = YES.

## Fault Related Actions

Fault	([BUS num],	faulttype,	R,	X);	
Fault	([BRANCH nearbusnum farbusnum ckt],	faultlocation,	faulttype,	R,	X);
		SLG			
		LL			
		3PB			
		DLG			

### Fault([Bus num, faulttype, R, X]);

### Fault([BRANCH nearbusnum farbusnum ckt], faultlocation, faulttype, R, X);

Call this function to calculate the fault currents for a fault. If the fault element is a bus then do not specify the fault location parameter. If the fault element is a branch, then the fault location is required.

- [Bus num] : This specifies the bus at which the fault occurs. You may also specify the bus using secondary keys or labels.  
[BUS "name\_nomkv"]  
[BUS "label"]
- [BRANCH nearbusnum farbusnum ckt] : This specifies the branch on which the fault occurs. You may also specify the branch using secondary keys or labels.  
[BRANCH "name\_kv1" "name\_kv2" ckt]  
[BRANCH "buslabel1" "buslabel2" ckt]  
[BRANCH "label"]
- Faultlocation : This specifies the percentage distance along the branch where the fault occurs. This percent varies from 0 (meaning at the nearbus) to 100 (meaning at the far bus)
- Faulttype : This specified the type of fault which occurs. There are four options:  
SLG : Single Line To Ground fault  
LL : Line to Line Fault  
3PB : Three Phase Balanced Fault  
DLG : Double Line to Group Fault.
- R, X : These parameters are optional and specify the fault impedance. If none are specified, then a fault impedance of zero is assumed.

## ATC (Available Transfer Capability) Related Actions

ATCDetermine	([transactor seller], [AREA num] [ZONE num] [SUPERAREA "name"] [INJECTIONGROUP "name"] [BUS num] [SLACK]	[transactor buyer], DoDistributed); [AREA num] YES [ZONE num] NO [SUPERAREA name] [INJECTIONGROUP name] [BUS num] [SLACK]
ATCRestoreInitialState;		
ATCIncreaseTransferBy	(amount);	
ATCTakeMeToScenario	(RL, G, I);	
ATCDetermineATCFor	(RL, G, I, ApplyTransfer);	YES NO
ATCWriteResultsAndOptions	("filename", AppendFile);	
ATCWriteToExcel	("worksheetname");	
ATCWriteToText	("filename", filetype);	
ATCSetAsReference;		

### ATCDetermine([transactor seller], [transactor buyer], ApplyTransfer, DoDistributed);

Use this action to calculate the Available Transfer Capability (ATC) between a seller and a buyer. Note that the buyer and seller must not be same thing. Other options regarding ATC calculations should be set using a DATA section via the ATC\_Options object type. If the distributed ATC add-on is installed then the optional DoDistributed flag may be set to indicate that the ATC should be solved using the distributed methods.

- [transactor seller] : The seller (or source) of power. There are six possible settings:  
[AREA num], [AREA "name"], [AREA "label"]  
[ZONE num], [ZONE "name"], [ZONE "label"]  
[SUPERAREA "name"], [SUPERAREA "label"]  
[INJECTIONGROUP "name"], [INJECTIONGROUP "label"]  
[BUS num], [BUS "name\_nomkv"], [BUS "label"]  
[SLACK]
- [transactor buyer] : The buyer (or sink) of power. There are six possible settings which are the same as for the seller.
- DoDistributed : Set to yes to use the distributed ATC solution method.

### ATCRestoreInitialState;

Call this action to restore the initial state for the ATC tool.

### ATCIncreaseTransferBy(amount);

Call this action to increase the transfer between the buyer and seller .

### ATCTakeMeToScenario(RL, G, I);

Call this action to set the present case according to Scenario RL, G, I.

### ATCDetermineATCFor(RL, G, I, ApplyTransfer);

Call this action to determine the ATC for Scenario RL, G, I.

- ApplyTransfer : Set this value to YES to leave the system state at the transfer level that was determined. When using the Iterated Linear then Full Contingency solution method, the system state will retain the transfer level but the contingency will not be applied.

### ATCWriteResultsAndOptions("filename", AppendFile);

Writes out all information related to ATC analysis as an auxiliary file. This includes Contingency Definitions, Limit Monitoring Settings, Solution Options, ATC Options, ATC results, as well as any Model Criteria that are used by the Contingency Definitions.

- "filename" : Name of auxiliary file.
- AppendFile : Optional parameter. YES means to append results to existing "filename." NO means to overwrite "filename" with the results. Default setting is YES.

### ATCWriteToExcel("worksheetname");

Sends ATC analysis results to an Excel spreadsheet. This script command is available only for Multiple Scenarios ATC analysis.

- "worksheetname" : The name of the Excel sheet where the results will be sent to.



**ATCWriteToText("filename", filetype);**

This is used with Multiple Scenario ATC analysis. Multiple files are created with "filename" as the primary identifier and the Interface scenario label appended to the end of the filename. Separate files are created for each of the Interface scenarios. Results inside the files are separated into sections based on the number of Rating/Load scenarios.

- "filename" : Primary identifier for the name of the file in which to save the results.  
"filename" gets appended with the Interface scenario label to complete the filename.
- filetype : Either TAB or CSV. This indicates the delimiter to use when writing out the file(s). This is an optional parameter with TAB being the default if omitted.

**ATCSetAsReference;**

Call this action to set the present system state to the reference state for ATC analysis.

## GIC (Geomagnetically Induced Current) Related Actions

GICCalculate	(MaxField, Direction, SolvePF);
GICClear;	
GICTimeVaryingCalculate	(TheTime, SolvePF);
GICTimeVaryingAddNewTime	(NewTime);
GICTimeVaryingDeleteAllTimes;	
GICWriteOptions	("FileName", KeyField);

### **GICCalculate(MaxField, Direction, SolvePF);**

Calculates the "Single Snapshot" using GICSolution Options

MaxField	: Maximum Electric Field in Volts/km
Direction	: Storm Direction, Degrees from 0 to 360
SolvePF	: Select YES or NO to include GIC in the Power Flow

### **GICClear;**

Clear GIC Values

### **GICTimeVaryingCalculate(TheTime,SolvePF);**

Calculates "Time Varying Input" GIC at specified time

TheTime	: Specified Time Point
SolvePF	: Select YES or NO to include GIC in the Power Flow

### **GICTimeVaryingAddNewTime(NewTime);**

Adds a new input values at specified time

NewTime	: New Time for new input values
---------	---------------------------------

### **GICTimeVaryingDeleteAllTimes;**

Delete All Input Time values

### **GICWriteOptions("FileName", KeyField);**

Calculates the "Single Snapshot" using GICSolution Options

FileName	: Name of Aux file name to write out the options
KeyField	: KeyField indicates the identifier that should be used for the data. Valid entries are PRIMARY, SECONDARY, or LABEL. The default setting is PRIMARY. PRIMARY will save using bus numbers and other primary key fields. SECONDARY will save using bus name and nominal kV and other secondary fields. LABEL will save using device labels. If no labels are specified then the primary key field will be used.

## ITP (Integrated Topology Processing) Related Actions

CloseWithBreakers	(objecttype, "filtername" or [object identifier], OnlyEnergizeSpecifiedObjects);
SaveConsolidatedCase	("filename", filetype, [BusFormat, TruncateCtgLabels, AddCommentsForObjectLabels]);

### CloseWithBreakers(objecttype, "filtername" or [object identifier], OnlyEnergizeSpecifiedObjects);

This action is used to specify which objects are to be energized by closing breakers and to actually close those breakers. The status of an object will be set to closed if necessary in addition to closing the breakers. If only the status of an object needs to be changed to close an object, that will occur without requiring any breakers to be closed.

objecttype	: Objects that are valid to be energize. Only allowed for Generators, Loads, Transmission Lines, Switched Shunts, DC Lines, Injection Groups, and Interfaces.
"filtername"	: The second parameter can either be the name of an advanced filter or an object identifier. When using an advanced filter, see the Using Filters in Script Commands section for more information on specifying the filtername.
[object identifier]	: The second parameter can either be the name of an advanced filter or an object identifier. When using an object identifier, the objecttype is applicable and no further specification of the type needs to be included with the object identifier as is done with some other script commands. The following describe the possible objecttypes and identifier options:
GEN:	[busnum id] ["name_nomkv" id] ["buslabel" id] ["label"]
LOAD:	[busnum id] ["name_nomkv" id] ["buslabel"] ["label"]
BRANCH:	[busnum1 busnum2 ckt] ["name_kv1" "name_kv2" ckt] ["buslabel1" "buslabel2" ckt] ["label"]
SHUNT:	[busnum id] ["name_nomkv" id] ["buslabel" id] ["label"]
INJECTIONGROUP:	["name"]
INTERFACE:	["name"]
DCLINE:	[num rectnum invnum] [num "rectnam_nomkv" "invname_nomkv"] [num "rectlabel" "invlabel"] ["label"]

OnlyEnergizeSpecifiedObjects : optional parameter, default is NO.  
 YES – No extra objects in addition to those specified in the filter can be energized. Each object will be evaluated individually.  
 NO – Extra objects could be energized in addition to those specified if a group of breakers required to energize a specified object also causes other objects to be energized. All objects will be evaluated collectively for determining which objects can be energized, i.e. breakers that cause one object to be energized might also be needed for another object to be energized.

**SaveConsolidatedCase("filename", filetype, [BusFormat, TruncateCtgLabels, AddCommentsForObjectLabels]);**

This action saves the full topology model into a consolidated case.

- "filename" : The name of the consolidated case file to be saved.
- Filetype : Optional parameter to specify the type of the file to be saved. If omitted, the latest version of the PWB will be used.
  - PWB – save a pwb file with the most recent version
  - PWBX – save a pwb with version X
  - PWPTI – save file with most recent PTI file format. When contingencies are present, they are saved to a .CON file.
  - PWPTIXX – save the file with PTI version XX, where XX is between 23 and 32
- BusFormat : optional parameter used to specify the bus identifier format in the .CON file used to store contingencies when saving a PTI file
  - Number – identify buses using number
  - Name8 – identify buses using the Name\_kV identifier truncated to 8 characters
  - Name12 – identify buses using the Name\_kV identifier truncated to 12 characters
- TruncateCTGLabels : optional parameter used to specify if the contingency labels should be truncated to 12 characters when saving the contingencies in PTI format
  - YES – truncate the contingency labels to 12 characters
  - NO – do not truncate the contingency labels
- AddCommentsForObjectLabels : (optional) YES adds object labels to the end of data records when saving a RAW file. (default NO)

## OPF (Optimal Power Flow) and SCOPF Related Actions

```

SolvePrimalLP
    ("filename1", "filename2", CreateIfNotFound1, CreateIfNotFound2);
    "file.aux"    "file.aux"    YES        YES
    STOP         STOP         NO         NO
InitializePrimalLP
    ("filename1", "filename2", CreateIfNotFound1, CreateIfNotFound2);
    "file.aux"    "file.aux"    YES        YES
    STOP         STOP         NO         NO
SolveSinglePrimalLPOuterLoop
    ("filename1", "filename2", CreateIfNotFound1, CreateIfNotFound2);
    "file.aux"    "file.aux"    YES        YES
    STOP         STOP         NO         NO
SolveFullSCOPF
    (BCMetho, "filename1", "filename2", CreateIfNotFound1, CreateIfNotFound2);
    POWERFLOW "file.aux"    "file.aux"    YES        YES
    OPF       STOP         STOP         NO         NO
OPFWriteResultsAndOptions ("filename");

```

### **SolvePrimalLP("filename1", "filename2", CreateIfNotFound1, CreateIfNotFound2);**

Call this action to perform a primal LP OPF solution. The parameters are all optional and specify a conditional response depending on whether the solution is successfully found. If parameters are not passed then default values will be used.

- "filename1" : The filename of the auxiliary file to be loaded if there is a successful solution. You may also specify STOP, which means that all AUX file execution should stop under the condition. Default Value = "".
- "filename2" : The filename of the auxiliary file to be loaded if there is a NOT successful solution. You may also specify STOP, which means that all AUX file execution should stop under the condition. Default Value = "".
- CreateIfNotFound1 : Set to YES or NO. YES means that objects which cannot be found will be created while reading in DATA sections of filename1. Default Value = NO.
- CreateIfNotFound2 : Set to YES or NO. YES means that objects which cannot be found will be created while reading in DATA sections of filename2. Default Value = NO.

### **InitializeLP("filename1", "filename2", CreateIfNotFound1, CreateIfNotFound2);**

This commands clears all the structures and results of previous primal LP OPF solutions. The parameters are all optional and specify a conditional response depending on whether the solution is successfully found. If parameters are not passed then default values will be used.

- "filename1" : The filename of the auxiliary file to be loaded if there is a successful solution. You may also specify STOP, which means that all AUX file execution should stop under the condition. Default Value = "".
- "filename2" : The filename of the auxiliary file to be loaded if there is a NOT successful solution. You may also specify STOP, which means that all AUX file execution should stop under the condition. Default Value = "".
- CreateIfNotFound1 : Set to YES or NO. YES means that objects which cannot be found will be created while reading in DATA sections of filename1. Default Value = NO.
- CreateIfNotFound2 : Set to YES or NO. YES means that objects which cannot be found will be created while reading in DATA sections of filename2. Default Value = NO.

### **SolveSinglePrimalLPOuterLoop("filename1", "filename2", CreateIfNotFound1, CreateIfNotFound2);**

This action is basically identical to the SolvePrimalLP action, except that this will only perform a single optimization. The SolvePrimalLP will iterate between solving the power flow and an optimization until this iteration converges. This action will only solve the optimization routine once, then resolve the power flow once and then stop.

**SolveFullSCOPF (BCMethod, "filename1", "filename2", CreateIfNotFound1, CreateIfNotFound2);**

Call this action to perform a full Security Constrained OPF solution. The parameters are all optional and specify a conditional response depending on whether the solution is successfully found. If parameters are not passed then default values will be used.

- BCMethod : The solution method to be used for solving the base case. The options are:  
POWERFLOW – for single power flow algorithm.  
OPF – for the optimal power flow algorithm.  
Default Value = POWERFLOW.
- "filename1" : The filename of the auxiliary file to be loaded if there is a successful solution. You may also specify STOP, which means that all AUX file execution should stop under the condition. Default Value = "".
- "filename2" : The filename of the auxiliary file to be loaded if there is a NOT successful solution. You may also specify STOP, which means that all AUX file execution should stop under the condition. Default Value = "".
- CreateIfNotFound1 : Set to YES or NO. YES means that objects which cannot be found will be created while reading in DATA sections of filename1. Default Value = NO.
- CreateIfNotFound2 : Set to YES or NO. YES means that objects which cannot be found will be created while reading in DATA sections of filename2. Default Value = NO.

**OPFWriteResultsAndOptions("filename");**

Writes out all information related to OPF analysis as an auxiliary file. This includes Limit Monitoring Settings, options for Areas, Buses, Branches, Interfaces, Generators, SuperAreas, OPF Solution Options.

## PV Related Actions

```
PVSetSourceAndSink      ([elementSource],      [elementSink]);
                        [INJECTIONGROUP "name"] [INJECTIONGROUP "name"]

PVRun;
PVRun                  ([elementSource],      [elementSink]);
                        [INJECTIONGROUP "name"] [INJECTIONGROUP "name"]

PVClear;
PVStartOver;
PVDestroy;

PVWriteResultsAndOptions ("filename",      AppendFile);

RefineModel            (objecttype,      filter,      Action,      Tolerance);
                        AREA      unspecified      TRANSFORMERTAPS      value
                        ZONE      "filtername"      SHUNTS
                                      OFFFAVR

PVQVTrackSingleBusPerSuperBus;
```

Changes were made with Simulator version 14 to eliminate the need for a PV study name. To maintain functionality with any existing processes that users might have in place using older script definitions, scripts from older versions of Simulator will still be supported if the name is specified. However, the name will just be ignored. The script formats given here reflect the changes for versions 14 and later.

The PVCreate script required in previous versions is no longer necessary starting with Simulator version 14. Versions starting with 14 will still recognize this action if is included and will simply set the source and sink for the study. This does the same thing as PVSetSourceAndSink.

It is highly recommended that for any new processes the new script formats specified here be used.

### **PVSetSourceAndSink([elementSource], [elementSink]);**

Call the function to specify the source and sink elements to perform the PV study.

[elementSource] : The source of power for the PV study. There is only one possible setting:  
[INJECTIONGROUP "name"] or [INJECTIONGROUP "label"]  
[elementSink] : The sink of power for the PV study. There is only one possible setting, which is the same as for the source.

### **PVRun;**

Call the function to start the PV study.

### **PVRun([elementSource], [elementSink]);**

Call the function to specify the source and sink elements and start the PV study.

[elementSource] : The source of power for the PV study. There is only one possible setting:  
[INJECTIONGROUP "name"] or [INJECTIONGROUP "label"]  
[elementSink] : The sink of power for the PV study. There is only one possible setting, which is the same as for the source.

### **PVClear;**

Call the function to clear all the results of the PV study.

### **PVStartOver;**

Call the function to start over the PV study. This includes clear the activity log, clear results, restore the initial state, set the current state as initial state, and initialize the step size.

### **PVDestroy;**

Call the function to destroy the PV study. This will remove all results and prevent any restoration of the initial state that is stored with the PV study.

### **PVWriteResultsAndOptions("filename", AppendFile);**

Call this action to save all the PV results and options in the auxiliary file "filename".

[AppendFile]: YES/NO optional parameter. Specifying NO will cause it to overwrite the existing file. Specifying YES will cause it to append to the file. If not specified, then YES is assumed.

### **RefineModel(objecttype, filter, Action, Tolerance);**

Call this function to refine the system model to fix modeling idiosyncrasies that cause premature loss of convergence during PV and QV studies.

- Objecttype : The objecttype being selected.  
AREA  
ZONE
- Filter : There are two options for the filter:  
RefineModel(..., "", ...);  
No filter specified means to select all objects of this type.  
RefineModel(..., "filtername", ...);  
See the Using Filters in Script Commands section for more information on specifying the filtername.
- Action : The way the model will be refined. Choices are:
  - TRANSFORMERTAPS : Fix all transformer taps at their present values if their  $V_{max} - V_{min}$  is less than or equal to the user specified tolerance.
  - SHUNTS : Fix all shunts at their present values if their  $V_{max} - V_{min}$  is less than or equal to the user specified tolerance.
  - OFFAVR : Remove units from AVR control, thus locking their MVAR output at its present value if their  $Q_{max} - Q_{min}$  is less or equal to the user specified tolerance.
- Tolerance : Tolerance value.

### **PVQVTrackSingleBusPerSuperBus;**

If the topology processing add-on is installed, then this script command can be used to reduce the number of monitored buses. The script action examines each monitored value for each bus and determines if that bus is part of a super bus and selects monitored buses so that only the pnode is monitored.



## QV Related Actions

```
QVRun      ("filename", InErrorMakeBaseSolvable);
           YES
           NO
           NOTE: The QV study is always performed on selected buses.

QVWriteResultsAndOptions ("filename");

RefineModel (objecttype, filter, Action, Tolerance);
           AREA unspecified TRANSFORMERTAPS value
           ZONE "filtername" SHUNTS
           OFFAVR

QVSelectSingleBusPerSuperBus;
```

### **QVRun("filename", InErrorMakeBaseSolvable);**

Call the function to start a QV study for the list of buses whose SELECTED? field is set to YES.

"filename" : This specifies the file to which to save a comma-delimited version of the results.

InErrorMakeBaseSolvable: This specifies whether to perform a solvability analysis of the base case if the pre-contingency base case cannot be solved. If not specified, then YES is assumed.

### **QVWriteResultsAndOptions("filename");**

Call this action to save all the QV results and options in the auxiliary file "filename".

### **RefineModel(objecttype, filter, Action, Tolerance);**

Call this function to refine the system model to fix modeling idiosyncrasies that cause premature loss of convergence during PV and QV studies.

Objecttype : The objecttype being selected.

AREA

ZONE

Filter : There are three options for the filter:

RefineModel(..., "", ...); – No filter specified means to select all objects of this type.

RefineModel(..., "filtername", ...); – See the Using Filters in Script Commands section for more information on specifying the filtername.

Action : The way the model will be refined. Choices are:

TRANSFORMERTAPS : Fix all transformer taps at their present values if their Vmax – Vmin is less than or equal to the user specified tolerance.

SHUNTS : Fix all shunts at their present values if their Vmax – Vmin is less than or equal to the user specified tolerance.

OFFAVR : Remove units from AVR control, thus locking their MVAR output at its present value if their Qmax – Qmin is less or equal to the user specified tolerance.

Tolerance : Tolerance value.

### **QVSelectSingleBusPerSuperBus;**

If the QV tool is being used on a full topology model, this action can be used to modify the monitored buses. This action examines the monitored buses and sets the monitored status so that only one bus is monitored for each pnode.

## TS (Transient Stability) Related Actions

```
TSSolveAll;  
TSSolve          (ContingencyName, [StartTime, StopTime, StepSize]);  
TSWriteOptions   ("FileName", [SaveDynamicModel, SaveStabilityOptions,  
                  SaveStabilityEvents, SaveResultsSettings, SavePlotDefinitions], KeyField);  
TSSaveTwoBusEquivalent ("AuxFileName", [BUS]);  
TSCalculateSMIBEigenValues;  
TSLoadGE         ("FileName", GENCCYN);  
TSLoadPTI        ("FileName", "MCREfilename", "MTRLOfilename", "GNETfilename",  
                  "BASEGENfilename");  
TSLoadBPA        ("FileName");  
TSGetResults     ("FileName", Single/Separate, [contingencies], [plots], StartTime,  
                  EndTime);  
TSAutoCorrect;
```

### **TSSolveAll;**

Solves all defined transient contingencies which are not set to skip.

### **TSSolve("ContingencyName", [StartTime, StopTime, StepSize]);**

Solves only the specified contingency.

ContingencyName : The name of the contingency to solve  
StartTime : (optional) Start time in seconds  
StopTime : (optional) Stop time in seconds  
StepSize : (optional) Step size in seconds

### **TSWriteOptions("FileName", [SaveDynamicModel, SaveStabilityOptions, SaveStabilityEvents, SaveResultsEvents, SavePlotDefinitions], KeyField);**

Save the transient stability option settings to an auxiliary file.

FileName : Name and path of the file to save  
SaveDynamicModel : (optional) NO doesn't save dynamic model (default YES)  
SaveStabilityOptions : (optional) NO doesn't save stability options (default YES)  
SaveStabilityEvents : (optional) NO doesn't save stability events (default YES)  
SaveResultsSettings : (optional) NO doesn't save results settings (default YES)  
SavePlotDefinitions : (optional) NO doesn't save plot definitions (default YES)  
KeyField : (optional) Specifies key: can be Primary, Secondary, or Label (default Primary)

### **TSSaveTwoBusEquivalent("AuxFileName", [BUS]);**

Save the two bus equivalent model of a specified bus to a PWB file. Initialization to the start time is always done before saving the two bus equivalent.

AuxFileName : Name and path for the output AUX file  
BUS : Bus can be specified in three ways:  
Number : [BUS busnum]  
Name/NomkV : [BUS "busname\_nominalKV"]  
Label : [BUS "buslabel"]

### **TSCalculateSMIBEigenValues;**

Calculate single machine infinite bus eigenvalues. Initialization to the start time is always done before calculating eigenvalues.

### **TSLoadGE("FileName", GENCCYN);**

Loads transient stability data stored in the GE DYD format.

FileName : Name of the DYD file to load  
GENCCYN : YES to split combined cycle units, NO to leave them alone

### **TSLoadPTI("FileName", "MCREfilename", "MTRLOfilename", "GNETfilename", "BASEGENfilename");**

Loads transient stability data in the PTI format.

FileName : Name of the DYR file to load  
MCREfilename : (optional) If not loading a MCRE file, specify ""  
MTRLOfilename : (optional) If not loading a MTRLO file, specify ""  
GNETfilename : (optional) If not loading a GNET file, specify ""  
BASEGENfilename : (optional) If not loading a BASEGEN file, specify ""

**TSLoadBPA("FileName");**

Loads transient stability data stored in the BPA format.

FileName : Name of the BPA file to load

**TSGetResults("FileName", Single/Separate, [Contingencies], [Plots, ObjectFields], StartTime, EndTime);**

Use this to save out results for specific variables from plots, subplots, and object/field pairs after a transient stability simulation has been run. If StartTime and StopTime are not specified, results for the entire simulation time are obtained.

FileName : Name of the CSV result file to write out

Single/Separate : Determines whether the results are all saved in one file (Single) with name "filename\_Results.csv" or whether results for each transient contingency is saved in a separate file (Separate) with name "filename\_ctgname.csv." A separate header file is also saved out, with a name of "filename\_Header.csv".

Contingencies : A list of contingency names for which to save out results

Plots, ObjectFields : A list of plots and object/field pairs to save out for the specified contingencies

StartTime : Start of the window of simulation time from which the results are to be retrieved

EndTime : End of the window of simulation time from which the results are to be retrieved

**TSAutoCorrect;**

Runs the auto correction of parameters for a transient stability run. If there are still validation errors after running this script that would prevent the stability simulation from running, then the remainder of a script will be aborted.

## DATA Section

```
DATA DataName(object_type, [list_of_fields], file_type_specifier, create_if_not_found)
{
data_list_1
.
.
.
data_list_n
}
```

Immediately following the DATA keyword, you may optionally include a DataName. By including the DataName, you can make use of the script command LoadData("filename", DataName) to call this particular data section from another auxiliary file. Following the optional DataName is the argument list.

### DATA Argument List

The DATA argument list identifies what the information section contains. A left and right parenthesis "(" ")" mark the beginning and end of the argument list.

The file\_type\_specifier parameter distinguishes the information section as containing custom auxiliary data (as opposed to Simulator's native auxiliary formats), and indicates the format of the data. Currently, the parser recognizes two values for file\_type\_specifier:

(blank) or AUXDEF or DEF	Data fields are space delimited
AUXCSV or CSV or CSVAUX	Data fields are comma delimited

The object\_type parameter identifies the type of object or data element the information section describes or models. For example, if object\_type equals BUS, then the data describes BUS objects.

There are some special object types that start with the keyword REMOVED. If these are loaded into Simulator while in Edit mode, the corresponding objects will be deleted. For example REMOVEDBUS will delete BUS objects, REMOVEDBRANCH will delete BRANCH objects, etc. Not all object types have a corresponding REMOVED object type, and simply prepending this keyword to the front of an object\_type will not allow this functionality. The objects that exist of this with this functionality are the ones that allow comparison of topological changes through the Difference Flows tool.

The list of object types Simulator's auxiliary file parser can recognize will grow as new applications are developed. Within Simulator, you will always be able to obtain a list of the available object types by going to the main menu and choosing Window, Export Case Object Fields, and then exporting the objects to Excel or a text file.

The create\_if\_not\_found field is optional. This specifies whether or not to create a new object if an existing one is not found. If the value is YES, objects will be created. If the value is NO, objects will not be created. If omitted, the default behavior of prompting the user about whether or not to create a non-existing object will continue. If loading an auxiliary file using the LoadAux script command, the create\_if\_not\_found field for the data section will override the CreateIfNotFound field with the script.

The `list_of_fields` parameter lists the types of values the ensuing records in the data section contain. The order in which the fields are listed in `list_of_fields` dictates the order in which the fields will be read from the file. Simulator currently recognizes many different field types, each identified by a specific field name. Because the available fields for an object may grow as new applications are developed for the convenience of our customers, you will always be able to obtain a list of the available object types and fields by going to the main menu and choosing Window, Export Object Fields, and then choosing to export to Excel or a text file. Certainly, only a subset of these fields would be found in a typical custom auxiliary file. In crafting applications to export custom auxiliary files, developers need concern themselves only the fields they need to communicate between their applications and Simulator. A few points of interest regarding the `list_of_fields` are:

- The `list_of_fields` may take up several lines of the text file.
- The `list_of_fields` should be encompassed by braces [ ].
- When encountering the PowerWorld comment string `'/'` in one of these lines of the text file, all text to the right is ignored.
- Blank lines, or lines whose first characters are `'/'` will be ignored as comments.
- Field names must be separated by commas.

Example:

```
DATA (BUS, [BusNomKV, Bus, // comment here
  ABCPhaseAngle:1, ABCPhaseAngle:2, ABCPhaseV, ABCPhaseV:1,
  // comments allowed here to

  // note that blank rows are ignored
  AreaNum, BusAngle, BusB, BusCat, BusEquiv, BusG,
  BusGenericSensV, BusKVVolt, BusLambda, BusLoadMVA, // more comment
  BusLoadMW, BusLongName])
```

## Field Naming

When listing fields, some field names may be augmented with a field location. These are in the format `variablename:location`. One example of this is the field `LineMW`. For a branch, there are two MW flows associated with the line: one MW flow at the from bus, and one MW flow at the to bus. So that the number of fields does not become huge, the same field name is used for both of these values. For the from bus flow, we write `LineMW:0`, and for the to bus flow, we write `LineMW:1`. Note that field names using a location of 0, such as `LineMW:0`, may simply leave off the `:0`.

There are several fields that can be referred to by the user-defined name for the field rather than using the location number. These are fields that might have their location numbers change when different auxiliary files are merged in the same case. Referring to these by name can eliminate this possible confusion. These fields can be defined in the format `variablename:location by name`. They can also be referred to by location number as well.

Fields that allow referring to the location by name are:

- Expressions – `"CustomExpression:my expression name"`
- Custom fields (Floating Point, Integer, and String) – `"CustomSingle:my custom single name"`. Using this format for custom fields requires that Custom Field Descriptions be created for the fields to be used.
- Calculated Fields – `"BGCalcField:my calculated field name"`

## Key Fields

Simulator uses certain fields to identify the specific object being described. These fields are called key fields. For example, the key field for BUS objects is `BusNum`, because a bus can be identified uniquely by its number. The key fields for GEN objects are `BusNum` and `GenID`. To properly identify each object, the object's key fields must be present. They can appear in any order in the `list_of_fields` (i.e. they need not be the first fields listed in `list_of_fields`). As long as the key fields are present, Simulator can identify the specific object. By going to the main menu and choosing Window and then Export Case Object Fields you will obtain a list of fields available for each object type in either Excel or text format. In this output, the key fields will appear with asterisks `*`.

## Data List

After the data argument list is completed, the Data list is given. The data section lists the values of the fields for each object in the order specified in `list_of_fields`. The data section begins with a left curly brace and ends with the a right curly brace. A few points of interest regarding the `value_list`:

- The `value_list` may take up several lines of the text file.
- Each new data object must start on its own line of text.
- When encountering the PowerWorld comment string `'/'` in one of these lines of the text file, all text to the right of this is ignored.
- Blank lines, or lines whose first characters are `'/'` will be ignored as comments.
- Remember that the right curly brace must appear on its own line at the end of the `data_list`.
- If the `file_type_specifier` is CSV, the values should be separated by commas. Otherwise, separate the field names using spaces.
- Strings can be enclosed in double quotes, but this is not required. You should however always inclose strings that contain spaces (or commas) in quotes. Otherwise, strings containing commas would cause errors for comma-delimited files, and spaces would cause errors for space-delimited formatted files.
- A string in the format `&[ModelExpressionName]:[#digits]:[#decimals]` will be treated as though the value of the named model expression is entered in the field, with `[#digits]` total and `[#decimals]` digits to the right of the decimal point. This format may be used in case information displays as well as in AUX files.

## Using Labels for Identification

Most data objects (such as buses, generators, loads, switched shunts, transmission lines, areas, zones, and interfaces) may have an alternative names assigned to them. These alternative names are called labels. Labels allow you to refer to equipment in the model in a way that may be unique to your organization. Labels may thus help clarify which elements are described by a particular set of data, especially when the short names employed by the power system model prove cryptic. Furthermore, since labels are likely to change less frequently than bus numbers, and since a label must, by definition, identify only one power system component, they may function as an immutable key for importing data from auxiliary files into different cases, even when bus numbering schemes change between the cases. Labels must be unique for devices of the same type, but the same label can be used for a device of a different type.

Information dialogs corresponding to buses, generators, loads, switched shunts, transmission lines, areas, zones, and interfaces feature a button called Labels. If you press this button, the device's Label Manager Dialog will appear. The Label Manager Dialog lists the labels associated with the device. You can delete a label from the list by selecting it and pressing the delete key on the keyboard or clicking the Delete button. You may add a label to the device by typing its name in the textbox and pressing the Add New button. You will not be allowed to add a Label that already exists for the same type of device. A single power system device may have multiple labels, but each label may be associated with only one device of a given type. For example, a bus could have the label Bus North while a generator could also have the same label, but there could not be another bus or generator with this same label.

You also may designate a particular label to be the primary label for the device by checking the box Primary before adding the label. Alternatively, you can select the device from the list and click the Make Primary button. A device's primary label is the one that is listed first in the Labels (All) field (`variablename = LabelsAll`) in a Case Information Display. This field lists all labels assigned to a device as a comma-delimited string. Any label can be used to import data from auxiliary data files.

Labels can be used to map data from an auxiliary data file to a power system device. Recall that auxiliary data files require you to include a device's key fields in each data record so that data may be mapped to the device. Labels provide an alternative key. Instead of supplying the bus number to identify a bus, for example, you can supply one of the bus's labels. The label will enable Simulator to associate the data with the device associated with that label. This mechanism performs most efficiently when the primary label is used, but other labels will also provide the mapping mechanism. The Label (for use in input from AUX or Paste) field (`variablename = Label`) is used for importing data using labels and is blank when viewing in a case information display. Keep in mind that all devices read via an auxiliary file using the label field should have a non-blank label. Otherwise, information for that device will not be read. Even if the primary or secondary key fields are provided with the device, as long as the label field is present, that is the only field that will be used to identify the device. New devices cannot be created by simply identifying them by label. Either the primary or secondary key fields must be present to create a new device and the label field should not be present.

Again, it is important to remember this: a single power system device may have multiple labels, but each label may be associated with only one device of a particular type. This is the key to enabling data to be imported from an auxiliary file using labels.

## Saving Auxiliary Files Using Labels

All devices that can be identified by labels will have the Labels (All) and Label (for use in input from AUX or Paste) fields available in their case information displays. In order to save auxiliary files that identify devices by label, the two label fields should be added to the case information display prior to saving the data in an auxiliary file. Because the Label (for use in input from AUX or Paste) field will be blank when saved in the auxiliary file, this field must be populated with one of the labels in the Labels (All) field before loading the auxiliary file back in. Keep in mind that devices with blank labels cannot be identified when loading in an auxiliary file, so avoid saving auxiliary files by label if all devices do not have labels. [Note that when saving out an entire case as an auxiliary file, the field "AllLabels" is included for each object type that allows labels and has some labels defined.](#)

Many devices require SUBDATA sections. These sections have custom formats specific to the type of information that they contain. When saving auxiliary files with devices that require SUBDATA sections, the user can choose to use primary or secondary key fields or labels to identify devices in the SUBDATA sections. The user will either be prompted when saving the devices, or there is an option to change the key field to use when saving subdata sections on the PowerWorld Simulator Options dialog under the Case Information Displays category. When choosing to use labels, if a device has a label, it will be used. If it is a device that can be identified by buses and bus labels exist, bus labels will be used. Finally, if the device does not have a label and the buses do not have labels, the primary key for the device will be used for identification.

Devices that have SUBDATA sections that contain other devices that can be identified by labels include: contingencies, interfaces, injection groups, post power flow solution actions, and owners.

The setting to choose which identifier to use for the SUBDATA sections does not just apply to SUBDATA sections. Often when saving groups of options, this setting will apply to everything being saved with those options and not just the SUBDATA sections. This includes contingency options, ATC options, limit monitoring settings, and PVQV options. In these cases, there will be a prompt asking the user to decide which identifier to use in the auxiliary file.

## Loading Auxiliary Files SUBDATA Sections Using Labels

The various SUBDATA sections that represent references to other objects can also be read using labels. Examples include contingencies, interfaces, injection groups, post power flow solution actions, and owners. When reading a SUBDATA section such as this, PowerWorld makes no assumption ahead of time about what identification was used to write this SUBDATA section. Instead, an order of precedence for the identification is as follows

	Identification	Explanation	Example
1	Key Fields	assumes that the strings represent Key Fields	BRANCH 8 9 1
2	Secondary Key Fields	assumes that the strings represent Secondary Key Fields	BRANCH Eight_138 Nine_230 1
3	Labels for component objects	the key/secondary key fields for some objects consist of references to other objects. An example of this is the BRANCH object that is described by the From Bus, To Bus, and Circuit ID. This assumes that labels of the component objects are used.	BRANCH Label8 Label9 1
4	Labels	Assumes that the string represents one of the Labels of the object	BRANCH LabelForBranch

## Special Use of Labels in SUBDATA

There are a few special cases where objects have fields that identify other devices. These devices can be identified by label but not in the conventional means because the label field applies to the object that contains the device and a SUBDATA section is not necessary. These special cases include: (Note all fields given below are by variable name because the use of labels is most relevant with auxiliary files.)

**ATC Scenarios:** ATC Scenario change records usually contain primary key fields to identify the devices that should be adjusted during the scenario. If using labels, these primary key fields will be replaced with a single Label field. The use of this field is different because the Label field refers to the device in the change record and not to the change record itself. When labels are used with ATC scenarios, device labels only can be used. Bus labels cannot be used to identify devices for which no label exists but a bus label does.

**ATC Extra Monitors:** ATC Extra Monitors identify either branches or interfaces to monitor during the ATC analysis. These devices are identified in the WhoAmI field of ATC Extra Monitor records. Usually, the WhoAmI field is a special format that

contains key field tags. Optionally, this field can use the label of the device for the extra monitor. If the device label is not available, the standard format will be used. There is no option to use bus labels if they exist and the device labels do not.

**Model Conditions:** Devices in Model Conditions are usually identified by the WhoAmI field which is in a special format that contains key field tags. Optionally, this field can use the label of the device. If the device label does not exist, the standard format will be used. There is no option to use bus labels if they exist and the device labels do not.

**Model Expressions:** Model Expressions contain Model Fields. Model Fields are identified by the WhoAmI fields in the Model Expressions. Usually, the WhoAmI fields are in a special format that contains key field tags. Optionally, these fields can use the label of the device associated with the Model Field. If the device does not exist, the standard format will be used. There is no option to use bus labels if they exist and the device labels do not.

**Bus Load Throw Over Records:** Bus Load Throw Over Records are used with contingency analysis. These records have an option to identify the bus to which the load will be transferred by either number or name\_kV combination. If choosing to identify objects by label, the BusName\_NomVolt:1 field will contain the label of the bus instead of the name\_kV combination. Bus Load Throw Over Records will be saved in an auxiliary file if choosing to Save settings on the Contingency Analysis dialog.

**Injection Group Participation Points:** All participation points and the injection groups to which they belong can be listed on the Injection Group Display. Load, generator, and shunt devices that can be assigned to a participation point must be identified by bus and ID. The bus can be identified by either the number or name. When identifying by name, the BusName\_NomVolt field is used to provide the name\_kV combination for the bus. If choosing to identify devices by label, this field instead will contain the label of the device. If the device does not have a label but the bus does, the bus label will be used instead in conjunction with the ID of the device. Even if the device does contain a label, the ID field must be included in any auxiliary file that is going to be loaded because it is a key field. Injection groups can be included in other injection groups. Injection groups can be identified by label, even though this is not a normal thing to do. If any injection groups have labels and these injection groups are included in other injection groups, their labels will also appear in the BusName\_NomVolt field. If they do not have labels, they will be identified by the injection group name that appears in the PPntID field.



## SubData Sections

The format described thus far works well for most kinds of data in Simulator. It does not work as well however for data that stores a list of objects. For example, a contingency stores some information about itself (such as its name), and then a list of contingency elements, and possibly a list of limit violations as well. For data such as this, Simulator allows `<SubData>`, `</SubData>` tags that store lists of information about a particular object. This formatting looks like the following

```
DATA (object_type, [list_of_fields], file_type_specifier, create_if_not_found)
{
value_list_1
  <SUBDATA subobject_type1>
    precise format describing an object_type1
    precise format describing an object_type1
    .
    .
  </SUBDATA>
  <SUBDATA subobject_type2>
    precise format describing an object_type2
    precise format describing an object_type2
    .
    .
  </SUBDATA>
value_list_2
  .
  .
value_list_n
}
```

Note that the information contained inside the `<SubData>`, `</SubData>` tags may not be flexibly defined. It must be written in a precisely defined order that will be documented for each SubData type. The description of each of these SubData formats follows.

## ATC\_Options

### **RLScenarioName**

### **GScenarioName**

### **IScenarioName**

These three sections contain the pretty names of the RL Scenarios, G Scenarios, and I Scenarios. Each line consists of two values: Scenario Number and a name string enclosed in quotes.

Scenario Number : The scenarios are number 0 through the number of scenarios minus 1.  
Scenario Name : These represent the names of the various scenarios.

Example:

```
<SUBDATA RLScenarioName>
//Index Name
  0 "Scenario Name 0"
  1 "Scenario Name 1"
</SUBDATA>
```

### **ATCMemo**

This section contains the memo text for the ATC analysis.

Example:

```
<SUBDATA ATCMemo>
//Memo
"Comments for the ATC analysis"
</SUBDATA>
```

## ATCExtraMonitor

### **ATCFlowValue**

This subdata section contains a list of a flow values for specified transfer levels. Each line consists of two values: Flow Value (flow on the monitored element) and a Transfer Level (in MW).

Flow Value : Contains a string describing which monitor this belongs to.  
Transfer Level : Contains the value for this extra monitor at the last linear iteration.

Example:

```
<SUBDATA ATCFlowValue>
//MWFlow TransferLevel
  94.05 55.30
  105.18 80.58
  109.02 107.76
</SUBDATA>
```

## ATCScenario

### **TransferLimiter**

This subdata section contains a list of the TransferLimiters for this scenario. Each line contains fields relating one of the Transferlimiters. The fields are written out in the following order:

Limiting Element : Contains a description of the limiting element. The possible values are:  
"PowerFlow Divergence"  
"AREA num"  
"SUPERAREA name"  
"ZONE num"  
"BRANCH num1 num2 ckt"  
"INJECTIONGROUP name"  
"INTERFACE name"

- Limiting Contingency : The name of the limiting contingency. If blank, then this means it's a limitation in the base case.
- MaxFlow : The transfer limitation in MW in per unit.
- PTDF : The PTDF on the limiting element in the base case (not in percent).
- OTDF : The OTDF on the limiting element under the limiting contingency.
- LimitUsed : The limit which was used to determine the MaxFlow in per unit.
- PreTransEst : The estimated flow on the line after the contingency but before the transfer in per unit.
- MaxFlowAtLastIteration : The total transfer at the last iteration in per unit.
- IterativelyFound : Either YES or NO depending on whether it was iteratively determined.

Example:

```
<SUBDATA TransferLimiter >
"BRANCH 40767 42103 1" "contin" 2.84 -0.0771 -0.3883 -4.35 -4.35 -0.01 "-55.88" YES
"BRANCH 42100 42321 1" "Contin" 4.42 0.1078 0.5466 6.50 5.64 1.57 " 22.59" NO
"BRANCH 42168 42174 1" "Contin" 7.45 -0.0131 -0.0651 -1.39 -1.09 4.60 "-33.31" NO
"BRANCH 42168 42170 1" "Contin" 8.54 0.0131 0.0651 1.39 1.02 5.69 " 26.10" NO
"BRANCH 41004 49963 1" "Contin" 9.17 -0.0500 -0.1940 -4.39 -3.16 6.32 " 68.73" NO
"BRANCH 46403 49963 1" "Contin" 9.53 0.0500 0.1940 4.46 3.16 6.68 "-68.68" NO
"BRANCH 42163 42170 1" "Contin" 10.14 -0.0131 -0.0651 -1.39 -0.92 7.29 "-15.58" NO
</SUBDATA>
```

### ATCEXtraMonitor

This subdata section contains a list of the ATCEXtraMonitors for this scenario. Each line contains three fields relating one of the ATCEXtraMonitors. The first field describes the ATCEXtraMonitor which this subdata corresponds to. The second and third variables are the initial value and sensitivity for this extra monitor for the scenario. An optional fourth field may be included if we are using one of the iterated ATC solution options. This field must be the String "ATCFlowValue".

- Monitor Description : Contains a string describing which monitor this belongs to.
- InitialValue : Contains the value for this extra monitor at the last linear iteration.
- Sensitivity : Contains the sensitivity of this monitor.
- ATCFlowValue : A string which signifies that a block will follow which stores a list of flow values for specified transfer levels. Each line of the block consists of two values: Flow Value (flow on the monitored element) and a Transfer Level (in MW). The block is terminated when a line of text that starts with 'END' is encountered.

Example:

```
<SUBDATA ATCEXtraMonitor>
"Interface<KEY1>Left-Right</KEY1>" 40.0735 0.633295
"Branch<KEY1>2</KEY1><KEY2>5</KEY2><KEY3>1</KEY3>" 78.7410 0.266589
</SUBDATA>
```

## AUXFileExportFormatData

### DataBlockDescription

This subdata section is used to define the objects that should be included in an auxiliary file along with their fields, subdata sections, and any filter used to specify which objects should be included. Each line contains the following:

- ObjectType : Name of the object to include in the auxiliary file.
- [FieldList] : List of fields to include. Must be enclosed in brackets. This list can either be space-delimited or comma-delimited.
- [SubdataList] : List of subdata sections to include. This list must be enclosed in brackets and can be either space-delimited or comma-delimited. Include empty brackets to not include subdata or for objects that do not have any subdata sections.
- "Filter" : Description of the filter to use for determining which objects to include. This must be enclosed in double quotes. If no filter is to be used, empty double quotes should be included. Valid entries are: "", "filename", "AREAZONE", and "SELECTED". See the Using Filters in Script Commands section for more information on specifying the filename.

Example:

```
<SUBDATA DataBlockDescription>
  // ObjectType [FieldList] [SubdataList] "Filter"
  Area [AreaName, AreaNum] [] "SELECTED"
  Gen [BusNum, BusName, GenID] [BidCurve, ReactiveCapability] ""
</SUBDATA>
```

## AUXFileExportFormatDisplay

### **DataBlockDescription**

Same format as for the AUXFileExportFormatData subdata section.

Example:

```
<SUBDATA DataBlockDescription>
  // ObjectType [FieldList] [SubdataList] "Filter"
  DisplayArea [AreaName, AreaNum, SOAuxiliaryID] [] ""
  DisplayTransmissionLine [BusNum, BusNum:1, LineCircuit, SOAuxiliaryID]
    [Line] "Nominal Voltage > 138 kV"
</SUBDATA>
```

## BGCalculatedField

### **Condition**

Calculated Fields allow you to define a calculation over most network and aggregation objects along with a few other types of objects. The calculation can then be used to show an aggregation calculation on objects that link to these calculation objects in some manner. Part of the definition is a filter which specifies which objects to operate over. This subdata section is identical to the Condition subdata section of the Filter object type.

## Bus

### **MWMarginalCostValues**

### **MvarMarginalCostValues**

### **LPOPFMarginalControls**

These three sections contain specific values computed for an OPF solution. In MWMarginalCostValues or MvarMarginalCostValues these specific values are the MW or Mvar marginal prices for each constraint. In LPOPFMarginalControls the values are the sensitivities of the controls with respect to the cost of each bus.

Example:

```
<SUBDATA MWMarginalCostValues>
  //Value
  16.53
  0.00
  21.80
</SUBDATA>
```

## BusViewFormOptions

### **BusViewBusField**

### **BusViewFarBusField**

### **BusViewGenField**

### **BusViewLineField**

### **BusViewLoadField**

### **BusViewShuntField**

The values represent specific fields on the custom defined bus view onelines. Each line contains two values:

- Location : The various locations on the customized bus view contain slots for fields. This is the slot number.
- FieldDescription : This is a string enclosed in double quotes. The string itself is delimited by the @ character. The string contains five values:
- Name of Field : The name of the field. Special fields that appear on dialog by default have special names. Otherwise these are the same as the fieldnames of the AUX file format (for the "other fields" feature on the dialogs).
  - Total Digit : Number of total digits for a numeric field.
  - Decimal Points : Number of decimal points for a numeric field.
  - Color : This is the color of the field. It is not presently used.
  - Increment Value: This is the "delta per mouse" click for the field.

Example:

```
<SUBDATA BusViewLineField>
  0 "MW Flow@6@1@0@0"
  1 "MVar Flow@6@1@0@0"
  2 "MVA Flow@6@1@0@0"
  3 "BusAngle:1@6@2@0@0"
</SUBDATA>
```

## ColorMap

### **ColorPoint**

A colorpoint is simply described by a real number (between 0 and 100) indicating the percentage breakpoint, an integer describing the color, and a field indicating if the color should be used or the contour should be transparent. These three values are written on a single line of text. Each line contains two values:

- cmvalue : Real number between 0 and 100 (minimum to maximum value).
- cmcolor : Integer between 0 and 16,777,216. Value is determined by taking the red, green and blue components of the color and assigning them a value between 0 and 255. The color is then equal to  $red + 256 * green + 256 * 256 * blue$ .
- cmalpha : Integer between 0 and 255, where only 0 and 255 are valid values. A value of 0 indicates that the color point is transparent, while a value of 255 indicates that the color point is opaque. If the alpha channel is omitted, a default value of 255 (opaque) will be assigned.

Example:

```
<SUBDATA ColorPoint>
// Value Color Alpha
100.0000 127 255
 62.5000 65535 255
 50.0000 8388479 0
 12.5000 16711680 0
  0.0000 8323072 255
</SUBDATA>
```

## Contingency

### CTGElementAppend

Normally when reading in contingency definitions, the CTGElement SubData section is used to define the list of elements. When reading a CTGElement SubData section, all existing elements of the contingency are deleted and replaced with the ones read from the file. Using the CTGElementAppend as the SubData section will modify this behavior so that the elements are appended to the existing ones instead of deleted.

### CTGElement

A contingency element is described by up to four entries. All entries must be on a single line of text:

```
"Action" "ModelCriteria" Status //comment
```

Action : String describing the action associated with this element. See below for actions available.

ModelCriteria : This is the name of a ModelFilter or ModelCondition under which this action should be performed. This entry is optional. If it is not specified, then a blank (or no criteria) is assumed. If you want to enter a Status, then you must specify "" as the ModelCriteria.

Status : Four options  
CHECK – perform action if ModelCriteria is true  
ALWAYS – perform action regardless of ModelCriteria  
NEVER – do not perform action  
POSTCHECK – perform action if Model Criteria is true following implementation of other actions and solving the power flow  
This entry is optional. If it is not specified, then CHECK is assumed.

Comment : All text to the right of the comment symbol (//) will be saved with the CTGElement as a comment.

Possible Actions:

#### Transmission Line or Transformer outage or insertion

BRANCH	bus1# bus2# ckt	OPEN
		CLOSE
		OPENCBS
		CLOSECBS

Takes branch out of service, or puts it in service. Note: bus# values may be replaced by a string enclosed in single quotes where the string is the name of the bus followed by an underscore character and then the nominal voltage of the bus. These values may also be replaced by a string enclosed in single quotes which represents the label of the bus. Also, the entire sequence [bus1# bus2# ckt] may be replaced by the label of the branch.

#### Generator, Load, or Switched Shunt outage or insertion

GEN	bus# id	OPEN
LOAD	bus# id	CLOSE
SHUNT	bus# id	OPENCBS
INJECTIONGROUP	name	CLOSECBS

Takes a generator, load, or shunt out of service, or puts it in service. If specifying an injection group, the status of all devices in the injection group will be changed. Note: bus# values may be replaced by a string enclosed in single quotes where the string is the name of the bus followed by an underscore character and then the nominal voltage of the bus. These values may also be replaced by a string enclosed in single quotes which represents the label of the bus. Also, the sequence [bus1# ckt] or [name] may be replaced by the label of the device.

#### Generator, Load or Switched Shunt movement to another bus

GEN	bus1#	MOVE_PQ_TO	bus2#	value	MW
LOAD	bus1# id	MOVE_P_TO			MVAR
SHUNT		MOVE_Q_TO			PERCENT

Use to move generation, load or shunt at a bus1 over to bus2. This can be used on a bus or specific device basis in specifying what to move. Note: bus# values may be replaced by a string enclosed in single quotes where the string is the name of the bus followed by an underscore character and then the nominal voltage of the bus. These values may also be replaced by a string enclosed in single quotes which represents the label of the bus. When identifying specific devices, the device label can replace the bus number and device id.

### Generator, Load or Switched Shunt set or change a specific value

GEN	bus#	SET_P_TO	value	MW
LOAD	bus# id	SET_Q_TO		MVAR
SHUNT		SET_PQ_TO		PERCENT
		SET_VOLT_TO		
		CHANGE_P_BY		
		CHANGE_Q_BY		
		CHANGE_PQ_BY		
		CHANGE_VOLT_BY		

Use to set the generation, load, or shunt at a bus to a particular value. This can also be used on a specific device. This can also be used to change by a specified amount. The Voltage setpoints only apply to SHUNTS and GENs. Note: bus# values may be replaced by a string enclosed in single quotes where the string is the name of the bus followed by an underscore character and then the nominal voltage of the bus. These values may also be replaced by a string enclosed in single quotes which represents the label of the bus. When identifying specific devices, the device label can replace the bus number and device id.

### Bus outage causes all lines connected to the bus to be outage

BUS	bus#	OPEN
		OPENCBS

Takes all branches connected to the bus out of service. Also outages all generation, load, or shunts attached to the bus. Note: bus# values may be replaced by a string enclosed in single quotes where the string is the name of the bus followed by an underscore character and then the nominal voltage of the bus. These values may also be replaced by a string enclosed in single quotes which represents the label of the bus.

### Interface outage or insertion

INTERFACE	name	OPEN
		CLOSE
		OPENCBS
		CLOSECBS

Takes all monitored branches in the interface out of service, or puts them all in service. Note: the [name] may be replaced by the label of the interface.

### Injection Group change specific value

INJECTIONGROUP	name	CHANGE_P_TO	value	MW
		SET_P_TO		PERCENT
				MWMERITORDER
				PERCENTMERITORDER
				MWMERITORDEROPEN
				PERCENTMERITORDEROPEN

Use to set or change the MW generation/load in an injection group by a particular value. Note that MW and PERCENT OPTIONS will change each point in the injection group by a value in proportion to the participations factors of the group.

MWMERITORDER and PERCENTMERITORDER will modify only generator points in the injection group. Generators will be adjusted in order of highest participation factor to lowest with each generator in the list being adjusted until it hits either its maximum or minimum MW limit before moving to the next generator. This process continues until the desired injection is met. Generators will not be opened in the process so all generators will continue to provide Mvar support.

MWMERITORDEROPEN and PERCENTMERITORDEROPEN will modify only generator points in the injection group. When chosen and the change in the MW output is *negative*, the merit order dispatch will be modified. In this case, the generator in the injection group with the highest participation factor will have its status changed to Open, followed by the second generator and so on. This will continue until the amount of MW opened is as close to the desired amount without exceeding the desired amount of drop. If a generator would cause the desired gen drop amount to be exceeded, that generator is skipped and the next generator in merit order is processed. If the change requested is *positive*, generators will be increased toward their maximum MW output in the same manner as though this open were not chosen.

Note: the [name] may be replaced by the label of the injection group.

### Series Capacitor Bypass or Inservice

SERIESCAP	bus1# bus2# ckt	BYPASS
		INSERVICE

Bypasses a series capacitor, or puts it in service. Note: bus# values may be replaced by a string enclosed in single quotes where the string is the name of the bus followed by and underscore

character and then the nominal voltage of the bus. Note: bus# values may also be replaced by a string enclosed in single quotes which represents the label of the bus. Also, the entire sequence [bus1# bus2# ckt] may be replaced by the label of the branch.

#### Series Capacitor set impedance

SERIESCAP	bus1# bus2# ckt	SET_X_TO	value	PERCENT PU
-----------	-----------------	----------	-------	---------------

Changes the impedance a series capacitor either specifying a new per unit value or specifying a percentage of the value in the contingency reference case. Note: bus# values may be replaced by a string enclosed in single quotes where the string is the name of the bus followed by and underscore character and then the nominal voltage of the bus. Note: bus# values may also be replaced by a string enclosed in single quotes which represents the label of the bus. Also, the entire sequence [bus1# bus2# ckt] may be replaced by the label of the branch.

#### DC Transmission Line outage

DCLINE	bus1# bus2# ckt	OPEN OPENCBS
--------	-----------------	-----------------

Takes DC Line out of service. Note: bus# values may be replaced by a string enclosed in single quotes where the string is the name of the bus followed by an underscore character and then the nominal voltage of the bus. These values may also be replaced by a string enclosed in single quotes which represents the label of the bus. Also, the entire sequence [bus1# bus2# ckt] may be replaced by the label of the dc transmission line.

#### DC Line set a specific value or insertion

DCLINE	bus1# bus2# ckt	SET_P_TO	value	MW
		CHANGE_P_BY		PERCENT
		SET_I_TO		AMPS
		CHANGE_I_BY		
		CLOSE		
		CLOSECBS		

Use to set the DC Line setpoint to a particular value, or puts it in service. Note: bus# values may be replaced by a string enclosed in single quotes where the string is the name of the bus followed by an underscore character and then the nominal voltage of the bus. Note: bus# values may also be replaced by a string enclosed in single quotes which represents the label of the bus. Also, the entire sequence [bus1# bus2# ckt] may be replaced by the label of the dc transmission line. (Note: for the CLOSE and CLOSECBS choice, only the units of MW or AMPS may be used.)

#### MTDC Converter outage

DCCONVERTER	rec# bus#	OPEN OPENCBS
-------------	-----------	-----------------

Takes multi-terminal DC converter out of service. The rec# specifies the multi-terminal DC line record, while bus# specifies the AC bus to which the converter is connected. Note: bus# values may be replaced by a string enclosed in single quotes where the string is the name of the bus followed by an underscore character and then the nominal voltage of the bus. These values may also be replaced by a string enclosed in single quotes which represents the label of the bus.

#### MTDC Converter set a specific value or insertion

DCCONVERTER	rec# bus#	SET_P_TO	value	MW
		CHANGE_P_BY		PERCENT
		SET_I_TO		AMPS
		CHANGE_I_BY		
		CLOSE		
		CLOSECBS		

Use to set the multi-terminal DC converter setpoint to a particular value, or puts it in service. The rec# specifies the multi-terminal DC line record, while bus# specifies the AC bus to which the converter is connected. Note: bus# values may be replaced by a string enclosed in single quotes where the string is the name of the bus followed by an underscore character and then the nominal voltage of the bus. Note: bus# values may also be replaced by a string enclosed in single quotes which represents the label of the bus. (Note: for the CLOSE and CLOSECBS choice, only the units of MW or AMPS may be used.)

#### Phase Shifter set a specific value

PHASESHIFTER	bus1# bus2# ckt	SET_P_TO	value	MW
		CHANGE_P_BY		PERCENT

Use to set the phase shift angle to a particular value. Note: bus# values may be replaced by a string enclosed in single quotes where the string is the name of the bus followed by an underscore character and then the nominal voltage of the bus. These values may also be replaced by a string



enclosed in single quotes which represents the label of the bus. Also, the entire sequence [bus1# bus2# ckt] may be replaced by the label of the branch.

### 3-Winding Transformer outage or insertion

```
3WXFORMER | bus1# bus2# bus3# ckt | OPEN
           |                               | CLOSE
           |                               | OPENCBS
           |                               | CLOSECBS
```

Takes all three windings of a 3-winding transformer out of service, or puts them in service. Note: bus# values may be replaced by a string enclosed in single quotes where the string is the name of the bus followed by an underscore character and then the nominal voltage of the bus. Note: bus# values may also be replaced by a string enclosed in single quotes which represents the label of the bus. Also, the entire sequence [bus1# bus2# bus3# ckt] may be replaced by the label of the three winding transformer.

### Area Control Type Chage

```
AREA | area# | SET_TO | 'OFF'
      |      |          | 'PARTFAC'
      |      |          | 'AREASLACK bus#'
      |      |          | 'IGSLACK injectiongroup name'
```

Specify to change the make-up power for an area so that it is different during a contingency than the area control settings used in the reference case. The area may be set to toggle the control setting to OFF, PARTFAC, AREASLACK, and IGSLACK. The Area Control topic provides more information about these control types. If selecting Area Slack is chosen, then a bus must be specified which will act as the area slack during the contingency action. If selecting IG Slack, then an injection group must be specified by name.

Note: bus# values may be replaced by a string where the string is the name of the bus followed by an underscore character and then the nominal voltage of the bus. Note: bus# values may also be replaced by a string which represents the label of the bus.

In order for the Area contingency action to work correctly, there are contingency and power flow solution options that must be set correctly. Simulator does not automatically set these options so the user must make sure they are set.

- Area control must be enabled in the contingency base case, i.e. the Power Flow Solution Option for Island-Based AGC must be set to Disable (Use the Area and Super Area Dispatch settings).
- The contingency Make-Up Power option must be set to Same as Power Flow case.
- The option to Disable Automatic Generation Control (AGC) found with the Power Flow Solution Options must NOT be selected.

Another suggestion, although not a strict requirement, is that the area should be on area control prior to contingency analysis if a control type other than Off AGC is going to be set during a contingency. If a large ACE exists in the base case with area control off, switching the area on control during the contingency will zero out the ACE in addition to compensating for required make-up power.

### Execute a Power Flow Solution

```
| SOLVEPOWERFLOW
```

Include this action to cause the solution of the contingency to be split into pieces. Actions that are listed before each SOLVEPOWERFLOW call will be performed as a group.

### Calling of a name ContingencyBlock

```
| CONTINGENCYBLOCK | name
```

Calls a ContingencyBlock and executes each of the actions in that block.

### Make-Up Power Compensation.

Only valid immediately following a SET, CHANGE, OPEN or CLOSE action on a Generator, Shunt or Load. This describes how the change in MW or MVAR are picked up by buses throughout the system. The values specify participation factors. Note: bus# values may be replaced by a string enclosed in single quotes where the string is the name of the bus followed by an underscore character and then the nominal voltage of the bus.

```
COMPENSATION
bus#1 value1
bus#2 value2
...
END
```

Example:

```
<SUBDATA CTGElement>
// just some comments
// action
"BRANCH 40821 40869 1 OPEN" "" Model Criteria Status comment
"GEN 45041 1 OPEN" "" ALWAYS //Raver - Paul 500 kV
"BRANCH 42702 42727 1 OPEN" "Line X Limited" CHECK //Open Fern Hill
"GEN 40221 1 OPEN" "Interface L1" CHECK //Drop ~600 MW
"GEN 40227 1 OPEN" "Interface L2" CHECK //Drop ~1200 MW
"GEN 40221 1 OPEN" "Interface L3" CHECK //Drop ~600 MW
</SUBDATA>
```

Note: ContingencyElement object types can also be directly created inside their own DATA section as well. One of the key fields of the object is then the name of the contingency to which the ContingencyElement belongs.

## LimitViol

A LimitViol is used to describe the results of a contingency analysis run. Each Limit Violation lists eight values:

- ViolType** : One of six values describing the type of violation.
- BAMP – branch amp limit violation
  - BMVA – branch MVA limit violation
  - VLOW – bus low voltage limit violation
  - VHIGH – bus high voltage limit violation
  - INTER – interface MW limit violation
  - CUSTOM – Custom Monitor value
- ViolElement** : This field depends on the ViolType.
- for VLOW, VHIGH – "bus1#" or "busname\_buskv" or "buslabel"
  - for INTER – "interfacename" or "interfacelabel"
  - for BAMP, BMVA – "bus1# bus2# ckt violationbus# MWFlowDirection"
- violationbus# is the bus number for the end of the branch which is violated  
MWFlowDirection is the direction of the MW flow on the line. Potential values are "FROMTO" or "TOFROM".
- Note: each bus# may be replaced with the name underscore nominal kV string enclosed in single quotations. Or bus# values may be replaced by a string enclosed in single quotes representing the label of the bus. Also the entire sequence [bus1# bus2# ckt] may be replaced by the label of the branch.
- for CUSTOM – "custommonitorname deviceidentifier" where the deviceidentifier will use the key fields or label as specified by the option selected when saving
- Limit** : This is the numerical limit which was violated.
- ViolValue** : This is the numerical value of the violation.
- PTDF** : This field is optional. It only makes sense for interface or branch violations. It stores a sensitivity of the flow on the violating element during in the base case with respect to a transfer direction This must be calculated using the Contingency Analysis Other Actions related to Sensitivities.
- OTDF** : Same as for the PTDF.
- InitialValue** : This stores a number. This stores the base case value for the element which is being violated. This is used to compare against when looking at change violations.
- Reason** : This will say whether this was a pure violation, or is being reported as a violation because the change from the base case is higher than a specified threshold.
- LIMIT – means this is a violation of a line/interface/bus limit or simply a Custom Monitor
  - CHANGE – means this is being reported as a limit because the change from the initial value is higher than allowed

Example:

```
<SUBDATA LimitViol>
BAMP "1 3 1 1 FROMTO" 271.94031 398.48096 10.0 15.01 //Note OTDF/PTDF
// values can also be specified with name underscore nominal kV string
// enclosed inside a single quote as shown next
BAMP "'One_138' 'Three_138' 1 1 FROMTO" 271.94031 398.48096 10.0 15.01
INTER "Right-Top" 45.00000 85.84451 None None 56.000 LIMIT
</SUBDATA>
```

Note: ViolationCTG object types can also be directly created inside their own DATA section as well. One of the key fields of the object is then the name of the contingency to which the ViolationCTG belongs.

## Sim\_Solution\_Options

These describe the power flow solution options which should be used under this particular contingency. The format of the subdata section is two lines of text. The first line is a list of the fieldtypes for Sim\_Solution\_Options which should be changed. The second line is a list of the values. Note that in general, power flow solution options are stored at three different locations in contingency analysis. When implementing a contingency, Simulator gives precedence to these three locations in the following order:

1. Contingency Record Options (stored with the particular contingency).
2. Contingency Tool Options (stored with CTG\_Options).
3. The global solution options.

## WhatOccurredDuringContingency

Each line of this subdata section is part of a text description of what actually ended up being implemented for this contingency. This will list which actions were executed and which actions ended up being skipped because of their model criteria. Each line of the subdata section must be enclosed in quotes.

```
<SUBDATA WhatOccurredDuringContingency>
  "Applied: "
  " OPEN Branch Two      (2) TO Five      (5) CKT 1 | | CHECK | | ELEMENT"
</SUBDATA>
```

## ContingencyMonitoringException

Each line of this subdata section contains a string identifying a specially handled monitored element for this contingency followed by a string indicating how this monitored element should be handled with this contingency. The elements can be identified by their primary or secondary key fields or by label. The element descriptions should be enclosed in quotes because they contain spaces.

```
<SUBDATA ContingencyMonitoringException>
  "Branch '2' '3' '1' " "Exclude"
  "Branch 'Three_138.00' 'Four_138.00' " "Include"
  "Branch 'Line_2_5' " "Default"
</SUBDATA>
```

## CTG\_Options

### Sim\_Solution\_Options

These describe the power flow solution options which should be used under this particular contingency. The format of the subdata section is two lines of text. The first line is a list of the fieldtypes for Sim\_Solution\_Options which should be changed. The second line is a list of the values. Note that in general, power flow solution options are stored at three different locations in contingency analysis. When implementing a contingency, Simulator gives precedence to these three locations in the following order:

1. Contingency Record Options (stored with the particular contingency).
2. Contingency Tool Options (stored with CTG\_Options).
3. The global solution options.

## CTGElementBlock

### CTGElement

This format is the same as for the Contingency objecttype, however, you cannot call a ContingencyBlock from within a contingencyblock.

### CTGElementAppend

When a subdata section is defined as **CTGElementAppend** rather than **CTGElement**, the actions of this subdata section will be appended to the contingency actions, instead of replacing them. This format is the same as for the Contingency objecttype, however, you cannot call a ContingencyBlock from within a contingencyblock.

Note: CTGElementBlockElement object types can also be directly created inside their own DATA section as well. One of the key fields of the object is then the name of the contingency block to which the CTGElementBlockElement belongs.

## CustomColors

### CustomColors

These describe the customized colors used in Simulator, which are specified by the user. A custom color is an integer describing a color. Each custom color is written on a single line of text and is an integer between 0 and 16,777,216. The value is determined by taking the red, green, and blue components of the color and assigning them a value between 0 and 255. The color is then equal to  $\text{red} + 256 * \text{green} + 256 * 256 * \text{blue}$ . Each line contains only one integer that corresponds to the color specified.

Example:

```
<SUBDATA CustomColors>
  9823301
  8613240
</SUBDATA>
```

## CustomCaseInfo

### ColumnInfo

Each line of this SUBDATA section can be used for specifying the column width of particular columns of the respective Custom Case Information Sheet. The line contains two values – the column and then a column width. This is shown in the following example.

Example:

```
<SUBDATA ColumnInfo>
  "SheetCol"      133
  "SheetCol:1"   150
  "SheetCol:2"   50
</SUBDATA>
```

## DataGrid

### ColumnInfo

Contains a description of the columns which are shown in the respective data grid. Each line of text contains four fields: VariableName, ColumnWidth, TotalDigits, DecimalPoints

VariableName : Contains the variable which is shown in this column.  
ColumnWidth : The column width.  
TotalDigits : The total digits displayed for numerical values.  
DecimalPoints : The decimal points shown for numerical values.

Example:

```
DATA (DataGrid, [DataGridName])
{
  BUS
  <SUBDATA COLUMNINFO>
    BusNomVolt 100 8 2
    AreaNum    50 8 2
    ZoneNum    50 8 2
  </SUBDATA>
  BRANCHRUN
  <subdata COLUMNINFO >
    BusNomVolt:0 100 8 2
    BusNomVolt:1 100 8 2
    LineMW:0     100 9 3
  </SUBDATA>
}
```

## DynamicFormatting

### DynamicFormattingContextObject

This subdata section contains a list of the display object types which are chosen to be selected. Each line of the section consists of the following:

DisplayObjectType (WhichFields) (ListOfFields)

- DisplayObjectType : The object type of the display object. These are generally the same as the values seen in the subdata section SelectByCriteriaSetType of SelectByCriteriaSet object types. The only exception is the string CaseInfo, which is used for formatting applying to the case information displays.
- (WhichFields) : For display objects that can reference different fields, this sets which of those fields it should select (e.g. select only Bus Name Fields). The value may be either ALL or SPECIFIED.
- (ListOfFields) : If WhichFields is set to SPECIFIED, then a delimited list of fields follows.

Example:

```
<SUBDATA DynamicFormattingContextObject>
  // Note: CaseInfo applies to case information displays
  CaseInfo "SPECIFIED" BusName
  DisplayAreaField "ALL"
  DisplayBus
  DisplayBusField "SPECIFIED" BusName BusPUVolt BusNum
  DisplayCircuitBreaker
  DisplaySubstation
  DisplaySubstationField "SPECIFIED" SubName SubNum BusNomVolt BGLoadMVR
  DisplayTransmissionLine
  DisplayTransmissionLineField "ALL"
</SUBDATA>
```

### LineThicknessLookupMap

### LineColorLookupMap

### FillColorLookupMap

### FontColorLookupMap

### FontSizeLookupMap

### BlinkColorLookupMap

### XoutColorLookupMap

### FlowColorLookupMap

### SecondaryFlowColorLookupMap

The values of the lookup table for the characteristics that can be modified by the dynamic formatting tool. The first line contains the two following fields:

- fieldname : It is the field that the lookup table is going to look for.
- usediscrete : Set to YES or NO. If set to YES, the characteristic values will be discrete, meaning that the characteristic value will correspond exactly to the one specified in the table. If set to NO, the characteristic values will be continuous, which means the characteristic value will be an interpolation of the high and low closest values specified in the table.

The following lines contain two fields:

- fieldvalue : The value for the field.
- characteristicvalue : The corresponding characteristic value for such field value.

Example:

```
<SUBDATA LineColorLookupMap>
  // FieldName UseDiscrete
  BusPUVolt YES
  // FieldValue Color
  1.02 16711808
  1.05 8454143
  1.1 16744703
</SUBDATA>
```

## Filter

### Condition

Conditions store the conditions of the filter. Each condition is described by one line of text which can contain up to five fields:

variablename	: It is one of the fields for the object_type specified. It may optional be followed by a colon and a non-negative integer. If not specified, 0 is assumed. Example: on a LINE, 0 = from bus, 1 = to bus Thus:                   sgLineMW:0 = the MW flow leaving the from bus SgLineMW:1 = the MW flow leaving the to bus Note: this value may also be the string "_UseAnotherfilter" which would then be followed by either meets or notmeets and then the name of another Filter.			
Condition	: Possible Values	Alternate1	Alternate2	Requirements
	between	><		requires othervalue
	notbetween	~><		requires othervalue
	equal	=	==	
	notequal	<>	~=	
	greaterthan	>		
	lessthan	<		
	greaterthanorequal	>=		
	lessthanorequal	<=		
	contains			
	notcontains			
	startswith			
	notstartswith			
	inrange			
	notinrange			
	meets			
	notmeets			
	isblank			
	notisblank			
value	: The value to compare to. For fields associated with strings, this must be a string. For fields associated with real numbers, this must be a number. For fields associated with integers, this is normally an integer, except when the Condition is "inrange" or "notinrange". In this case, value is a comma/dash separated number string.			
(othervalue)	: If required, the other value to compare to.			
(FieldOpt)	: Optional integer value with following meanings. 0 - strings are case insensitive, use number fields directly (0 is the default value if not otherwise specified) 1 - strings are case sensitive, take ABS of field values			

Example:

```
DATA (FILTER, [objecttype, filtername, filtertype, prefilter])
{
BUS "a bus filter" "AND" "no"
  <SUBDATA CONDITION>
    BusNomVolt > 100
    AreaNum   inrange "1 - 5 , 7 , 90-95"
    ZoneNum   between
  </SUBDATA>
BRANCH "a branch filter" "OR" "no"
  <subdata CONDITION>
    BusNomVolt:0 > 100 // Note location 0 means from bus
    BusNomVolt:1 > 100 // Note location 1 means to bus
    LineMW:0 > 100 1 // Note, final field 1 denotes absolute value
    _UseAnotherFilter meets
  </SUBDATA>
}
```

## Gen

### **BidCurve**

BidCurve subdata is used to define a piece-wise linear cost curve (or a bid curve). Each bid point consists of two real numbers on a single line of text: a MW output and then the respective bid (or marginal cost).

Example:

```
<SUBDATA BidCurve>
// MW Price[$/MWhr]
100.00 10.6
200.00 12.4
400.00 15.7
500.00 16.0
</SUBDATA>
```

### **ReactiveCapability**

Reactive Capability subdata is used to the reactive capability curve of the generator. Each line of text consists of three real numbers: a MW output, and then the respective Minimum MVAR and Maximum MVAR output.

Example:

```
<SUBDATA ReactiveCapability>
// MW MinMVAR MaxMVAR
100.00 -60.00 60.00
200.00 -50.00 50.00
400.00 -30.00 20.00
500.00 - 5.00 2.00
</SUBDATA>
```

Note: ReactiveCapability object types can also be directly created inside their own DATA section as well. Two of the key fields of the object are then the bus number and generator ID of the generator to which the ReactiveCapability point belongs.

## GeoDataViewStyle

### **TotalAreaValueMap**

This subdata section is used to define the lookup table for determining the total area size of geographic data view objects based on the value of a selected field. Two values are entered for each mapping:

FieldValue: Value of the field selected for the Total Area attribute.

TotalArea: The total area size of the object.

Example:

```
<SUBDATA TotalAreaValueMap>
// FieldValue TotalArea
1.000 0
4.000 23
7.000 46
</SUBDATA>
```

### **RotationRateValueMap**

This subdata section is used to define the lookup table for determining the rotation rate of geographic data view objects based on the value of a selected field. Two values are entered for each mapping:

FieldValue: Value of the field selected for the Rotation Rate attribute.

RotationRate: The rotation rate of the object. Entered in Hz.

Example:

```
<SUBDATA RotationRateValueMap>
// FieldValue RotationRate
1.000 0.00
4.000 0.10
7.000 0.20
</SUBDATA>
```

## RotationAngleValueMap

This subdata section is used to define the lookup table for determining the rotation angle of geographic data view objects based on the value of a selected field. Two values are entered for each mapping:

FieldValue: Value of the field selected for the Rotation Angle attribute.

RotationAngle: The rotation angle of the object. Entered in degrees.

Example:

```
<SUBDATA RotationAngleValueMap>
// FieldValue RotationAngle
1.000 -90.0
4.000 0.0
7.000 90.0
</SUBDATA>
```

## LineThicknessValueMap

This subdata section is used to define the lookup table for determining the thickness of the border line around geographic data view objects based on the value of a selected field. Two values are entered for each mapping:

FieldValue: Value of the field selected for the Line Thickness attribute.

LineThickness: The line thickness of the border line around the object. This should be an integer value.

Example:

```
<SUBDATA LineThicknessValueMap>
// FieldValue LineThickness
1.000 1
4.000 2
7.000 3
</SUBDATA>
```

## GlobalContingencyActions

### CTGElementAppend

This format is the same as for the Contingency objecttype

### CTGElement

This format is the same as for the Contingency objecttype

Note: GlobalContingencyActionsElement object types can also be directly created inside their own DATA section as well.

## HintDefValues

### HintObject

Stores the values for the custom hints. Each line has one value:

FieldDescription : This is a string enclosed in double quotes. The string itself is delimited by the @ character. The string contains five values:

- Name of Field : The name of the field. Special fields that appear on dialog by default have special names. Otherwise these are the same as the fieldnames of the AUX file format (for the "other fields" feature on the dialogs).
- Total Digit : Number of total digits for a numeric field.
- Decimal Points : Number of decimal points for a numeric field.
- Include Suffix : Set to 0 for not including the suffix, and set to 1 to include it.
- Field Prefix : The prefix text.

Example:

```
<SUBDATA HintObject>
"BusPUVolt@4@1@1@PU Volt ="
"BusAngle@4@1@1@Angle ="
</SUBDATA>
```



## InjectionGroup

### PartPoint

A participation point is used to describe the contents of an injection group. Each participation point lists six values:

- PointType : One of four values describing the type of point.
- GEN : a generator
  - LOAD : a load
  - SHUNT : a switched shunt
  - INJECTIONGROUP : another injection group
- PointBusNum : The bus number of the partpoint if the type is a GEN, LOAD, or SHUNT. Value will be blank for an injection group type. Note: bus# values may be replaced by a string enclosed in double quotes where the string is the name of the bus followed by an underscore character and then the nominal voltage of the bus. These values may also be replaced by a string enclosed in double quotes that represents the label of the bus or a string representing the label of the generator, load, or switched shunt.
- PointID : For GEN, LOAD, or SHUNT type, this is the id for the partpoint. For an INJECTIONGROUP type, this is the name or label of the injection group.
- PointParFac : The participation factor for the point.
- ParFacCalcType : How the participation point is calculated. There are several options depending on the PointType.
- Generators : SPECIFIED, MAX GEN INC, MAX GEN DEC, or MAX GEN MW
  - Loads : SPECIFIED or LOAD MW
  - Shunts : SPECIFIED, MAX SHUNT INC, MAX SHUNT DEC, or MAX SHUNT MVAR
  - Injection Groups : SPECIFIED
- ParFacNotDynamic: Should the participation factor be recalculated dynamically as the system changes.

#### Example:

```
<SUBDATA PartPoint>
  "GEN" 1 "1" 1.00 "SPECIFIED" "NO"
  "GEN" 4 "1" 104.96 "MAX GEN INC" "NO"
  "GEN" 6 "1" 50.32 "MAX GEN DEC" "YES"
  "GEN" 7 "1" 600.00 "MAX GEN MW" "NO"
  "LOAD" 2 "1" 5.00 "SPECIFIED" "NO"
  "LOAD" 6 "1" 200.00 "LOAD MW" "YES"
</SUBDATA>
```

Note: PartPoint object types can also be directly created inside their own DATA section as well. One of the key fields of the PartPoint object is then the name of the injection group to which the participation point belongs.

## Interface

### InterfaceElement

A interfaces's subdata contains a list of the elements in the interface. Each line contains a text descriptions of the interface element. Note that this text description must be encompassed by quotation marks. There are eleven kinds of elements allowed in an interface. Please note that the direction specified in the monitoring elements is important.

"BRANCH num1 num2 ckt"	: Monitor the MW flow on the branch starting from bus num1 going to bus num2 with circuit ckt. (order of bus numbers defines the direction)
"AREA num1 num2"	: Monitor the sum of the AC branches that connect area1 and area2.
"ZONE num1 num2"	: Monitor the sum of the AC branches that connect zone1 and zone2.
"BRANCHOPEN num1 num2 ckt"	: When monitoring the elements in this interface, monitor them under the contingency of opening this branch.
"BRANCHCLOSE num1 num2 ckt"	: When monitoring the elements in this interface, monitor them under the contingency of closing this branch.
"DCLINE num1 num2 ckt"	: Monitor the flow on a DC line.
"INJECTIONGROUP 'name'"	: Monitor the net injection from an injection group (generation contributes as a positive injection, loads as negative).
"GEN num1 id"	: Monitor the net injection from a generator (output is positive injection)
"LOAD num1 id"	: Monitor the net injection from a load (output is negative injection).
"MSLINE num1 num2 ckt"	: Monitor the MW flow on the multi-section line starting from bus num1 going to bus num2 with circuit ckt.
"INTERFACE 'name' "	: Monitor the MW flow on the interface given by name.

Note: bus# values may be replaced by a string enclosed in single quotes where the string is the name of the bus followed by an underscore character and then the nominal voltage of the bus. Labels may also be use as follows.

- bus# values for all element types may be replaced by a string enclosed in single quotes where the string is the label of the bus.
- for GEN or LOAD elements, the section *num1 id* may be replaced by the device's label.
- For MSLINE, DCLINE, or BRANCH elements, the *num1 num2 ckt* section may be replaced by the device's label.

For the interface element type "BRANCH num1 num2 ckt" and "DCLINE num1 num2 ckt", an optional field can also be written specifying whether the flow should be measured at the far end. This field is either YES or NO.

Example:

```
<SUBDATA InterfaceElement
  "BRANCH 8 9 1" NO // monitor the flow from bus 8 to bus 9 on this branch

  "BRANCH 12 33 1" YES // monitor the flow from bus 12 to bus 33 on branch
                        // measurefarend is set to true, therefore, we are
                        // monitoring the MW flow that arrives at bus 33
  // the following demonstrates the format when bus names and
  // nominal voltages are used.
  "BRANCH 'Twelve_230' 'name33_230' 1" YES

  "AREA 2 1" // monitor tie line flow from area 2 to area 1
  "ZONE 66 53" // monitor tie lines flows from zone 66 to zone 53
  "BRANCHOPEN 5 6 1" // doe monitoring after branch opens
  "BRANCHCLOSE 7 10 1" // doe monitoring after branch closes
</SUBDATA>
```

Note: InterfaceElement object types can also be directly created inside their own DATA section as well. One of the key fields of the InterfaceElement object is then the name of the interface to which the interface element belongs.

## KMLExportFormat

### DataBlockDescription

This subdata section is used to describe the objects and fields that should be saved to a KML file. Same format as for the AUXFileExportFormatData subdata section.

## LimitSet

### **LimitCost**

LimitCost records describe the piece-wise unenforceable constraint cost records for use by unenforceable line/interface limits in the OPF or SCOPF. Each row contains two values

PercentLimit : Percent of the transmission line limit.  
Cost : Cost used at this line loading percentage value.

Example:

```
<SUBDATA LimitCost>
//Percent Cost [$/MWhr]
100.00 50.00
105.00 100.00
110.00 500.00
</SUBDATA>
```

## Load

### **BidCurve**

BidCurve subdata is used to define a piece-wise linear benefit curve (or a bid curve). Each bid point consists of two real numbers on a single line of text: a MW output and then the respective bid (or marginal cost). These costs must be increasing for loads.

Example:

```
<SUBDATA BidCurve>
// MW Price[$/MWhr]
100.00 16.0
200.00 15.7
400.00 12.4
500.00 10.6
</SUBDATA>
```

## LPVariable

### **LPVariableCostSegment**

Stores the cost segments for the LP variables. Each line contains four values:

Cost (Up) : Cost associated with increasing the LP variable.  
Minimum value : Minimum limit of the LP variable.  
Maximum value : Maximum limit of the LP variable.  
Artificial : Whether the cost segment is artificial or not.

Example:

```
<SUBDATA LPVariableCostSegment>
//Cost(Up) Minimum Maximum Artificial
-20000.0000 -10000000000.5801 -0.6000 YES
16.2343 -0.6000 0.0000 NO
16.5526 0.0000 0.6000 NO
16.8708 0.6000 1.2000 NO
17.1890 1.2000 1.8000 NO
17.5073 1.8000 2.4000 NO
20000.0000 2.4000 9999999999.4199 YES
</SUBDATA>
```

## ModelCondition

### **Condition**

ModelConditions are the combination of an object and a Filter. They are used to return when the particular object meets the filter specified. As a results, the subdata section here is identical to the Condition subdata section of a Filter. See the description there.

## ModelExpression

### LookupTable

LookupTables are used inside Model Expressions sometimes. These lookup table represent either one or two dimensional tables. If the first string in the SUBDATA section is "x1x2", this signals that it is a two dimensional lookup table. From that point on it will read the first row as "x2" lookup points, and the first column in the remainder of the rows as the x1 lookup values.

Example:

```
DATA (MODELEXPRESSION, [CustomExpression,ObjectType,CustomExpressionStyle,
CustomExpressionString,WhoAmI,VariableName,WhoAmI:1,VariableName:1], AUXDEF)
{
// The following demonstrated a one dimensional lookup table
22.0000, "oned", "Lookup", "", "Gen<KEY1>1</KEY1><KEY2>1</KEY2>",
"Gen<KEY1>1</KEY1><KEY2>1</KEY2><VAR>GenMW</VAR>", "", ""
  <SUBDATA LookupTable>
    // because it does not start with the string x1x2 this will
    // represent a one dimensional lookup table
    x1      value
      0.000000  1.000000
      11.000000  22.000000
      111.000000  222.000000
  </SUBDATA>
0.0000, "twod", "Lookup", "",
"Gen<KEY1>1</KEY1><KEY2>1</KEY2>",
"Gen<KEY1>1</KEY1><KEY2>1</KEY2><VAR>GenMW</VAR>",
"Gen<KEY1>6</KEY1><KEY2>1</KEY2>",
"Gen<KEY1>6</KEY1><KEY2>1</KEY2><VAR>GenMW</VAR>"
  <SUBDATA LookupTable>
    // because this starts with x1x2 this represent a two dimensional
    // lookup table. The first column represents lookup values for x1.
    // The first row represents lookup values for x2
    x1x2      0.100000  0.300000 // these are lookup heading for x2
      0.000000  1.000000  3.000000
      11.000000  22.000000  33.000000
      111.000000  222.000000  333.000000
  </SUBDATA>
}
```

## ModelFilter

### ModelCondition

A Model Filter's subdata contains a list of each ModelCondition in the filter. Because a list of Model Conditions is stored within Simulator, this subdata section only requires the name of each ModelCondition on each line and whether or not the condition is using the NOT operator as part of the Model Filter.

Example:

```
<SUBDATA ModelCondition>
// ModelConditionName NotCondition
"Name of First Model Condition" "NO"
"Name of Second Model Condition" "NO"
"Name of Third Model Condition" "NO"
</SUBDATA>
```

## MTDCRecord

An example of the entire multi-terminal DC transmission line record is given at the end of this record description. Each of the SUBDATA sections is discussed first.

### **MTDCBus**

For this SUBDTA section, each DC Bus is described on a single line of text with exactly 8 fields specified.

- DCBusNum : The number of the DC Bus. Note DC bus numbers are independent AC bus numbers.
- DCBusName : The name of the DC bus enclosed in quotes.
- ACTerminalBus : The AC terminal to which this DC bus is connected (via a MTDCConverter). If the DC bus is not connected to any AC buses, then specify as zero. You may also specify this as a string enclosed in double quotes with the bus name followed by an underscore character, following by the nominal voltage of the bus.
- DCResistanceToground : The resistance of the DC bus to ground. Not used by Simulator.
- DCBusVoltage : The DC bus voltage in kV.
- DCArea : The area that this DC bus belongs to.
- DCZone : The zone that this DC bus belongs to.
- DCOwner : The owner that this DC bus belongs to.

Note: MTDCBus object types can also be directly created inside their own DATA section as well. One of the key fields of the object is then the number of the MTDCRecord to which the MTDCBus belongs.

### **MTDCConverter**

For this SUBDTA section, each AC/DC Converter is described by exactly 24 field which may be spread across several lines of text. Simulator will keep reading lines of text until it finds 24 fields. All text to the right of the 24<sup>th</sup> field (on the same line of text) will be ignored. The 24 fields are listed in the following order:

- BusNum : AC terminal bus number.
- MTDCNBriges : Number of bridges for the converter.
- MTDCCConvEBas : Converter AC base voltage.
- MTDCCConvAngMxMn : Converter firing angle.
- MTDCCConvAngMxMn:1 : Converter firing angle max.
- MTDCCConvAngMxMn:2 : Converter firing angle min.
- MTDCCConvComm : Converter commutating resistance.
- MTDCCConvComm:1 : Converter commutating reactance.
- MTDCCConvXFRat : Converter transformer ratio.
- MTDCFixedACTap : Fixed AC tap.
- MTDCCConvTapVals : Converter tap.
- MTDCCConvTapVals:1 : Converter tap max.
- MTDCCConvTapVals:2 : Converter tap min.
- MTDCCConvTapVals:3 : Converter tap step size.
- MTDCCConvSetVL : Converter setpoint value (current or power).
- MTDCCConvDCPF : Converter DC participation factor.
- MTDCCConvMarg : Converter margin (power or current).
- MTDCCConvType : Converter type.
- MTDCMaxConvCurrent : Converter Current Rating.
- MTDCCConvStatus : Converter Status.
- MTDCCConvSchedVolt : Converter scheduled DC voltage.
- MTDCCConvIDC : Converter DC current.
- MTDCCConvPQ : Converter real power.
- MTDCCConvPQ:1 : Converter reactive power.

Note: MTDCConverter object types can also be directly created inside their own DATA section as well. One of the key fields of the object is then the number of the MTDCRecord to which the MTDCConverter belongs.

## MTDCTransmissionLine

For this SUBDTA section, each DC Transmission Line is described on a single line of text with exactly 5 fields specified:

DCFromBusNum : From DC Bus Number.  
DCToBusNum : To DC Bus Number.  
CKTID : The DC Circuit ID.  
Resistance : Resistance of the DC Line in Ohms.  
Inductance : Inductance of the DC Line in mHenries (Not used by Simulator).

Example:

```
DATA (RECORD, [Num,Mode,ControlBus])
{
//-----
// The first Multi-Terminal DC Transmission Line Record
//-----
1  "Current"  "SYLMAR3 (26098)"
  <SUBDATA Bus>
    //-----
    // DC Bus data must appear on a single line of text
    // The data consists of exactly 8 values
    // DC Bus Num, DC Bus Name, AC Terminal Bus, DC Resistance to ground,
    // DC Bus Voltage, DC Bus Area, DC Bus Zone, DC Bus Owner
    3  "CELILO3P"      0  9999.00  497.92  40  404  1
    4  "SYLMAR3P"     0  9999.00  439.02  26  404  1
    7  "DC7"          41311 9999.00  497.93  40  404  1
    8  "DC8"          41313 9999.00  497.94  40  404  1
    9  "DC9"          26097 9999.00  439.01  26  404  1
    10 "DC10"         26098 9999.00  439.00  26  404  1
  </SUBDATA>
  <SUBDATA Converter>
    //-----
    // convert subdata keeps reading lines of text until it has found
    // values specified for 24 fields. This can span any number of lines
    // any values to the right of the 24th field found will be ignored
    // The next converter will continue on the next line.
    //-----
    41311  2  525.00  20.25  24.00  5.00  0.0000  16.3100
           0.391048  1.050000  1.000000  1.225000  0.950000  0.012500
           1100.0000  1650.0000  0.0000 "Rect"  1650.0000 "Closed"
           497.931  1100.0000  547.7241  295.3274
    41313  4  232.50  15.36  17.50  5.00  0.0000  7.5130
           0.457634  1.008700  1.030000  1.150000  0.990000  0.010000
           2000.0000  2160.0000  0.1550 "Rect"  2160.0000 "Closed"
           497.940  2000.0000  995.8800  561.8186
    26097  2  230.00  20.90  24.00  5.00  0.0000  16.3100
           0.892609  1.000000  1.100000  1.225000  0.950000  0.012500
           -1100.0000  1650.0000 "" "Inv"  1650.0000 "Closed"
           439.009  1100.0000 -482.9099  274.5227
    26098  4  232.00  17.51  20.00  5.00  0.0000  7.5130
           0.458621  1.008700  1.100000  1.120000  0.960000  0.010000
           439.0000  2160.0000 "" "Inv"  2160.0000 "Closed"
           439.000  1999.9999 -878.0000  544.2775
  </SUBDATA>
  <SUBDATA TransmissionLine>
    //-----
    // DC Transmission Segment information appears on a single line of
    // text. It consists of exactly 5 value
    // From DCBus, To DCBus, Circuit ID, Line Resistance, Line Inductance
    //-----
    3      4      "1"          19.0000  1300.0000
    7      3      "1"          0.0100  0.0000
    8      3      "1"          0.0100  0.0000
    9      4      "1"          0.0100  0.0000
    10     4      "1"          0.0100  0.0000
  </SUBDATA>
//-----
// A second Multi-Terminal DC Transmission Line Record
//-----
}
```

```

2  "Current"  "SYLMAR4 (26100)"
<SUBDATA Bus>
  5  "CELILLO4P"      0 9999.00  497.92  40 404  1
  6  "SYLMAR4P"      0 9999.00  439.02  26 404  1
 11  "DC11"          41312 9999.00  497.93  40 404  1
 12  "DC12"          41314 9999.00  497.94  40 404  1
 13  "DC13"          26099 9999.00  439.01  26 404  1
 14  "DC14"          26100 9999.00  439.00  26 404  1
</SUBDATA>
<SUBDATA Converter>
 41312  2  525.00  20.26  24.00  5.00  0.0000  16.3100
 0.391048 1.050000 1.000000 1.225000 0.950000 0.012500
1100.0000 1650.0000 0.0000 "Rect" 1650.0000 "Closed"
497.931 1100.0000 547.7241 295.3969
 41314  4  232.50  15.45  17.50  5.00  0.0000  7.5130
 0.457634 1.008700 1.030000 1.150000 0.990000 0.010000
2000.0000 2160.0000 0.1550 "Rect" 2160.0000 "Closed"
497.940 2000.0000 995.8800 562.9448
26099  2  230.00  20.90  24.00  5.00  0.0000  16.3100
 0.892609 1.000000 1.100000 1.225000 0.950000 0.012500
-1100.0000 1650.0000 "" "Inv" 1650.0000 "Closed"
439.009 1100.0000 -482.9099 274.5227
26100  4  232.00  17.51  20.00  5.00  0.0000  7.5130
 0.458621 1.008700 1.100000 1.120000 0.960000 0.010000
439.0000 2160.0000 "" "Inv" 2160.0000 "Closed"
439.000 1999.9999 -878.0000 544.2775
</SUBDATA>
<SUBDATA TransmissionLine>
  5  6  "1"  19.0000  1300.0000
 11 5  "1"  0.0100  0.0000
 12 5  "1"  0.0100  0.0000
 13 6  "1"  0.0100  0.0000
 14 6  "1"  0.0100  0.0000
</SUBDATA>
}

```

Note: MTDCTransmissionLine object types can also be directly created inside their own DATA section as well. One of the key fields of the object is then the number of the MTDCRecord to which the MTDCTransmissionLine belongs.

## MultiSectionLine

### Bus

A multi section line's subdata contains a list of each dummy bus, starting with the one connected to the From Bus of the MultiSectionLine and proceeding in order to the bus connected to the To Bus of the Line. Note: bus# values may be replaced by a string enclosed in double quotes where the string is the name of the bus followed by an underscore character and then the nominal voltage of the bus, or the string may represent the label of the bus.

Example:

```

//-----
// The following describes a multi-section line that connects bus
// 2 - 1 - 5 - 6 - 3
//-----
DATA (MultiSectionLine, [BusNum, BusName, BusNum:1, BusName:1,
                        LineCircuit, MSLineNSections, MSLineStatus] )
{
2 "Two" 3 "Three" "&1" 2 "Closed"
  <SUBDATA Bus>
    1
    5
    6
  </SUBDATA>
}

```

## BusRenumber

This subdata section allows renumbering of the dummy buses. The entries in the subdata section must be the new bus number that should be assigned to each dummy bus followed by the name of the new bus. The entries can be either space or comma delimited. The bus number must be specified, but the name is optional. If the name is not included and a new bus needs to be created, the name will be the same as the number. If an incorrect number of dummy buses is entered for a multi-section line, none of the dummy buses will be updated for that line. If a dummy bus number is specified that matches an existing bus that is another dummy bus, the other dummy bus will be assigned to a new bus number and the current dummy bus will be assigned to the number specified in the data.

Example:

```
DATA (MultiSectionLine, [BusNum, BusNum:1, LineCircuit] )
{
1 2 "1"
  <SUBDATA BusRenumber>
    3 "Bus 3"
    4 "Bus 4"
    5 "Bus 5"
  </SUBDATA>
22 33 "1"
  <SUBDATA BusRenumber>
    14 "Bus 14"
    15 "Bus 15"
  </SUBDATA>
}
```

## Nomogram

### InterfaceElementA

### InterfaceElementB

InterfaceElementA values represent the interface elements for the first interface of the nomogram.

InterfaceElementB values represent the interface elements for the second interface of the nomogram. The format of these SUBDATA sections is identical to the format of the InterfaceElement SUBDATA section of a normal Interface.

### NomogramBreakPoint

This subdata section contains a list of the vertex points on the nomogram limit curve.

Example:

```
<SUBDATA NomogramBreakPoint>
// LimA LimB
-100 -20
-100 100
80 50
60 -10
</SUBDATA>
```

## NomogramInterface

### InterfaceElement

This follows the same convention as the InterfaceElement SUBDATA section described with the Interface objecttype.



## Owner

### **Bus**

This subdata section contains a list of the buses which are owned by this owner. Each line of text contains the bus number. As an alternative to specifying the bus number, a string enclosed in double quotes may be used where the string represents the name of the bus followed by an underscore character and then the nominal voltage of the bus, or the string may represent the label of the bus.

Example:

```
<SUBDATA Bus>
  1
  35
  65
</SUBDATA>
```

### **Load**

This subdata section contains a list of the loads which are owned by this owner. Each line of text contains the bus number followed by the load id. As an alternative to specifying the bus number, a string enclosed in double quotes may be used where the string represents the name of the bus followed by an underscore character and then the nominal voltage of the bus, or the string may represent the label of the bus. Also, instead of specifying the bus and load id, the label of the load enclosed in double quotes may be used.

Example:

```
<SUBDATA Load>
  5 1 // shows ownership of the load at bus 5 with id of 1
  423 1
</SUBDATA>
```

### **Gen**

This subdata section contains a list of the generators which are owned by this owner and the fraction of ownership. Each line of text contains the bus number, followed by the gen id, followed by an integer showing the fraction of ownership. As an alternative to specifying the bus number, a string enclosed in double quotes may be used where the string represents the name of the bus followed by an underscore character and then the nominal voltage of the bus, or the string may represent the label of the bus. Also, instead of specifying the bus and generator id, the label of the generator enclosed in double quotes may be used.

Example:

```
<SUBDATA Gen>
  78 1 50 // shows 50% ownership of generator at bus 78 with id of 1
  23 3 70
</SUBDATA>
```

### **Branch**

This subdata section contains a list of the branches which are owned by this owner and the fraction of ownership. Each line of text contains the from bus number, followed by the to bus number, followed by the circuit id, followed by an integer showing the fraction of ownership. As an alternative to specifying the bus numbers, strings enclosed in double quotes may be used where the string represents the name of the bus followed by an underscore character and then the nominal voltage of the bus, or the string may represent the label of the bus. Also instead of specifying the two numbers and a circuit id, the label of the branch enclosed in double quotes may be used.

Example:

```
<SUBDATA Branch>
  6 10 1 50 // shows 50% ownership of line from bus 6 to 10, circuit 1
</SUBDATA>
```

## PostPowerFlowActions

### **CTGElementAppend**

This format is the same as for the Contingency objecttype

## CTGElement

This format is the same as for the Contingency objecttype

Note: PostPowerFlowActionsElement object types can also be directly created inside their own DATA section as well.

## PWCaseInformation

### PWCaseHeader

This subdata section contains the Case Description in free-formatted text. Note: as it is read back into Simulator all spaces from the start of each line are removed.

## PWFormOptions

### PieSizeColorOptions

There can actually be several PieSizeColorOptions subdata sections for each PWFormOptions object. The first line of each subdata section, the first line of text consist of exactly four values

ObjectName : The objectname of the type of object these settings apply to. Will be either be BRANCH or INTERFACE.  
FieldName : The fieldname for the pie charts that these settings apply to.  
UseDiscrete : Set to YES to use a discrete mapping of colors and size scalars instead of interpolating for intermediate values.  
UseOtherSettings : Set to YES to default these settings to the BRANCH MVA values for BRANCH object. This allows you to apply the same settings to all pie charts.

After this first line of text, if the UseOtherSettings Value is NO, then another line of text will contain exactly three values:

ShowValue : This is the percentage at which the value should be drawn on the pie chart.  
NormalSize : This is the scalar size multiplier which should be used for pie charts below the lowest percentage specified in the lookup table.  
NormalColor : This is the color which should be used for pie charts below the lowest percentage specified in the lookup table.

Finally the remainder of the subdata section will contain a lookup table by percentage of scalar and color values. This lookup table will consist of consecutive lines of text with exactly three values

Percentage : This is the percentage at which the follow scalar and color should be applied.  
Scalar : A scalar (multiplier) on the size of the pie charts.  
Color : A color for the pie charts.

Example:

```
<SUBDATA PieSizeColorOptions>
  // ObjectName FieldName UseDiscrete UseOtherSettings
  Branch MVA YES NO
  // ShowValue NormalSize NormalColor
  80.0000 1.0000 16776960
  // Percentage Scalar Color
  80.0000 1.5000 33023
  100.0000 2.0000 255
</SUBDATA>
<SUBDATA PieSizeColorOptions>
  // ObjectName FieldName UseDiscrete UseOtherSettings
  Branch MW YES YES
</SUBDATA>
```

## PWLPOPFCTGViol

### **OPFControlSense**

### **OPFBusSenseP**

### **OPFBusSenseQ**

This stores the control sensitivities for each contingency violation during OPF/SCOPF analysis. Each line contains one value:

Sensitivity : The value of the sensitivity with respect to each control in OPFControlSense or with respect to each bus in OPFBusSenseP and OPFBusSenseQ.

Example:

```
<SUBDATA OPFControlSense>
// Value
  1.000441679
  2.447185E-7
 -1.1109307E-6
  1.6427327E-7
  0
</SUBDATA>
```

## PWLPTabRow

### **LPBasisMatrix**

This subdata section stores the basis matrix associated with the final LP OPF solution. Each line contains two values:

Variable : The basic variable.  
Value : The sensitivity of the constraint to the basic variable.

Example:

```
<SUBDATA LPBasisMatrix>
// Var Value
  1 1.00000
  2 1.00000
  5 1.00000
  6 1.00000
</SUBDATA>
```

## PWPVResultListContainer

### **PWPVResultObject**

This subdata section contains the results of a particular PV Curve scenario. The data consists of two general sections: the first three rows of text contain the "independent axis" of the PV Curve. The first row starts with the string INDNOM and is followed by a list of numbers representing the nominal shift, the second row starts with INDEXP and is followed by the export shift, and the third row starts with INDIMP and is followed by the import shift. Following after these rows is a list of all the tracked quantities. Each tracked quantity row consists of three parts which are separated by the strings ?f= and &v= . The first part of the string represents a description of the power system object being tracked, the second part represents the field name being tracked, and the third contains a list of all the values at the various shift levels.

Example:

```
<SUBDATA PWPVResultObject>
INDNOM          0.00  500.00 1000.00 1500.00 1750.00 1875.00 1975.00
INDEXP          0.00  500.00 1000.00 1500.00 1750.00 1875.00 1975.00
INDIMP          0.00 -417.23 -701.58 -890.58 -952.60 -975.35 -990.43
Bus '3'?f=BusPUVolt&v= 0.993  0.983  0.964  0.939  0.926  0.919  0.914
Bus '5'?f=BusPUVolt&v= 1.007  1.000  0.982  0.956  0.940  0.932  0.926
Gen '4' '1'?f=GenMVR&v= 19.99  245.27 523.62 831.13 986.84 1060.6 1118.7
Gen '6' '1'?f=GenMVR&v= -6.59 -120.84 -131.37 -39.53 48.35 103.8 154.5
</SUBDATA>
```

## LimitViol

This subdata section contains the limit violations of a particular PV Curve scenario. This subdata section would only exist if using the option to monitor limit violations with the PV tool. Each row consists of an identifier, either VLOW or VHIGH, to indicate the type of limit violation followed by the bus identifier based on the key field identifier chosen. The bus can be identified by number, name and nominal kV combination, or label. The bus identifier is followed by the limit in use to identify a voltage violation and this is followed by the voltage at the bus.

Example:

```
<SUBDATA LimitViol>
  VLOW 3      1.00000    0.99017
  VLOW 5      1.00000    0.98245
</SUBDATA>
```

## PVBusInadequateVoltages

This subdata section contains a list of buses that are considered to have inadequate voltages at each transfer level for a particular PV Curve scenario. This subdata section would only exist if using the option to store inadequate voltages. The data consists of two general sections: the first row starts with the string INDNOM and is followed by a list of numbers representing the nominal shift. The second and subsequent rows list the buses and inadequate voltages for any bus that has an inadequate voltage at any transfer level. Each row starts with the bus identifier followed by the voltages at that bus at the corresponding shift levels. If a voltage is not inadequate at a particular transfer level, a blank entry will appear instead of a voltage value. The bus identifier is based on the key field identifier chosen and can be number, name and nominal kV combination, or label.

Example:

```
<SUBDATA PVBusInadequateVoltages>
  // INDNOM ShiftLevel1 ShiftLevel2 ...
  // BUS Voltage1 Voltage2 ...
  INDNOM      0.000 100.000 200.000 300.000 400.000 500.000
  "Bus '3'"   0.99269 0.99278 0.99282 0.99280 0.99273 0.99262
  "Bus '4'"   "      1.00000 " " " "
</SUBDATA>
```

## PWQVResultListContainer

### PWPVResultObject

This subdata section contains the results of a particular QV Curve scenario. These results will exist when tracking quantities with the QV curve tool. The data consists of two general sections: the first three rows of text contain the "independent axis" of the QV Curve. The first three rows start with the strings INDNOM, INDEXP, and INDIMP and are followed by a list of numbers representing the setpoint voltage representing the V of the QV curve. Following after these rows is a list of all the tracked quantities. Each tracked quantity row consists of three parts which are separated by the strings ?f= and &v= . The first part of the string represents a description of the power system object being tracked, the second part represents the field name being tracked, and the third contains a list of all the values at the various setpoint voltage levels.

Example:

```
<SUBDATA PWPVResultObject>
  INDNOM      1.100    1.093    1.083    1.073    1.063
  INDEXP      1.100    1.093    1.083    1.073    1.063
  INDIMP      1.100    1.093    1.083    1.073    1.063
  Bus '1'?f=BusPUVolt&v=1.05000 1.05000 1.05000 1.05000 1.05000
  Bus '1'?f=BusKVVolt&v=144.89999 144.89999 144.89999 144.89999 144.89999
</SUBDATA>
```

## QVCurve

### QVPoints

This subdata section contains a list of the QV Curve points calculated for the respect QVCurve. Each line consists of exactly six values:

PerUnitVoltage : The per unit voltage of the bus for a QV point.  
FictitiousMvar : The amount of Mvar injection from the fictitious generator at this QV point.  
ShuntDeviceMvar : The Mvar injection from any switched shunts at the bus.

TotalMvar : The total Mvar injection from switched shunts and the fictitious generator.  
 ReservesMvar : Total amount of Mvar reserves available at the bus.  
 ReservesTotalMvar : Total Mvar injection from the switched shunts, fictitious generator, and available reserves.

Example:

```

DATA (QVCURVE, [BusNum,CaseName,qv_VQ0,qv_Q0,qv_Vmax,qv_QVmax,qv_VQmin,qv_Qmin,
               qv_Vmin,qv_QVmin,Qinj_Vmax,Qinj_0,Qinj_min,Qinj_Vmin])
{
5 "BASECASE" 0.880 0.000 1.100 312.490 0.480 -221.072
              0.180 -86.334 191.490 -77.373 -244.075 -89.562
  <SUBDATA QVPoints>
    // NOTE: This bus has a constant impedance
    // switched shunt value of -100 Mvar at it.
    //V(PU), Q(MVR), Q_shunt(MVR), Q_tot(MVR), Q_res(MVR), Q_tot_res(MVR)
    1.1000, 312.4898, -121.0000, 191.4898, 0.0000, 191.4898
    0.9800, 124.6619, -95.9656, 28.6963, 0.0000, 28.6963
    0.7800, -96.6202, -60.7808, -157.4010, 0.0000, -157.4010
    0.5800, -206.9895, -33.5960, -240.5855, 0.0000, -240.5855
    0.3800, -207.4962, -14.4113, -221.9075, 0.0000, -221.9075
  </SUBDATA>
}
  
```

## QVCurve Options

### **Sim\_Solution\_Options**

This subdata section contains solution options that will be used when running QV Curves. See explanation under the CTG\_Options object type for more information.

## SelectByCriteriaSet

### **SelectByCriteriaSetType**

This subdata section contains a list of the display object types which are chosen to be selected. Each line of the section consists of the following:

DisplayObjectType : The object type of the display object.  
 (FilterName) : This field is optional, but must be given if either of the following fields is given. See the Using Filters in Script Commands section for more information on specifying the filename.  
 (WhichFields) : For display objects that can reference different fields, this sets which of those fields it should select (e.g. select only Bus Name Fields). The value may be either ALL or SPECIFIED.  
 (ListOfFields) : If WhichFields is set to SPECIFIED, then a delimited list of fields follows.

Example:

```

<SUBDATA SelectByCriteriaSetType>
  DisplayAreaField "" "ALL"
  DisplayBus ""
  DisplayBusField "Name of Bus Filter" "SPECIFIED" BusName BusPUVolt BusNum
  DisplayCircuitBreaker ""
  DisplaySubstation ""
  DisplaySubstationField "" "SPECIFIED" SubName SubNum BusNomVolt BGLoadMVR
  DisplayTransmissionLine ""
  DisplayTransmissionLineField "" "ALL"
</SUBDATA>
  
```

## Area

This subdata section contains a list of areas which were chosen to be selected. Each line of the section consists of either the number or the name. When generated automatically by PowerWorld we also include the other identifier as a comment.

Example:

```
<SUBDATA Area>
 18 // NEVADA
 22 // SANDIEGO
 30 // PG AND E
 52 // AQUILA
</SUBDATA>
```

## Zone

This subdata section contains a list of zones which were chosen to be selected. Each line of the section consists of either the number or the name. When generated automatically by PowerWorld we also include the other identifier as a comment.

Example:

```
<SUBDATA Zone>
 680 // ID SOLUT
 682 // WY NE IN
</SUBDATA>
```

## ScreenLayer

This subdata section contains a list of screen layers which were chosen to be selected. Each line of the section consists of either the name.

Example:

```
<SUBDATA ScreenLayer>
"Border"
"Transmission Line Objects"
</SUBDATA>
```

## ShapefileExportDescription

This object uses the same subdata sections as **SelectByCriteriaSet**. The only distinction is that only buses and lines can be exported.

## StudyMWTransactions

### ImportExportBidCurve

This subdata section contains the piecewise linear transactions cost curves for areas involved in a MW transaction. Costs are only for areas that are not on OPF control. Curves must be monotonically increasing. Each line corresponds to a point in the cost curve, and it has two values:

MW : The MW value. Use negative values for imports (purchase) and positive values for exports (sales)  
Price : The price in \$/MWh.

Example:

```
<SUBDATA ImportExportBidCurve>
 //MW Price[$/MWh]
-20.00 5.00
-10.00 10.00
 0.00 15.00
 10.00 20.00
 20.00 45.00
 30.00 70.00
</SUBDATA>
```

## SuperArea

### **SuperAreaArea**

This subdata section contains a list of areas within each super area. Each line of text contains two values, the area number followed by a participation factor for the area that can be optionally used.

Example:

```
<SUBDATA SuperAreaArea>
  1  48.9
  5  34.2
 25  11.2
</SUBDATA>
```

## TSSchedule

### **SchedPoint**

This section stores the schedule time points used in Time Step Simulation. Each line contains seven values:

Date	:	The date of the point.
Hour	:	The hour of the point.
Pointtype	:	An integer specifying the point type. 0 – Numeric 1 – Boolean (Yes/No, Closed/Open) 2 – Text
Numeric Value	:	The numeric value if point type is Numeric. Otherwise it is just zero.
Boolean Value	:	The boolean value if point type is Boolean. Otherwise it is just false.
Text value	:	The text value if point type is Text. Otherwise it is just an empty string.
Audiofilename	:	The audio filename associated to the point. If none, it is just an empty string.

Example:

```
<SUBDATA SchedPoint>
 //Date   Hour      PointType  NValue  BValue  TValue  AValue
 5/8/2006          0          1.00   NO
 5/8/2006 6:00:00 AM  0          1.10   NO
 5/8/2006 12:00:00 PM 0          1.25   NO
</SUBDATA>
```

## UserDefinedDataGrid

### **ColumnInfo**

This follows the same convention as the ColumnInfo SUBDATA section described with the DataGrid objecttype.

# SCRIPT Section for Display Auxiliary File

```
SCRIPT ScriptName
{
script_statement_1;
.
script_statement_n;
}
```

Scripts may optionally contain a ScriptName. This enables you to call a particular SCRIPT by using the LoadScript action (see General Actions). After the optional name, the SCRIPT section begins with a left curly brace and ends with a right curly brace. Inside of this, script statements can be given. In general, a script statement has the following format

```
Keyword(arg1, arg2, ...);
```

- Statement starts with a keyword.
- The keyword is followed by an argument list which is encompassed in parentheses ( ).
- The arguments are separated by commas.
- If a single argument is a list of things, this list is encompassed by braces [ ]. (eg. SetData)
- Statements end with a semicolon.
- Statements may take up several lines of the text file.
- You may put more than one statement on a single text line.

Those familiar with using Simulator will know that there is a RUN and EDIT mode in Simulator. Some features in Simulator are only available in one mode or the other. This functionality will be preserved in the script language. In earlier versions of the software, certain functionality was organized by the "submode" feature. While existing scripts designed to work with submodes will still function as before, moving between submodes is no longer necessary.

By default, when a script is initially started, you will be placed in RUN mode.



## AXD Actions

The following script commands are available for AXD files

```
AutoInsertBorders;
AutoInsertBuses(LocationSource, MapProjection, AutoInsertBranches, InsertIfNotAlreadyShown,
    "Bus" "Simple Conic" YES/NO YES/NO
    "Substation" "Mercator"
    "File" "x,y"
    "filename", FileCoordinates);
    "x,y"
    "lon,lat"
AutoInsertLoads(MinkV, InsertTextFields, InsertEquivObjects);
    YES/NO YES/NO
AutoInsertGens(MinkV, InsertTextFields);
    YES/NO
AutoInsertSwitchedShunts(MinkV, InsertTextFields);
    YES/NO
AutoInsertLines(MinkV, InsertTextFields, InsertEquivObjects, InsertZBRPieCharts, InsertMSLines,
    YES/NO YES/NO YES/NO YES/NO
SBRImpedance, NoStubsZBRs, SingleCBZRs);
    YES/NO YES/NO
AutoInsertLineFlowObjects(MinkV, InsertOnlyIfNotAlreadyShown, LineLocation, Size, FieldDigits,
    YES/NO 0, 1, or 2
FieldDecimals, TextPosition, ShowMW, ShowMvar, ShowMVA, ShowUnits, ShowComplex);
    YES/NO YES/NO YES/NO YES/NO YES/NO YES/NO
AutoInsertSubstations(LocationSource, MapProjection, AutoInsertBranches,
    "Bus" "Simple Conic" YES/NO
    "Substation" "Mercator"
    "File" "x,y"
InsertIfNotAlreadyShown, "FileName", FileCoordinates);
    YES/NO "x,y"
    "lon,lat"
AutoInsertLineFlowPieCharts(MinkV, InsertOnlyIfNotAlreadyShown, InsertMSLines, Size);
    YES/NO YES/NO
AutoInsertInterfaces(InsertPieCharts, PieChartSize);
    YES/NO
ResetStubLocations(ZBRImpedance, NoStubsZBRs);
    YES/NO
FixFlowArrowLineEnds("OnelineName", "LayerName");
```

### **AutoInsertBorders;**

Use this action to automatically insert borders according to the settings in the AutoInsertBordersOptions object

### **AutoInsertBuses(LocationSource, MapProjection, AutoInsertBranches, InsertIfNotAlreadyShown, "filename", FileCoordinates);**

Use this action to automatically insert buses based on specified location data.

LocationSource : "Bus", "Substation" or "File"  
MapProjection : "Simple Conic", "Mercator" or "x,y"  
AutoInsertBranches : YES to insert transmission lines when finished, NO not to  
InsertOnlyIfNotAlreadyShown : YES if only buses that are not already shown should be inserted, NO to insert all buses.  
"filename" : (optional) path to location source file (if LocationSource is "File")  
FileCoordinates : (optional) format of coordinates in file "x,y" or "lon,lat" (if LocationSource is "File")

### **AutoInsertLoads(MinkV, InsertTextFields);**

Use this action to automatically insert loads.

MinkV : Minimum kV level to insert (default=0)  
InsertTextFields : (optional) insert text fields (default=YES)

### **AutoInsertSwitchedShunts(MinkV, InsertTextFields);**

Use this action to automatically insert switched shunts.

MinkV : Minimum kV level to insert (default=0)  
InsertTextFields : (optional) insert text fields (default=YES)

**AutoInsertLines(MinkV, InsertTextFields, InsertEquivObjects, InsertZBRPieCharts, InsertMSLines, ZBRImpedance, NoStubsZBRs, SingleCBZBRs);**

Use this action to automatically insert lines.

MinkV : (optional) minimum kV level to insert (default=0)  
InsertTextFields : (optional) insert text fields (default=YES)  
InsertEquivObjects : (optional) insert Equivalenced Objects (default=YES)  
InsertZBRPieCharts : (optional) insert pie charts for lines with no limit and bus ties (default=NO)  
InsertMSLines : (optional) insert MultiSection Lines (default=YES)  
ZBRImpedance : (optional) maximum PU impedance for bus ties (default =0.0001)  
NoStubsZBRs : (optional) ignore stubs for bus ties (default=YES)  
SingleCBZBRs : (optional) only insert a single circuit breaker (default=YES)

**AutoInsertLineFlowObjects(MinkV, InsertOnlyIfNotAlreadyShown, LineLocation, Size, FieldDigits, FieldDecimals, TextPosition, ShowMW, ShowMvar, ShowMVA, ShowUnits, ShowComplex);**

Use this action to automatically insert line flow objects.

MinkV : Minimum kV level to insert (default=0)  
InsertOnlyIfNotAlreadyShown: (optional) if existing line flow objects are ignored (default=YES)  
LineLocation : (optional) where to insert flow objects (default=0)  
0 : middle  
1 : 10%/90%  
2 : after stubs  
Size : (optional) size (default=5.0)  
FieldDigits : (optional) total digits in field (default=6)  
FieldDecimals : (optional) digits to the right of the decimal (default=2)  
TextPosition : (optional) position of fields relative to flow object (default=YES)  
YES : above  
NO : below  
ShowMW : (optional) show MW field (default=YES)  
ShowMvar : (optional) show Mvar field (default=YES)  
ShowMVA : (optional) show MVA field (default=YES)  
ShowSuffix : (optional) show field units (default=YES)  
ShowComplex : (optional) show complex form (MW+jMvar) (default=NO)

**AutoInsertSubStations(LocationSource, MapProjection, AutoInsertBranches, InsertIfNotAlreadyShown, "filename", FileCoordinates);**

Use this action to automatically insert substations based on specified location data.

LocationSource : "Bus", "Substation" or "File"  
MapProjection : "Simple Conic", "Mercator" or "x,y"  
AutoInsertBranches : YES to insert transmission lines when finished, NO not to  
InsertOnlyIfNotAlreadyShown : YES if only buses that are not already shown should be inserted, NO to insert all buses.  
"filename" : (optional) path to location source file (if LocationSource is "File")  
FileCoordinates : (optional) format of coordinates in file "x,y" or "lon,lat" (if LocationSource is "File")

**AutoInsertLineFlowPieCharts(MinkV, InsertOnlyIfNotAlreadyShown, InsertMSLines, Size);**

Use this action to automatically insert line flow pie charts.

MinkV : Minimum kV level to insert (default=0)  
InsertOnlyIfNotAlreadyShown: (optional) if existing line flow objects are ignored (default=YES)  
InsertMSLines : (optional) insert pie charts for Multi-Section Lines (default=YES)  
Size : (optional) size (default=5.0)

**AutoInsertInterfaces(InsertPieCharts, PieChartSize);**

Use this action to automatically insert line flow objects.

InsertPieCharts : (optional) Insert pie charts as well (default=YES)  
PieChartSize : (optional) default size of interface pie charts (default=50.0)

**ResetStubLocations(ZBRImpedance, NoStubsZBRs);**

Use this action to reset stub locations.

ZBRImpedance : (optional) max P.U. impedance for bus ties (default=0.0001)  
NoStubsZBRs : (optional) Ignore stubs for bus ties (default=YES)

### **FixFlowArrowLineEnds("OnelineName", "LayerName");**

The unmoving line flow arrow indicators that can be displayed on lines may not always be setup correctly. This script corrects the flow arrows so that they look at the end of the line to which they are the closest to determine the direction of flow.

- "OnelineName" : The name of the oneline to which this action should be applied. The oneline must already be open.
- "LayerName" : Name of the screen layer in which this action should be applied.

### **General Script Commands**

The following script commands defined above in the general SCRIPT section are available for display auxiliary files as well:

- ExitProgram
- LoadScript
- LoadData
- SelectAll
- UnSelectAll
- SetData
- SaveData
- SaveDataWithExtra
- CreateData
- DeleteFile
- RenameFile
- CopyFile
- SetCurrentDirectory
- SaveObjectFields

# DATA Section for Display Auxiliary File

```
DATA DataName(object_type, [list_of_fields], file_type_specifier, create_if_not_found)
{
data_list_1
.
.
.
data_list_n
}
```

Immediately following the DATA keyword, a DataName may optionally be included. Currently, this is not used for anything because there are no script commands that are supported with the display auxiliary files.

## DATA Argument List

The DATA argument list identifies what the information section contains. A left and right parenthesis "( )" mark the beginning and end of the argument list.

The file\_type\_specifier parameter distinguishes the information section as containing custom auxiliary data (as opposed to Simulator's native auxiliary formats), and indicates the format of the data. Currently, the parser recognizes two values for file\_type\_specifier:

(blank) or AUXDEF or DEF	Data fields are space delimited
AUXCSV or CSV or CSVAUX	Data fields are comma delimited

The object\_type parameter identifies the type of object or data element the information section describes or models. For example, if object\_type equals DISPLAYBUS, then the data describes Display BUS objects.

The list of object types Simulator's display auxiliary file parser can recognize will grow as new applications are developed. Within Simulator, a list of available object types can be obtained by going to the main menu and choosing Window, Export Display Object Fields, and then exporting the objects to Excel or a text file.

The list\_of\_fields parameter lists the types of values the ensuing records in the data section contain. The order in which the fields are listed in list\_of\_fields dictates the order in which the fields will be read from the file. Simulator currently recognizes different display field types, each identified by a specific field name. Because the available fields for an object may grow as new applications are developed for the convenience of our customers, you will always be able to obtain a list of the available object types and their fields by going to the main menu and choosing Window, Export Display Object Fields, and then choosing to export in either Excel or text format. Certainly, only a subset of these fields would be found in a typical custom auxiliary file. In crafting applications to export custom auxiliary files, developers need concern themselves only the fields they need to communicate between their applications and Simulator. A few points of interest regarding the list\_of\_fields are:

- The list\_of\_fields may take up several lines of the text file.
- The list\_of\_fields should be encompassed by braces [ ].
- When encountering the PowerWorld comment string '/' in one of these lines of the text file, all text to the right is ignored.
- Blank lines, or lines whose first characters are '/' will be ignored as comments.
- Field names must be separated by commas.

Example:

```
DATA (DISPLAYBUS, [BusNom, SOAuxiliaryID, // comment here
SOX, SOY, SOTickness, SOColor, SOUseFillColor, SOFillColor,
// comments allowed here too

// note that blank rows are ignored
SOSize, SOwidth, SOorientation, SOLevel, SOImmobile,
SLName, SOStyle, SODashed, // more comments
SOBelongsToGroup])
```

One general note regarding the field names is warranted. Some field names may be augmented with a field location. One example of this is the field BusNum used when identifying branches. Bus numbers must be used to identify the from and the to end of branches. To keep the number of fields from becoming too large, the same field name is used for both of these values. The from bus number is written as BusNum:0 and the to bus number is written as BusNum:1. Note that the :0 may be left off of field names.

The `create_if_not_found` field is optional. This specifies whether or not to create a new object if an existing one is not found. If the value is YES, objects will be created. If the value is NO, objects will not be created. If omitted, the default behavior of prompting the user about whether or not to create a non-existing object will continue. If loading an auxiliary file using the `LoadAxd` script command, the `create_if_not_found` field for the data section will override the `CreateIfNotFound` field with the script.

## Key Fields

Simulator uses certain fields to identify the specific object being described. These fields are called key fields. For example, the key field for `DISPLAYBUS` objects is `BusNum`, because a bus can be identified uniquely by its number. The key fields for `DISPLAYGEN` objects are `BusNum` and `GenID`. To properly identify each object, the object's key fields must be present. They can appear in any order in the `list_of_fields` (i.e. they need not be the first fields listed in `list_of_fields`). As long as the key fields are present, Simulator can identify the specific object.

Display objects have an additional key field used for identification because multiple objects can be present on the same one-line diagram that represent the same power system element. This extra key field is `SOAuxiliaryID`. This is a field that is unique for each type of display object and other key field combination. If there are two display buses that represent bus one in the power system, the `SOAuxiliaryID` field will be different for both. Simulator will automatically create unique identifiers when these objects are created graphically. They can also be user specified but are forced to be unique. This field does not need to be present when reading in a display auxiliary file, but if it is missing, Simulator assumes that the ID is "1". This field is the only key field identifier for objects that do not link to power system elements such as background lines and pictures, and therefore, should always be included when reading in these objects or the expected results may not be achieved.

By going to the main menu and choosing Help, Export Display Object Fields you will obtain a list of fields available for each object type. In this output, the key fields will appear with asterisks \*.

## Data List

After the data argument list is completed, the Data list is given. The data section lists the values of the fields for each object in the order specified in `list_of_fields`. The data section begins with a left curly brace and ends with a right curly brace. A few points of interest regarding the `value_list`:

- The `value_list` may take up several lines of the text file.
- Each new data object must start on its own line of text.
- When encountering the PowerWorld comment string `'/'` in one of these lines of the text file, all text to the right of this is ignored.
- Blank lines, or lines whose first characters are `'/'` will be ignored as comments.
- Remember that the right curly brace must appear on its own line at the end of the `data_list`.
- If the `file_type_specifier` is CSV, the values should be separated by commas. Otherwise, separate the field names using spaces.
- Strings can be enclosed in double quotes, but this is not required. You should however always inclose strings that contain spaces (or commas) in quotes. Otherwise, strings containing commas would cause errors for comma-delimited files, and spaces would cause errors for space-delimited formatted files.
- A string in the format `"&[ModelExpressionName]:[#digits]:[#decimals]"` will be treated as though the value of the named model expression is entered in the field, with `[#digits]` total and `[#decimals]` digits to the right of the decimal point. This format may be used in case information displays as well as AUX files.

## Special Data Sections

There are several object types that should be noted here because they can impact the reading of an entire display auxiliary file, overall look of the resulting one-line diagram, or require special input to properly import/export the object.

### GeographyDisplayOptions

Most objects supported in the display auxiliary file have coordinates that can be specified in the appropriate data sections. What these coordinates specify can be controlled by the `GEOGRAPHYDISPLAYOPTIONS` object. This object has only two fields available: `MapProjection` and `ShowLonLat`. There are three possible settings for `MapProjection`: `"x,y"`, `"Simple Conic"`, and `"Mercator"`. The choice of projection will determine how the `x,y` values for display objects are interpreted. `ShowLonLat` can be either `"YES"` or `"NO"`. If `ShowLonLat` is `"YES"`, the setting specified for the `MapProjection` will be the longitude,latitude projection used when reading/writing the object `x,y` values. If `ShowLonLat` is `"NO"`, the `x,y` values will always be interpreted as `x,y` regardless of the `MapProjection` setting. This object should be placed in the display auxiliary file before any other objects containing coordinates

are read. If this object is not included in the auxiliary file, the coordinates will be interpreted based on the current settings of map projection and whether or not coordinates are showing longitude,latitude.

## Picture

PICTURE objects represent background images that cannot be stored in a text file format. To properly include a PICTURE object in a display auxiliary file, the file containing the image must be saved and read along with the auxiliary file. The `FileName` field indicates the name and location of the image file. If the image file cannot be found when reading in a display auxiliary file and attempting to create a new object, no PICTURE object will be created. If attempting to update an existing object and the image file cannot be found, the object will not be updated with a new image, but the `FileName` field will be updated with the specified file name.

## PWFormOptions

One-line display options that affect the current display settings can be changed by using the PWFORMOPTIONS object. Usually, this object specifies named sets of options that can be selected and used to change the various one-line display options through the GUI. By including a specially named object, the current options can be changed through a display auxiliary file. PWFORMOPTIONS are named using the `ObjectName` field. Setting this field to "THESE\_OPTIONS\_ARE\_APPLIED\_TO\_THE\_CURRENT\_DISPLAY" will apply the specified set of options to the current one-line when the file is read. When saving the entire one-line to a display auxiliary file, a PWFORMOPTIONS object with this name is added to the file by default.

## View

Different views can be specified in the display auxiliary file using the VIEW object. Usually, this object is used to specify named sets of options used to select and change the view through the GUI. By including a specially named object, the current view can be changed through a display auxiliary file. VIEW objects are named using the `ViewName` field. Setting this field to "THIS\_VIEW\_IS\_APPLIED\_TO\_THE\_CURRENT\_DISPLAY" will apply the specified set of view options to the current one-line when the file is read. When saving the entire one-line to a display auxiliary file, a VIEW object with this name is added to the file by default.

## SubData Sections

The format described thus far works well for most kinds of data in Simulator. It does not work as well however for data that stores a list of objects. For example, a contingency stores some information about itself (such as its name), and then a list of contingency elements, and possibly a list of limit violations as well. For data such as this, Simulator allows <SubData>, </SubData> tags that store lists of information about a particular object. This formatting looks like the following

```
DATA (object_type, [list_of_fields], file_type_specifier, create_if_not_found)
{
value_list_1
  <SUBDATA subobject_type1>
    precise format describing an object_type1
    precise format describing an object_type1
    .
    .
  </SUBDATA>
  <SUBDATA subobject_type2>
    precise format describing an object_type2
    precise format describing an object_type2
    .
    .
  </SUBDATA>
value_list_2
  .
  .
value_list_n
}
```

Note that the information contained inside the <SubData>, </SubData> tags may not be flexibly defined. It must be written in a precisely defined order that will be documented for each SubData type. The description of each of these SubData formats follows.

### ColorMap

Same format as in data auxiliary files.

### CustomColors

Same format as in data auxiliary files.

### DisplayDCTramissionLine

### DisplayInterface

### DisplayMultiSectionLine

### DisplaySeriesCapacitor

### DisplayTransformer

### DisplayTransmissionLine

### Line

#### **Line**

This is a list of points defining the graphical line used to represent the object. Each set of coordinates can be enclosed in square brackets, [ ], or the brackets can be eliminated. The brackets will be included when Simulator generates an auxiliary file. The individual coordinates are separated by the specified delimiter, either a space or a comma, and if the brackets are included, the same delimiter should be used to separate sets of coordinates. The list

of points is in a somewhat free form and sets of coordinates can span multiple lines. Each point should either be in x,y coordinates or longitude,latitude coordinates. Which coordinates should be used depends on the current option settings for map projection and whether or not coordinates should be shown in longitude,latitude. If the display auxiliary file is automatically generated by Simulator, a comment will be included in the subdata section indicating the coordinate system in use during file creation.

Example using brackets and a comma delimiter:

```
<SUBDATA Line>
//Coordinates are x,y
[14.00000000, 63.00000000], [14.00000000, 60.00000000],
[20.00000000, 45.00000000], [20.00000000, 42.00000000]
</SUBDATA>
```

Example with no brackets and a space delimiter:

```
<SUBDATA Line>
//Coordinates are x,y
14.00000000 63.00000000 14.00000000 60.00000000
20.00000000 45.00000000 20.00000000 42.00000000
</SUBDATA>
```

## DynamicFormatting

Same format as in data auxiliary files.

## Filter

Same format as in data auxiliary files.

## GeoDataViewStyle

Same format as in data auxiliary files.

## PieChartGaugeStyle

### **ColorMap**

This is a lookup table by percentage of scalar and color values. This lookup table will consist of consecutive lines of text with exactly three values:

Percentage	:	This is the percentage at which the following scalar and color should be applied.
Scalar	:	A scalar (multiplier) on the size of the pie chart/gauge.
Color	:	A color for the pie chart/gauge.

Example:

```
<SUBDATA ColorMap>
//Percentage Scalar Color
85.0000 1.5000 33023
100.0000 2.0000 255
</SUBDATA>
```

## PWFormOptions

Same format as in data auxiliary files.

## SelectByCriteriaSet

Same format as in data auxiliary files.

## UserDefinedDataGrid

Same format as in data auxiliary files.



## View

### **ScreenLayer**

This is a list of screen layer names that are hidden in the current view. Each screen layer name is on a separate line of text.

Example:

```
<SUBDATA ScreenLayer>  
//These are hidden screen layers  
  "pie layer"  
</SUBDATA>
```