Auxiliary File Format for Simulator 12.0

Last Updated: June 12, 2006

PowerWorld Corporation
2001 South First St
Champaign, IL 61820
(217) 384-6330
http://www.powerworld.com
info@powerworld.com

Table of Contents

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. Currently, there are no script commands that are available for *display auxiliary* files.

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)
{
data_list_1
    .
data_list_n
}

DATA DataName2(object_type, [list_of_fields], file_type_specifier)
{
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 { }. The Script commands available in Simulator 12.0 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 an 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, but additionally a feature called a “submode” will be used.

Submodes limit what script commands can be called. Only those commands available to the submode can be executed. To switch sumbodes, use the EnterMode (mode, submode) script command.

You will always be in one of the submodes when executing a script. By default, when a script is initially started, you will be placed in the RUN, POWERFLOW submode.

General Actions

Generic Data Actions

Available to you regardless of the Mode or SubMode

```
RenameFile          ("oldfilename", "newfilename");
CopyFile            ("oldfilename", "newfilename");
DeleteFile          ("filename");
LoadAux             ("filename", CreateIfNotFound);
LoadScript          ("filename", ScriptName);
LoadData            ("filename", DataName, CreateIfNotFound);

SelectAll           (objecttype, filter);
UnSelectAll         (objecttype, filter);
Delete              (objecttype, filter);

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

SetData             (objecttype, [fieldlist], [valuelist], filter);
CreateData          (objecttype, [fieldlist], [valuelist]);
ChangeData          (objecttype, [fieldlist], [valuelist], filter); (NOT AVAILABLE YET)

WriteTextToFile     ("filename", "text...");
SetCurrentDirectory ("filedirectory", CreateIfNotFound);
```

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.

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.

DeleteFile(“filename”);

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

“filename” : The file name to delete.

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 can not be found will be created while reading in DATA sections from filename.

LoadScript(“filename”, ScriptName);

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.

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 can not be found will be created while reading in DATA sections from filename. If this parameter is not specified, then NO is assumed.

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 specified means to select all objects of this type.
SelectAll(objecttype, “filtername”); – “filtername” means select those that meet the filter
SelectAll(objecttype, AREAZONE); – AREAZONE means select those that meet the area/zone filters

UnSelectAll(objecttype, filter);

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

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 specified means to delete all objects of this type.
Delete(objecttype, “filtername”); – “filtername” means delete those that meet the filter.
Delete(objecttype, AREAZONE); – AREAZONE means delete those that meet the area/zone filters.
Delete(objecttype, SELECTED); – SELECTED means delete those that are selected.

SaveData(“filename”, filetype, objecttype, [fieldlist], [subdatalist], filter);

Use this action to save data in a custom defined format. A filter may optionally be specified to save only object which meet a filter.

“filename” : The file path and name to save.
filetype : AUXCSV or CSVAUX: save as a comma-delimited auxiliary data file.
AUXDEF or DEFAUX: save as a space-delimited auxiliary data file.
CSV: save as a normal CSV file without the AUX file syntax.
objecttype : The objecttype being saved.
[fieldlist] : A list of fields that you want to save.
[subdatalist] : A list of the subdata objecttypes to save with the
filter : There are four options for the filter:
SaveData(…);– No filter specified means to save all objects of this type.
SaveData(…, “filtername”); – “filtername” means save those that meet the filter.
SaveData(…, AREAZONE); – AREAZONE means save those that meet the area/zone filters.
SaveData(…, SELECTED); – SELECTED means save those that are selected.

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(…, “filtername”); – “filtername” sets data for all objects that meet the filter.

SetData(…, AREAZONE); – AREAZONE sets data for all objects that meet the area/zone filters.

SetData(…, SELECTED); – SELECTED sets data for all objects that are selected

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.

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.

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 can not be found, the directory will be created. If this parameter is not specified, then NO is assumed.

PowerWorld Simulator Actions

Available to you regardless of the Mode or SubMode

```
NewCase;
OpenCase               ("filename"); // assumes to open as PWB
OpenCase               ("filename", openfiletype );
                                    PWB
                                    GE
                                    PTI
                                    IEEECF
SaveCase               ("filename");    // assumes save as PWB
SaveCase               ("filename", savefiletype);
                                    PWB, PWB5, PWB6, PWB7,
                                    PTI23, PTI24, PTI25, PTI26, PTI27,
                                    GE
                                    IEEECF
EnterMode              (mode, submode);
                        EDIT  CASE
                        RUN   POWERFLOW
                              CONTINGENCY
                              ATC
                              FAULT
                              PV
                              QV

OpenOneline            ("filename", "view", FullScreen);

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

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

SaveYbusInMatlabFormat ("filename",    IncludeVoltages);
SaveJacobian           ("JacFileName",  "JIDFileName",   FileType,   JacForm)
                                                        M           R
                                                        TXT         P
SetParticipationFactors(Method, ConstantValue, Object);
                       MAXMWRAT                [Area num]
                       RESERVE                 [Zone num]
                       CONSTANT                SYSTEM
                                               DISPLAYFILTERS
GenForceLDC_RCC        (filter);
                       unspecified
                       "filtername"
                       SELECTED
                       PREFILTER
DirectionsAutoInsert   (Source, Sink, DeleteExisting, UseDisplayFilters, Start, Increment);
                       AREA     AREA  YES             YES                value  value
                       ZONE     ZONE  NO              NO
                       INJECTIONGROUP
                                SLACK
```

NewCase;

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

OpenCase(“filename”, OpenFileType);

This action will open a case stored in “filename” of the type OpenFileType;

“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, GE, IEEECF.

SaveCase(“filename”, SaveFileType);

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

“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. It may be one of the following strings
PWB, PWB5, PWB6, PWB7
PTI23, PTI24, PTI25, PTI26, PTI27
GE, IEEECF

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 is 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.

OpenOneline(“filename”, “view”, FullScreen);

Use this action to open a oneline diagram.

“filename” : The file name of the oneline 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 oneline diagram will be open in full screen mode. If this parameter is not specified, then NO is assumed.

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.

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.

CaseDescriptionClear;

Use this action 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.

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.

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.
JacForm : R – Rectangular coordinates Jacobian.
P – Polar coordinates Jacobian.

SetParticipationFactors(Method, ConstantValue, Object);

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

Method : 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 : Specify the constant value for use with.
Object : Specify which generators to set the participation factor for.
[Area *Num*]
[Zone *Num*]
SYSTEM
DISPLAYFILTERS

GenForceLDC_RCC(filter);

Use this action to force generators in the case onto line drop / reactive current compensation.

filter : There are four options for the filter:
GenForceLDC_RCC; – No filter specified means to set all generators.
GenForceLDC_RCC(“filtername”); – “filtername” means to set generators that meet the filter.
GenForceLDC_RCC(AREAZONE); – AREAZONE means to set generators that meet the area/zone filters.
GenForceLDC_RCC(SELECTED); – SELECTED means to set generators that are selected.

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

Use this action to auto-insert directions to the case

Source : AREA – Use areas as the source.
ZONE – Use zones as the source.
INJECTIONGROUP – use injection groups as the source.
Sink : AREA – Use areas as the sink.
ZONE – Use zones as the sink.
INJECTIONGROUP – use injection groups as the sink.
SLACK – Use the slack as the 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.
Incremement : The increment for subsequent directions.

Edit Mode Actions

Case Submode Actions

The following script commands are available during the case submode of Edit mode

```
Equivalence;
DeleteExternalSystem;
SaveExternalSystem ("filename", Savefiletype, withties);
Scale  (scaletype,       basedon,  [parameters],            ScaleMarker);
        LOAD             MW         [P, Q]                   BUS
        GEN              FACTOR     [P] means constant pf    AREA
        INJECTIONGROUP                                       ZONE
                                                             OWNER
        BUSSHUNT                    [P, +Q, -Q]
Move   ([elementA],                 [destination parameter]);
        [GEN numA idA]              [numB idB]
        [LOAD numA idA]             [numB idB]                  // NOT AVAILABLE YET
        [BRANCH numA1 numA2 cktA]   [numB1 numB2 cktB]          // NOT AVAILABLE YET

Combine  ([elementA],               [elementB]);
         [GEN numA idA]             [GEN numB idB]
         [LOAD numA idA]            [LOAD numB idB]             // NOT AVAILABLE YET
         [BRANCH numA1 numA2 cktA]  [BRANCH numB1 numB2 cktB]   // NOT AVAILABLE YET

SplitBus      ([element],  NewBusNumber,  InsertBusTieLine,  LineOpen);
              [BUS num]    num            YES                YES
                                          NO                 NO

MergeBuses    ([element],  Filter);
              [BUS num]    unspecified
              "filtername"
              SELECTED
              PREFILTER

TapTransmissionLine
       ([element],               PosAlongLine,  NewBusNumber,  ShuntModel,  TreatAsMSLine);
       [BRANCH numA1 numA2 cktA]  value in %     num             LINESHUNTS   YES
                                                                CAPACITANCE  NO
InterfacesAutoInsert  (Type,  DeleteExisting,  UseFilters,   Prefix,   Limits);
                       AREA   YES              YES           "pre"     ZEROS
                       ZONE   NO               NO                      AUTO
                                                                       [value1, ..., value8]
```

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.

DeleteExternalSystem;

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

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. My be one of the following strings
PWB, PWB5, PWB6, PWB7
PTI23, PTI24, PTI25, PTI26, PTI27
GE
IEEECF

WithTies : An optional parameter. One must specify the file type explicitly in order to use the WithTies parameter. Allows one to save transmission lines that tie the 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.

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 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.

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

NOTE: THIS ACTION IS ONLY AVAILABLE FOR GENERATORS
Use this action to move a generator, load, or transmission line.

[elementA] : The object that should be moved. Must be one of the following formats:
[GEN busnum id]
[LOAD busnum id]
[BRANCH busnum1 busnum2 ckt]

[destination parameters] : These parameters have different meanings depending on object type of the element.
GEN - [busnum id]
LOAD – [busnum id]
BRANCH – [busnum1 busnum2 id]

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 can not combine a BRANCH and a LOAD.

[elementA] : The object that should be moved. Must be one of the following formats:
[GEN busnum id]
[LOAD busnum id]
[BRANCH busnum1 busnum2 ckt]

[elementB] : The object that element A should be combined with. Must the same format as for elementA.

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

Use this action to split buses

Element : Enter the description of which bus to split by enclosing the word bus followed by the bus number in brackets.
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.

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 specified means to merge all buses into one.
MergeBuses([element], “filtername”); – “filtername” means merge those that meet the filter.
MergeBuses([element], AREAZONE); – AREAZONE means to merge those that meet area/zone filters.
MergeBuses([element], SELECTED); – SELECTED means merge those that are selected.

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 numA1 numA2 cktA].
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.
LINESHUNT – 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.

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.
DeleteExsiting : 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.

Run Mode Actions

The following script commands are available during any of the submodes of 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   (StoreoForBranches, StoreForInterfaces, LinearMethod);
                                  True                True                AC
                                  False               False               DC
                                                                          DCPS
CalculateLODF  ([BRANCH nearbusnum farbusnum ckt], LinearMethod);

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

CalculateVoltSense  ([BUS num]);

CalculateFlowSense  ([flowelement],          FlowType);
                     [INTERFACE "name"]      MW
                     [BRANCH num1 num2 ckt]  MVAR
                                             MVA

CalculateLossSense  (FunctionType);
CalculateVoltToTransferSense
       ([transactor seller], [transactor buyer], TransferType,  TurnOffAVR);
              same as above for PTDFs            [P, Q or PQ]   YES
                                                                NO
CalculateVoltSelfSense  (filter);
                         unspecified
                         "filtername"
                         SELECTED
                         PREFILTER
SetSensitivitiesAtOutOfServiceToClosest;
ZeroOutMismatches;
```

Animate(DoAnimate);

Use this action to animate all the open oneline diagrams.

DoAnimate : Set to YES or NO. YES means to start the animation of the open oneline 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]
[ZONE num]
[SUPERAREA “name”]
[INJECTIONGROUP “name”]
[BUS num]
[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.
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.

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”]
[BRANCH nearbusnum farbusnum ckt]
direction : The type of the transactor. Either BUYER or SELLER.
[transactor buyer] : The transactor of power. There are six possible settings:
[AREA num]
[ZONE num]
[SUPERAREA “name”]
[INJECTIONGROUP “name”]
[BUS num]
[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”]
[BRANCH num1 num2 ckt]
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 can not 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 can not 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]
[ZONE num]
[SUPERAREA "name"]
[INJECTIONGROUP "name"]
[BUS num]
[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("filtername"); – "filtername" means to set generators that meet the filter.
CalculateVoltSelfSense(AREAZONE); – AREAZONE means to set generators that meet the area/zone filters.
CalculateVoltSelfSense(SELECTED); – SELECTED means to set generators that are selected.

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.

PowerFlow Submode Actions

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

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
       (BCMethod, "filename1", "filename2", CreateIfNotFound1, CreateIfNotFound2);
        POWERFLOW "file.aux"   "file.aux"   YES                YES
        OPF       STOP         STOP         NO                 NO
OPFWriteResultsAndOptions  ("filename");

DiffFlowSetAsBase;
DiffFlowClearBase;
DiffFlowMode   (diffmode);
                PRESENT
                BASE
                DIFFERENCE
```

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.

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.
Default 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 can not 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 can not be found will be created while reading in DATA sections of filename2. Default Value = NO.

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.

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 can not 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 can not 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 can not 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 can not 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 can not 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 can not 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.

DiffFlowSetAsBase;

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

DiffFlowClearBase;

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

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.

Contingency Submode Actions

```
CTGSolveAll;
CTGSolve                   ("ContingencyName");
CTGSetAsReference;
CTGRestoreReference;
CTGProduceReport           ("filename");
CTGWriteResultsAndOptions  ("filename");
CTGAutoInsert;
CTGCalculateOTDF           ([transactor seller], [transactor buyer], LinearMethod);
```

CTGSolveAll;

Call this action to solve all the contingencies which are not marked skip.

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.

CTGProduceReport("filename");

Produces a text-based contingency analysis report using the settings defined in CTG_Options.

CTGWriteResultsAndOptions("filename");

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.

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.

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.

ATC Submode Actions

```
ATCDetermine               ([transactor seller],      [transactor buyer]);
                            [AREA num]                [AREA num]
                            [ZONE num]                [ZONE num]
                            [SUPERAREA "name"]        [SUPERAREA name]
                            [INJECTIONGROUP "name"]   [INJECTIONGROUP name]
                            [BUS num]                 [BUS num]
                            [SLACK]                   [SLACK]
ATCRestoreInitialState;
ATCIncreaseTransferBy      (amount);
ATCTakeMeToScenario        (RL,   G,     I);
ATCDetermineATCFor         (RL,   G,     I);
ATCWriteResultsAndOptions  ("filename");
ATCWriteToExcel            ("worksheetname");
```

ATCDetermine([transactor seller], [transactor buyer]);

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.

[transactor seller] : The seller (or source) of power. There are six possible settings:
[AREA num]
[ZONE num]
[SUPERAREA “name”]
[INJECTIONGROUP “name”]
[BUS num]
[SLACK]

[transactor buyer] : The buyer (or sink) of power. There are six possible settings which are the same as for the seller.

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);

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

ATCWriteResultsAndOptions(“filename”);

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.

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.

Fault Submode 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.

[BRANCH nearbusnum farbusnum ckt]
: This specifies the branch on which the fault occurs.

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.

PV Submode Actions

```
PVCreate            ("name",  [element source],         [element sink]);
                              [INJECTIONGROUP "name"]    [INJECTIONGROUP "name"]

PVSetSourceAndSink  ("name",  [element source],         [element sink]);
                              [INJECTIONGROUP "name"]    [INJECTIONGROUP "name"]

PVRun               ("name");
PVClearResults      ("name");
PVStartOver         ("name");
PVDestroy           ("name");

PVWriteResultsAndOptions   ("filename");

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

PVCreate("name", [elementSource], [elementSink]);

Call the function to create a PV study with "name" as identifier. You may optionally specify the source and sink elements for the study transaction.

“name” : String that identifies the PV study to be created.
[element source] : The source of power for the PV study. There is only one possible setting: [INJECTIONGROUP "name"].
[element sink] : The sink of power for the PV study. There is only one possible setting, which is the same as for the source.

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

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

"name" : String that identifies the PV study for which the source and sink elements are to be assigned to.
[element source] : The source of power for the PV study. There is only one possible setting: [INJECTIONGROUP "name"].
[element sink] : The sink of power for the PV study. There is only one possible setting, which is the same as for the source.

PVRun("name");

Call the function to start the PV study called "name".

"name" : String that identifies the PV study.

PVClearResults("name");

Call the function to clear all the results of the PV study called "name".

"name" : String that identifies the PV study.

PVStartOver("name");

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

"name" : String that identifies the PV study.

PVDestroy("name");

Call the function to destroy the PV study called "name".

"name" : String that identifies the PV study.

PVWriteResultsAndOptions("filename");

Call this action to save all the PV 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", …);
"filtername" means select those that meet the filter.

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.

QV Submode 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
```

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 can not 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", …); – "filtername" means select those that meet the filter.

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.

DATA Section

```
DATA DataName(object_type, [list_of_fields], file_type_specifier)
{
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. Simulator currently recognizes the following object types:

3WXFormer
Area
ATC_Options
ATCExtraMonitor
ATCGeneratorChange
ATCInterfaceChange
ATCLineChange
ATCLineChangeB
ATCScenario
ATCZoneChange
BGCalculatedField
Branch
Bus
BusGroupSwapRec
BusNumberSwap
BusViewFormOptions
CaseInfo_Options
ColorMap
Contingency
Ctg_AutoInsert_Options
Ctg_Options
CTGElementBlock
CustomCaseInfo
CustomCaseInfoRow
CustomColors
CustomExpression
DataGrid
DCTransmissionLine
DefDrawArea
DefDrawBackground
DefDrawBus
DefDrawGen
DefDrawInterface
DefDrawLineTransformer
DefDrawLoad
DefDrawShunt
DefDrawSubstation
DefDrawSuperArea
DefDrawZone
Direction
DynamicFormatting
Equiv_Options
ExportBidCurve
Filter
GEMotor
Gen
GlobalContingencyActions
GScenarioName
HintDefValues
IG_AutoInsert_Options
ImportBidCurve
ImportExportBidCurve
InjectionGroup
Interface
InterfaceAElement
InterfaceBElement
IScenarioName
Limit_Monitoring_Options
LimitCost
LimitSet
Load
LODF_Options
LPOPFMarginalControls
LPVariable
MessLog_Options
ModelCondition
ModelExpression
ModelFilter
MTDCRecord
MultiSectionLine
MvarMarginalCostValues
MWMarginalCostValues
MWTransaction
Nomogram
NomogramInterface
OPF_Options
OPFSolutionSummary
Owner
PieSizeColorOptions
PostPowerFlowActions
PTDF_Options
PVCurve_Options
PWCaseInformation
PWFormOptions
PWLPOPFCTGViol
PWLPTabRow
PWOPFTimePoint
PWPVResultListContainer
QVCurve
QVCurve_Options
RLScenarioName
Scale_Options
Schedule
ScreenLayer
SelectByCriteriaSet
ShapefileExportDescription
ShortCut
Shunt
Sim_Environment_Options
Sim_Misc_Options
Sim_Simulation_Options
Sim_Solution_Options
StudyMWTransactions
Substation
SuperArea
TLR_Options
TransferLimiter
Transformer
TSSchedule
UC_Options
ViolationCTG
WhatOccurredDuringContingency
XFCorrection
Zone

The list of object types Simulator's auxiliary file parser can recognize will grow as new applications for the technology are found. Within Simulator, you will always be able to obtain a list of the available object_types by going to the main menu and choosing Help, Export Object Fields, and then exporting the fields to Excel.

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 over 2,000 different field types, each identified by a specific *field name*. Because the available fields for an object may grow as new applications are developed, you will always be able to obtain a list of the available object_types by going to the main menu and choosing Help, Export Object Fields. 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 encompased 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])
```

One general note regarding the field names however. Some field names may be augmented with a field 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 fieldnames such as LineMW:0 may simply leave off the :0.

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 Help, Export 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 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.

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 possible 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)
{
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 sceanario. 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 senstivity 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>
```

BGCalculatedField

Condition

Bus Group Calculated Fields are used for creating a calculation that operates on the buses in a area, zone, or substation. Part of the definition is a filter which specifies which buses 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) and an integer describing the color 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.

Example:

```
<SUBDATA ColorPoint>
  // Value Color
  100.0000 127
   62.5000 65535
   50.0000 8388479
   12.5000 16711680
    0.0000 8323072
</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 are 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 use must specify "" as the ModelCriteria.

Status : Three options
CHECK – perform action if ModelCriteria is true
ALWAYS – perform action regardless of ModelCriteria
NEVER – do not perform action
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
```

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 and underscore character and then the nominal voltage of the bus.

Generator, Load, or Switched Shunt outage or insertion

```
GEN       | bus# id           | OPEN
LOAD      | bus# id           | CLOSE
SHUNT     | bus# id
INJECTIONGROUP | name
```

Takes a generator, load, or shunt 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 and underscore character and then the nominal voltage of the bus.

Generator, Load or Switched Shunt movement to another bus

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

Use to move generation, load or shunt at a bus1 over to bus2. 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.

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

```
GEN       | bus#            | SET_P_TO    | value | MW
LOAD      |                 | 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. Also can use 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 and underscore character and then the nominal voltage of the bus.

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

```
BUS       | bus#            | OPEN
```

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 and underscore character and then the nominal voltage of the bus.

Interface outage or insertion

```
INTERFACE | name            | OPEN
                            | CLOSE
```

Takes all monitored branches in the interface out of service, or puts them all in service

Injection Group change specific value

```
INJECTIONGROUP | name       | CHANGE_P_TO| value | MW
                            | SET_P_TO   |       | PERCENT
                                                 | MWMERITORDER
                                                 | PERCENTMERITORDER
```

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 points in the injection group by closing or opening points in the group in order of descending participation factors until the total change greater than or equal to the requested change has been achieved.

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.

DC Transmission Line outage or insertion

```
DCLine    | bus1# bus2# ckt | OPEN
                            | CLOSE
```

Takes DC Line 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 and underscore character and then the nominal voltage of the bus.

DC Line set a specific value

```
DCLINE    | bus1# bus2# ckt   | SET_P_TO    | value  | MW
                              | CHANGE_P_BY |        | PERCENT
                              | SET_I_TO    |        | AMPS
                              | CHANGE_I_BY |
```

Use to set the DC Line setpoint 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 and underscore character and then the nominal voltage of the bus.

Phase Shifter set a specific value

```
PHASESHIFTER| bus1# bus2# ckt | SET_P_TO    | value  | MW
                              | CHANGE_P_BY |        | PERCENT
                              | SET_I_TO    |
                              | CHANGE_I_BY |
```

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 and underscore character and then the nominal voltage of the bus.

3-Winding Transformer outage or insertion

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

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 and underscore character and then the nominal voltage of the bus.

Execute a Power Flow Solution

```
SOLVEPOWERFLOW
```

Include this action to cause the solution of the contingency to be split into pieces. Actions that area 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 and 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                     Model Criteria  Status   comment
  "BRANCH 40821 40869 1  OPEN" ""               ALWAYS //Raver - Paul 500 kV
  "GEN 45041 1  OPEN"          ""               ALWAYS //Trip Unit #2
  "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
  "GEN 40227 1  OPEN"          "Interface L3+"  CHECK  //Drop ~1200 MW
</SUBDATA>
```

LimitViol

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

ViolType : One of five 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

ViolElement : This field depends on the ViolType.
for VLOW, VHIGH – “bus1#” or “busname_buskv”
for INTER – “interfacename”
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.

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
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>
```

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 precendence 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 on which actions ended up being skipped because of their model criteria. Each line of the subdata section must be enclosed in quotes.

Example:

```
<SUBDATA WhatOccurredDuringContingency>
  "Applied: "
  "  OPEN Branch Two      (2)  TO  Five     (5) CKT 1 |  | CHECK | "
</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 precendence 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 can not 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 can not call a ContingencyBlock from within a contingencyblock.

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 red + 256*green + 256*256*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

Condition :

Possible Values	Alternate1	Alternate2	Requirements
between	><		requires othervalue
notbetween	~><		requires othervalue
equal	=	==	
notequal	<>	~=	
greaterthan	>		
lessthan	<		
greaterthanorequal	>=		
lessthanorequal	<=		
contains			
notcontains			
startswith			
notstartswith			
inrange			
notinrange			

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
   </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>
```

GlobalContingencyActions

CTGElementAppend

This format is the same as for the Contingency objecttype

CTGElement

This format is the same as for the Contingency objecttype

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 Preffix : 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 two values describing the type of violation.
GEN – a generator
LOAD – a load
PointBusNum : The bus number of the partpoint. Note: bus# values may be replaced by a string enclosed in double quotes where the string is the name of the bus followed by and underscore character and then the nominal voltage of the bus.
PointID : The generator or load id for the partpoint.
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
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>
```

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 five 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.
“AREA *num1 num2*” : Monitor the sum of the tie line MW flows from area to area.
“ZONE *num1 num2*” : Monitor the sum of the tie line MW flows from zone to zone.
“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.
“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 (contributes as a positive injection)
“LOAD *num1 id*” : Monitor the net *injection* from a load (contributes as a negative injection).

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.

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>
```

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”, the 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 it. Because a lis of Model Conditions is stored within Simulator, this subdata section only stores the name of each ModelCondition on each line.

Example:

```
<SUBDATA ModelCondition>
  "Name of First Model Condition"
  "Name of Second Model Condition"
  "Name of Third Model Condition"
</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 the DC bus numbers are independent of the 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.

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 24th field (on the same line of text) will be ignored. The 24 fields are listed in the following order:

BusNum : AC terminal bus number.
MTDCNBridges : Number of bridges for the converter.
MTDCConvEBas : Converter AC base voltage.
MTDCConvAngMxMn : Converter firing angle.
MTDCConvAngMxMn:1 : Converter firing angle max.
MTDCConvAngMxMn:2 : Converter firing angle min.
MTDCConvComm : Converter commutating resistance.
MTDCConvComm:1 : Converter commutating reactance.
MTDCConvXFRat : Converter transformer ratio.
MTDCFixedACTap : Fixed AC tap.
MTDCConvTapVals : Converter tap.
MTDCConvTapVals:1 : Converter tap max.
MTDCConvTapVals:2 : Converter tap min.
MTDCConvTapVals:3 : Converter tap step size.
MTDCConvSetVL : Converter setpoint value (current or power).
MTDCConvDCPF : Converter DC participation factor.
MTDCConvMarg : Converter margin (power or current).
MTDCConvType : Converter type.
MTDCMaxConvCurrent : Converter Current Rating.
MTDCConvStatus : Converter Status.
MTDCConvSchedVolt : Converter scheduled DC voltage.
MTDCConvIDC : Converter DC current.
MTDCConvPQ : Converter real power.
MTDCConvPQ:1 : Converter reactive power.

TransmissionLine

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  "CELILO4P"         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>
}
```

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 and underscore character and then the nominal voltage of the bus.

Example:

```
//---------------------------------------------------------------------------
//  The following describes a multi-section line that connnects 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>
}
```

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>
```

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.

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.

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.

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 branchs 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.

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

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 (a particular contingency or the base case). 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>
```

QVCurve

QVPoints

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

PerUnitVoltage : The per unit voltage of the bus for a QV point.
FictitiousMvar : The amount of Mvars injected by the fictitious generator at this QV point.
ShuntDeviceMvar : The Mvars injected by any switched shunts at the bus.
TotalMvar : The total Mvars injected by switched shunts and the fictitious generator.

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.
     1.1000,  312.4898, -121.0000,  191.4898
     0.9800,  124.6619,  -95.9656,   28.6963
     0.7800,  -96.6202,  -60.7808, -157.4010
     0.5800, -206.9895,  -33.5960, -240.5855
     0.3800, -207.4962,  -14.4113, -221.9075
     0.1800,  -86.3336,   -3.2284,  -89.5620
   </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) : A filter to apply to these types of objects. This field is optional, but must be given if either of the following fields are given.
(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.
Price : The price in $/MWh.

Example:

```
<SUBDATA ImportExportBidCurve>
  //MW   Price[$/MWh]
    0.00   0.00
   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

SchedPointList

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 SchedPointList>
   //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>
```

DATA Section for Display Auxiliary File

```
DATA DataName(object_type, [list_of_fields], file_type_specifier)
{
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. Simulator currently recognizes the following object types:

CaseInformationMemo
ColorMap
Contour
CustomColors
DisplayArea
DisplayAreaField
DisplayAreaPie
DisplayBranchGauge
DisplayBranchPie
DisplayBus
DisplayBusField
DisplayBusPie
DisplayCircuitBreaker
DisplayDCTransmissionLine
DisplayDCTransmissionLineField
DisplayGen
DisplayGenericModelField
DisplayGenField
DisplayGenPie
DisplayInjectionGroup
DisplayInjectionGroupPie
DisplayInterface
DisplayInterfaceField
DisplayInterfacePie
DisplayLineFlowArrow
DisplayLoad
DisplayLoadField
DisplayMultiSectionLine
DisplayMultiSectionLineField
DisplayMultiSectionLinePie
DisplayOwner
DisplayOwnerField
DisplayOwnerPie
DisplaySeriesCapacitor
DisplaySeriesCapacitorField
DisplayShunt
DisplayShuntField
DisplayShuntPie
DisplaySubstation
DisplaySubstationField
DisplaySubstationPie
DisplaySuperArea
DisplaySuperAreaField
DisplaySuperAreaPie
DisplayTransformer
DisplayTransformerField
DisplayTransformerPie
DisplayTransmissionLine
DisplayTransmissionLineField
DisplayZone
DisplayZoneField
DisplayZonePie
DocumentLink
DynamicFormatting
Ellipse
Filter
GeographyDisplayOptions
Group
Line
OnelineField
OnelineLink
Picture
PieChartGaugeStyle
PWFormOptions
Rectangle
Screenlayer
SelectByCriteriaSet
Text
TextBox
View

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

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 over 2,000 different field types, each identified by a specific *field name*. Because the available fields for an object may grow as new applications are developed, you will always be able to obtain a list of the available object_types by going to the main menu and choosing Help, Export Display Object Fields. 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 encompased 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, SOThickness, 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.

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.

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 `OOName` 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 possible 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)
{
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.

DisplayDCTramisssionLine

DisplayInterface

DisplayMultiSectionLine

DisplaySeriesCapacitor

DisplayTransformer

DisplayTransmissionLine

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.

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.

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>
```