ELECTRIC POWER RESEARCH INSTITUTE

REFERENCE GUIDE

The Distribution System Simulator™ (DSS)

Roger C. Dugan
Sr. Technical Executive
Electric Power Research Institute, Inc.
November, 2008

License

Table of Contents

Figures

Summary

The Open Distribution System Simulator (OpenDSS, or simply, DSS) is a comprehensive electrical system simulation tool for electric utility distribution systems. OpenDSS technically refers to the open-source implementation of the DSS. It is implemented as both a stand-alone executable program and a COM DLL designed to be driven from a variety of existing software platforms. The executable version adds a basic user interface on to the solution engine to assist users in developing scripts and viewing solutions.

The program basically supports all rms steady-state (i.e., frequency domain) analyses commonly performed for utility distribution systems. In addition, it supports many new types of analyses that are designed to meet future needs, many of which are being dictated by the deregulation of US utilities and the formation of distribution companies worldwide. Many of the features found in the program were originally intended to support distributed generation analysis needs. Other features support energy efficiency analysis of power delivery and harmonics analysis. The DSS is designed to be indefinitely expandable so that it can be easily modified to meet future needs.

The OpenDSS program has been used for:

- Distribution Planning and Analysis
- General Multi-phase AC Circuit Analysis
- Analysis of Distributed Generation Interconnections
- Annual Load and Generation Simulations
- Wind Plant Simulations
- Analysis of Unusual Transformer Configurations
- Harmonics and Interharmonics analysis
- Neutral-to-earth Voltage Simulations
- Development of IEEE Test feeder cases
- And more ….

The program has several built-in solution modes, including:

- Snapshot Power Flow
- Daily Power Flow
- Yearly Power Flow
- Harmonics
- Dynamics
- Faultstudy
- And others …

Through the COM interface, the user is able to add other solution modes and features externally and perform the functions of the simulator, including definition of the model data. Thus, the DSS could be implemented entirely independently of any database or fixed text file circuit definition. For example, it can be driven entirely from a MS Office tool through VBA, or from any other 3rd party analysis program that can handle COM. Users commonly drive the OpenDSS with the familiar Mathworks Matlab program. This provides powerful external analytical capabilities as well as excellent graphics for displaying results.

Many users find the scripting interface available with the stand-alone executable version sufficient for nearly all their work. As users find themselves repeatedly needing a feature for their work, the feature is implemented within the solution and connected to the text-based command interface.

The COM interface also provides direct access to the text-based command interface as well as numerous methods and properties for accessing many of the parameters and functions of the simulator's models. Through the command interface, user-written programs can generate scripts to do several desired functions in sequence. The input may be redirected to a text file to accomplish the same effect as macros and also provide some database-like characteristics. Many of the results can be retrieved through the COM interface as well as from various output files. Output files are typically written in Comma-separated Value (CSV) format that imports easily into other tools for post processing.

The experienced software developer has two additional options for using the OpenDSS tool:

1. Downloading the source code and modifying it to suit special needs.

2. Developing DLLs that plug into generic containers the OpenDSS provides [*being revised!*]. This allows developers to concentrate on the model of the device of interest while letting the DSS executive take care of all other aspects of the distribution system model. Such DLLs can be written in most common programming languages.

The DSS structure is illustrated in Figure 1

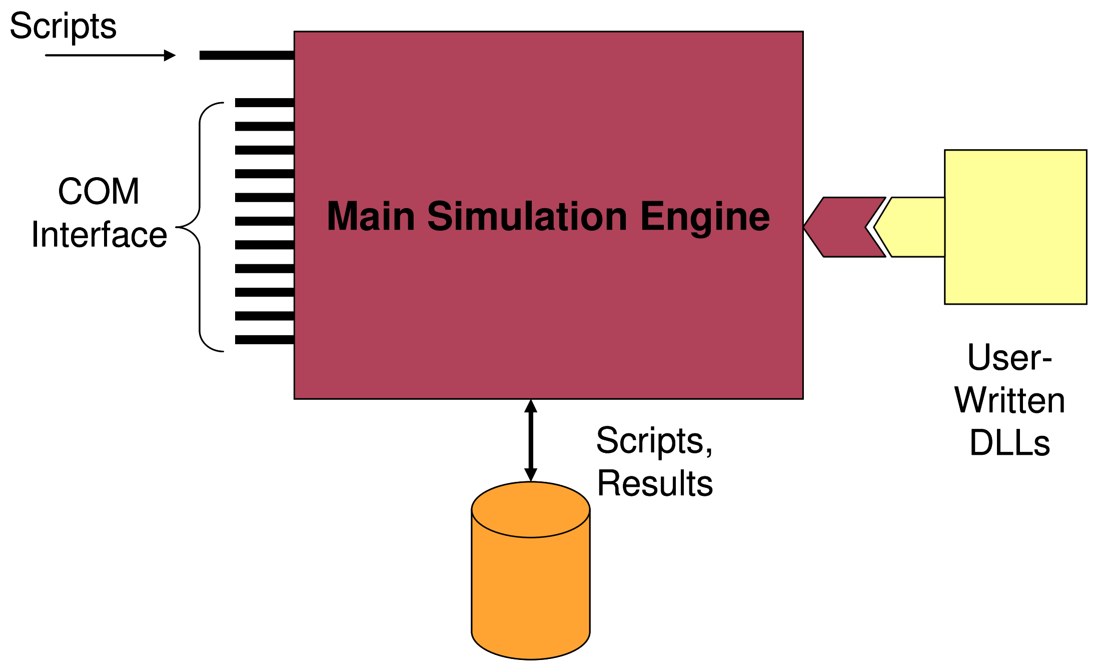


Figure 1. DSS Structure

Brief History and Objectives of the DSS

Development of the DSS began in April 1997 at Electrotek Concepts, Inc. Roger Dugan was the principal author of the software supported shortly thereafter by Tom McDermott. The two comprised the development team until late 2001 when Tom left Electrotek. Roger continued maintaining and evolving the program alone until recently when Tom again became part of the development team through the OpenDSS project. The DSS had been acquired by EPRI Solutions in 2004, which was united with EPRI in 2007. In 2008, EPRI released the software under open source to cooperate with other grid modernization efforts active in the Smart Grid area.

There were two events that triggered the development of the DSS in 1997:

1. EPRI had issued an RFP earlier for software to support the application of distributed generation to distribution systems. While Electrotek was not awarded a contract there had been enough thinking about the project to have arrived at a function spec for a simulator that would support the vast majority of electrical system analysis for evaluating DG for distribution planning purposes.

2. Roger Dugan was serving as Chair of the IEEE PES Software Engineering Working Group and one of the hot topics at that time was object-oriented programming and data representations. The late Mark Enns of Electrocon had issued a challenge to the group for someone to implement some of the principles we had been discussing in a power system analysis program. The distribution system simulator concept seemed the perfect vehicle to experiment with some of those ideas and also provide a useful tool to support the needs of a consulting company.

Prior to 1997, Electrotek had been performing DG studies for distribution planning using conventional distribution system analysis methods that are still employed by many tools today. We were well aware of the limitations of these methods and wanted a tool that was more powerful and flexible. One issue we discovered early on is that no two DG planning studies were exactly alike and we were constantly having to adjust our models. Thus, we needed a tool that would not lock us in to particular types of analysis. Other issues we set out to address were:

1. The distribution system model data we received from utilities came in all sorts of different formats, each with various strengths and various modeling limitations. One design goal was to develop an object-oriented circuit description language that minimized the conversion effort.

2. Window-based programs can be very user friendly, but we noticed some of the distribution system analysis tools really limited what you can do by limiting the interaction to what is available on the dialog forms. By being fundamentally script-driven, the DSS program gets around much of the limited-dialog issue and allows the user to better adapt the analysis process to the problem at hand.

3. The greater value of DG is often found on the subtransmission system serving a distribution planning area. Many distribution planning tools represent only the radial distribution system. Because of the methods employed by the economists

we were working with at the time, we wanted a tool that would allow us to model several substations and the distribution circuits between them simultaneously.

4. A key capability desired for the tool was to capture the time- and location-dependent value of DG. The location-dependent value is captured by modeling the DG in its actual location on the circuit. Capturing the time-dependent value requires extraordinary loadshape modeling capability.

5. There are many instances where it becomes necessary to model elements with many phases – not just three. For example, power poles in North America with multiple circuits may have as many as 4 circuits sharing a common neutral. Also, we wanted to be able to model what happens when a 69 kV line falls into the 13.2 kV distribution line. Or to model a communications signal passing from one voltage level to another through the interwinding capacitance of the transformer – this was not possible in many traditional distribution system analysis tools.

6. We wanted a tool that could seamlessly incorporate harmonics analysis into the power flow analysis without requiring the user to laboriously enter nonlinear device models. We also recognized that we would need at least simple dynamics analysis sufficient for DG interconnection evaluation. Simple models are built in and this feature continues to be developed.

7. Recognizing that distribution automation was going to become increasingly important, we wanted a testbed upon which to evaluate control algorithms and their impact on the operation of the system.

8. Recognizing that it was impossible to satisfy all possible user needs, we wanted a program that would allow users to write their own models or solution procedures commensurate with their capabilities.

9. We wanted a program that would simulate the behavior of devices on the distribution system as they would actually occur for changing load and system faults and other disturbances. This is important for modeling DG interactions and it also allows the tool to be used for many other things as well, such as energy efficiency analysis.

The present version of the OpenDSS has achieved the majority of these goals and has evolved into an extraordinary tool that has acquired many other features not commonly found in other distribution system analysis tools. While DG analysis continues to be one of its key uses, many other types of analysis have been performed with the tool.

The DSS is a general-purpose frequency-domain simulation engine that has special features for creating models of electric power distribution systems and performing many types of analysis related to distribution planning and power quality. It does not perform electromagnetic transients (time domain) simulations; all types of analysis are currently in the frequency domain (i.e., sinusoidal steady state, but not limited to 60 Hz).

Most electrical engineers learned how to write nodal admittance equations in their early University courses, and this is how the DSS represents circuits. Each element of the system is represented by a “primitive” nodal admittance (Y) matrix. This is generally straightforward, although it can be tricky for some power system elements such as transformers. Each primitive Y is then coalesced into one large system Y matrix, and the system of equations representing the distribution system is solved using sparse matrix solvers like one might also find in Matlab. One trick is that nonlinear behaviors of some devices (e.g., some load models) are modeled by current source injections, which some refer to as "compensation" currents. That is, the current predicted from the linear portion of the model that resides in the system Y matrix is compensated by an external injection to iteratively obtain the correct current. This is a common technique for representing loads in distribution system analysis tools. One advantage of this method is that it allows for quite flexible load models, which is important for performing some types of analysis such as done in energy efficiency studies.

The program's heritage is closer to a harmonic flow analysis program, or even a dynamics program, than a typical power flow program. This may seem a strange place to start designing a tool that will be used a great deal for power flow studies, but it gives the tool great modeling flexibility particularly for accommodating all sorts of load models and unusual circuit configurations.

WHY DELPHI?

The top structure of the OpenDSS that maintains the data definitions of the circuit elements is written in Object Pascal using the Delphi environment (originally from Borland, now from Codegear.com). The linear solvers employed by the DSS have been written in C and C++.

Since Pascal is seldom taught in engineering education any more, we are often asked why we used Delphi. At the time the DSS was started, the main developer, Roger Dugan, had been tinkering with the relatively new 32-bit versions of Delphi for a couple of years and wanted to try it on a large, hard-working engineering program that would not have the usual fixed problem size limitations of typical engineering programs of the time. A rapid development environment was needed because he was also doing engineering work and the DSS was mainly developed in spare time. The main alternative at the time would have been Visual Basic, but the performance of VB was not acceptable. Fortunately, when Tom McDermott joined Electrotek a few months later, he also knew Pascal and was able to contribute immediately to the development. Thus, the die was cast and the main part of the program has remained in Delphi as it has evolved.

While the choice of programming languages is largely one of personal preference, here are a few things that might be considered advantages with Delphi:

1. The compiler is fast. A typical full build of the 50,000+ line program takes only about 10-15 seconds. Re-builds are even faster. This allows fast debug-and-test cycles.

2. By default, the program is completely linked into one relatively compact EXE file and can be installed simply by copying the program to a desired disk location. This has made it convenient to distribute new versions to users. No complicated installs.

3. The text processing speed for reading circuit scripts has exceeded expectations and has proven more than adequate for the task. Our previous experience with

other engineering programs had led us to believe this was going to be a problem, but we were pleasantly surprised. You should find that the OpenDSS processes large circuit description scripts with relative ease. The math processing speed appears to be comparable to any compiled language.

4. Writing COM interfaces and DLLs is an easy task.

5. The structure of the Pascal language enforces a discipline that helps avoid the introduction of bugs. Most of the time, once you can get it to compile it will work. You should also find the code fairly readable. So if you are only interested in seeing how we did something, it should be easy to understand.

While we develop using the professional Delphi version, our intent is to keep OpenDSS compatible with the “Turbo” version that is presently offered for no cost.

INSTALLATION

Installation of the OpenDSS program is one of the easiest you will encounter in programs today: Simply copy the program files to a folder of your choosing (such as C:\OpenDSS) and start the program.

The following are the key program files:

1. OpenDSS.exe	(Stand alone executable)
2. OpenDSSEngine.DLL*	(The COM DLL version)
3. KLUSolve.DLL	(Sparse matrix solver)
4. DSSgraph.DLL	(Support for DSS graphics output)

* The OpenDSSEngine.DLL will have to be registered if you intend to access it from other programs such as Matlab and VBA in MS Office. You can do this by issuing the following command to the DOS command prompt when in the folder you have placed the files:

```
Regsvr32  OpenDSSEngine.DLL
```

There may be a .BAT file provided to perform this function. Registration need only be done once. After registering, the OpenDSSEngine will show up as “OpenDSS Engine” or “OpenDSSEngine.DSS” in the list of available object references in most development environments. For example, to instantiate an OpenDSSEngine object in Matlab, you would issue the following statement:

```
%instantiate the DSS Object
Obj = actxserver('OpenDSSEngine.DSS');
```

Except for the two DLLs indicated, the program in its present form is completely self-contained. If you are solely interested in creating scripts and executing the program manually, you can simply start the exe file and operate the DSS through the user interface provided. Most users use this mode most of the time. For convenience, you might drag a short cut to the desktop or some other convenient spot. You may also pin the exe to your start menu (right-click on the exe file and look at your options).

Note that when the program is executed, it will create an “.ini” file in the folder in which you installed the program files. The purpose of this file is to remember the contents of the user control panel and global settings such as the preferred text editor and base frequency. Thus, the program will start the next time with the same script window contents and defaults as left in the previous session.

Summary of Simulation Capabilities

The present version of the DSS is capable of performing the following analyses/simulations:

POWER FLOW

The DSS is designed to perform a basic distribution-style power flow in which the bulk power system is the dominant source of energy. It solves networked (meshed) systems as well as radial systems. It is intended to be used for distribution companies that may also have transmission or subtransmission systems. Therefore, it can also be used to solve small- to medium-sized networks with a transmission-style power flow.

The circuit model employed can be either a full multi-phase model or a positive-sequence model. The default is a full multi-phase model. For capacity studies, the latter is sufficient and will execute much faster. Due to the complex multi-phase models that may be created with myriad unbalances, the user is presently required to create positive-sequence models outside the DSS by defining a single-phase model of the circuit. However, by setting the proper flag, all power reports will report 3-phase quantities.

The power flow executes in numerous solution modes including the standard single Snapshot mode, Daily mode, Dutycycle Mode, Monte Carlo mode, and several modes where the load varies as a function of time. (See the Help command for "Set Mode" for the up-to-date listing of solution modes). The time can be any arbitrary time period. Commonly, for planning purposes it will be a 24-hour day, a month, or a year. Users may also write external macros or programs to drive the load models in some other manner.

When a power flow is completed, the losses, voltages, flows, and other information are available for the total system, each component, and certain defined areas. For each instant in time, the losses are reported as kW losses, for example. Energy meter models may be used to integrate the power over a time interval,.

The power flow can be computed for both radial distribution (MV) circuits and network (meshed) systems. While the accuracy of some algorithms, such as the calculation of expected unserved energy, may depend on part of the circuit model being radial, the power flow solution is general. It works best on systems that have at least one stiff source.

The two basic power flow solution types are

1. Iterative power flow
2. Direct solution

For the iterative power flow, loads and distributed generators are treated as injection sources. In the Direct solution, they are included as admittances in the system admittance matrix, which is then solved directly without iterating. Either of these two types of solutions may be used for any of the several solution modes by setting the global LoadModel property to “Admittance” or “Powerflow” (can be abbreviated A or P).

There are two iterative power flow algorithms currently employed:

1. "Normal" current injection mode
2. "Newton" mode.

The Normal mode is faster, but the Newton mode is more robust for circuit that are difficult to solve. The default is Normal.

Typically, power flow calculations will use an iterative solution with non-linear load models, and fault studies will use a direct solution with linear load models.

FAULT STUDIES

The DSS will perform fault (short-circuit) studies in several ways:

- A conventional fault study for all buses ("Set Mode=Faultstudy"), reporting currents and voltages on all phases for all types of faults: All-phase fault, SLG faults on each phase, L-L and L-L-G faults. Since transformers will be represented in actual winding configuration, this is an excellent circuit model debugging tool as well as a tool for setting relays and sizing fuses.

- A single snapshot fault. User places a fault on the system at a particular bus, defining the type of fault and the value of the fault resistance. A fault is a circuit element just like any other and can be manipulated the same way.

- Applying faults randomly. (Monte Carlo fault study mode -- solution mode= "MF"). User defines Fault objects at locations where faults are desired. This is useful for such analyses as examining what voltages are observed at a DG site for various faults on the utility system.

HARMONIC FLOW ANALYSIS

The DSS is a general-purpose frequency domain circuit solver. Therefore, harmonic flow analysis is quite natural and one of the easiest things to do with the program. The user defines various harmonic spectra to represent harmonic sources of interest. The spectra are connected to Load, Generator, voltage source, and current source objects as desired. There are reasonable default spectra for these elements.

A snapshot power flow is first performed to initialize the problem. The solution must converge before proceeding. If convergence is difficult to achieve, a direct solution is generally sufficient. Harmonic sources are then initialized to appropriate magnitudes and phases angles to match the solution.

Once a solution is achieved, the user simply issues the command:

```
Solve mode=harmonics
```

The DSS then solves for each frequency presently defined in any of the circuit elements.

Monitors are placed around the circuit to capture the results.

Frequency sweeps are performed similarly. The user defines spectra containing values for the frequencies (expressed as harmonics of the fundamental) of interest and assigns them to appropriate voltage or current sources. These sources may be defined to perform the sweeps in three different ways:

1. Positive Sequence: Phasors in 3-phase sources maintain a positive sequence relationship at all frequencies. That is, all three voltages and currents are equal in magnitude and displaced by 120 degrees.

2. Zero Sequence: All three voltages or currents are equal in magnitude and in phase.

3. No sequence: Phasors are initialized with the power flow solution and are permitted to rotate independently with frequency. If they are in a positive sequence relationship at fundamental frequency, they will be in a negative sequence relationship at the 2nd harmonic, and a zero sequence relationship at the 3rd harmonic, etc. In between integer harmonics, the phasors will be somewhere in between.

DYNAMICS

The DSS can perform basic machine dynamics simulations. The original intent was to provide sufficient modeling capability to evaluate DG interconnections. The built-in model has a simple single-mass model that is adequate for many DG studies for common distribution system fault conditions. In additional, users may implement more sophisticated models by writing a DLL for the Generator model or by controlling the Generator model from an external program containing a more detailed governor and/or exciter model.

An induction machine model was developed and used to help develop the IEEE Test Feeder benchmark dealing with large induction generation on distribution systems.

LOAD PARAMETRIC VARIATION

The capabilities for doing parametric evaluation are provided for a variety of variables. Certain variables will be allowed to vary according to a function (e.g., load growth) or vary randomly for Monte Carlo and statistical studies. See the Set Mode command documentation for descriptions of the Monte Carlo solution modes.

Basic Usage

DSS Control Panel

There is a multiple-window "Control Panel" for constructing and testing scripts, but the main access to scripts and results is generally through script files and output files or the COM interface.

Communication to the DSS is fundamentally accomplished through text strings passed to the DSS command processor. However, if you are driving the DLL version from another program, there are many functions provided in the COM interface to directly execute common commands (like "solve") and set properties of circuit elements without going through the command language. That might be a little faster for some operations, although the DSS command processor's text parsing is often more than fast enough for most applications.

Simulation results can be returned as arrays of values in the COM interface or in text through CSV files. Only a few standard text file reports will be provided by the base DSS software component (see Show and Export commands). The intent is for users to add customized reports through Excel worksheets or whatever application they use to control the DSS in special ways.

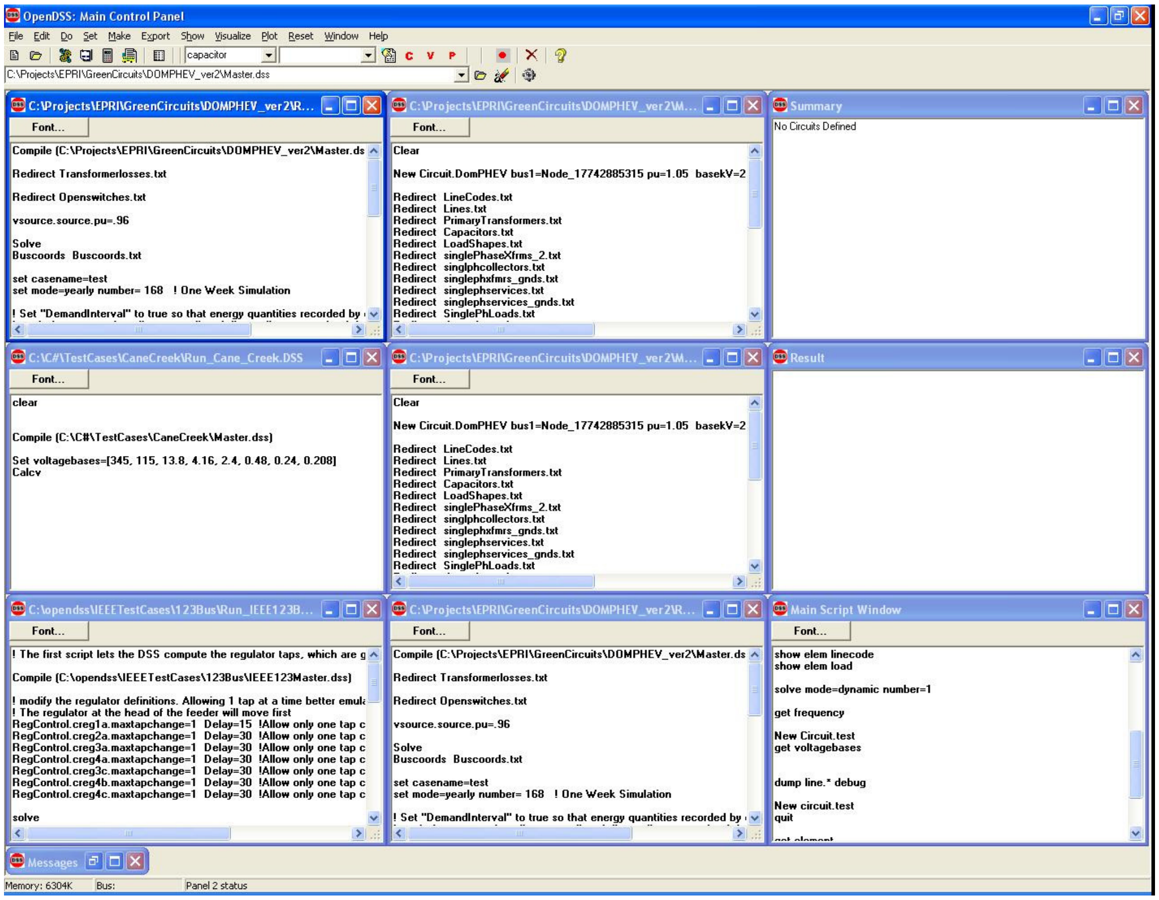
The DSS control panel automatically appears in the EXE version. The panel may also be invoked from the DSS COM DLL using the Panel or Show Panel commands or the ShowPanel method in the DSS COM interface.

(Caveat: When invoked from a MS Excel application, Excel may trap some of the key strokes, so it doesn't always work as smoothly as it does in the standalone application.)

The control panel (Figure 2) is a typical Windows multiple-document interface. There is a parent window containing several child windows. In Figure 2, the windows are tiled. There can be multiple script windows open at any one time. The user may execute a script, or a portion of a script, in any of the open windows at any time. This allows the user to organize scripts in a logical manner and have them available at any time. All scripts operate on the presently active circuit, or they may define a new circuit that becomes active.

Scripts or script fragments are executed by selecting the script lines to be executed. The user can right-click on the selection and then click on the Do Selected option, which has a short-cut key (ctrl-d). The selection may also be executed from the Do menu or the speed button directly below the Do menu item (Figure 3).

Of course, the script in the windows can redirect the DSS command input to a file, which can, in turn, redirect to another file. Thus, scripts can be quite voluminous without having to appear in a window on the control panel.


OpenDSS: Main Control Panel
File Edit Do Set Make Export Show Visualize Plot Reset Window Help
capacitor
C V P
C:\Projects\EPRI\GreenCircuits\DOMPHEV_ver2\Master.dss
C:\Projects\EPRI\GreenCircuits\DOMPHEV_ver2\R...
C:\Projects\EPRI\GreenCircuits\DOMPHEV_ver2\M...
Summary
C:\C#\TestCases\CaneCreek\Run_Cane_Creek.DSS
Result
C:\opendss\IEEETestCases\123Bus\Run_IEEE123B...
Main Script Window
Font...
Messages

```
Compile (C:\Projects\EPRI\GreenCircuits\DOMPHEV_ver2\Master.ds

Redirect Transformerlosses.txt

Redirect Openswitches.txt

vsource.source.pu=.96

Solve
Buscoords  Buscoords.txt

set casename=test
set mode=yearly number= 168   ! One Week Simulation

! Set "DemandInterval" to true so that energy quantities recorded by
```

```
Clear

New Circuit.DomPHEV bus1=Node_17742885315 pu=1.05  basekV=2

Redirect  LineCodes.txt
Redirect  Lines.txt
Redirect  PrimaryTransformers.txt
Redirect  Capacitors.txt
Redirect  LoadShapes.txt
Redirect  singlePhaseXfrms_2.txt
Redirect  singlphcollectors.txt
Redirect  singlephxfmrs_gnds.txt
Redirect  singlephservices.txt
Redirect  singlephservices_gnds.txt
Redirect  SinglePhLoads.txt
```

```
No Circuits Defined
```

```
clear


Compile (C:\C#\TestCases\CaneCreek\Master.dss)

Set voltagebases=[345, 115, 13.8, 4.16, 2.4, 0.48, 0.24, 0.208]
Calcv
```

```
Clear

New Circuit.DomPHEV bus1=Node_17742885315 pu=1.05  basekV=2

Redirect  LineCodes.txt
Redirect  Lines.txt
Redirect  PrimaryTransformers.txt
Redirect  Capacitors.txt
Redirect  LoadShapes.txt
Redirect  singlePhaseXfrms_2.txt
Redirect  singlphcollectors.txt
Redirect  singlephxfmrs_gnds.txt
Redirect  singlephservices.txt
Redirect  singlephservices_gnds.txt
Redirect  SinglePhLoads.txt
```

```
! The first script lets the DSS compute the regulator taps, which are g

Compile (C:\opendss\IEEETestCases\123Bus\IEEE123Master.dss)

! modify the regulator definitions. Allowing 1 tap at a time better emula
! The regulator at the head of the feeder will move first
RegControl.creg1a.maxtapchange=1  Delay=15  !Allow only one tap c
RegControl.creg2a.maxtapchange=1  Delay=30  !Allow only one tap c
RegControl.creg3a.maxtapchange=1  Delay=30  !Allow only one tap c
RegControl.creg4a.maxtapchange=1  Delay=30  !Allow only one tap c
RegControl.creg3c.maxtapchange=1  Delay=30  !Allow only one tap c
RegControl.creg4b.maxtapchange=1  Delay=30  !Allow only one tap c
RegControl.creg4c.maxtapchange=1  Delay=30  !Allow only one tap c

solve
```

```
Compile (C:\Projects\EPRI\GreenCircuits\DOMPHEV_ver2\Master.ds

Redirect Transformerlosses.txt

Redirect Openswitches.txt

vsource.source.pu=.96

Solve
Buscoords  Buscoords.txt

set casename=test
set mode=yearly number= 168   ! One Week Simulation

! Set "DemandInterval" to true so that energy quantities recorded by
```

```
show elem linecode
show elem load

solve mode=dynamic number=1

get frequency

New Circuit.test
get voltagebases


dump line.* debug

New circuit.test
quit
```

```
Memory: 6304K    Bus:    Panel 2 status
```

Figure 2. Example Control panel window for the DSS

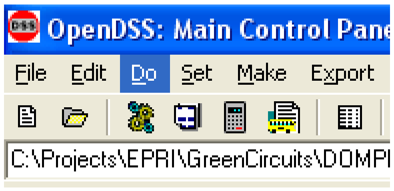

OpenDSS: Main Control Pane
File Edit Do Set Make Export
C:\Projects\EPRI\GreenCircuits\DOMPI

Figure 3. Do command and speed buttons

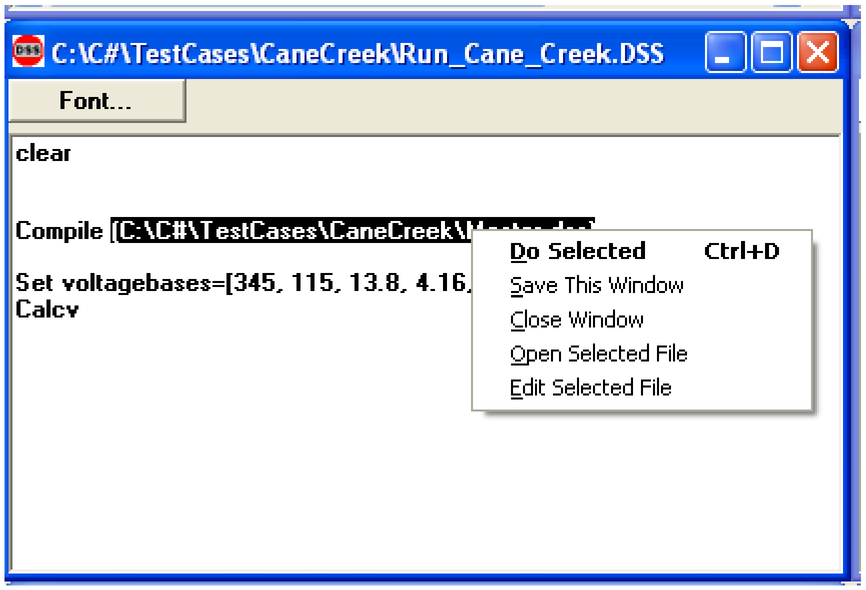
Each script window on the Control panel implements a Windows RichEdit text form. Thus, it has many of the standard text editing features one might expect. However, third party text editors (we use Editplus) can provide substantially more capability. It is recommended that one of the first commands to issue after starting the DSS for the first time is

Set Editor= *filename*

It is generally necessary to issue this command one time. The program will remember this value after it is shut down.

When a user right-clicks on a file name in one of the script windows, the popup menu gives the user the option of “Open” or “Edit” for the selected file (Figure 4). “Open” will open the selected file in another script window in the DSS program. “Edit” will open the selected file in the Editor of the user’s choosing. If no Editor was specified, the program will attempt to open the Notepad editor supplied with Windows.

If the file name contains no blanks, one need only click anywhere within the file name to execute the Open or Edit commands.



C:\C#\TestCases\CaneCreek\Run_Cane_Creek.DSS

```
clear

Compile [C:\C#\TestCases\CaneCreek\

Set voltagebases=[345, 115, 13.8, 4.16,
Calcv
```

Figure 4. Compiling a File containing a circuit description script from a script window in the control panel.

Export Menu Commands

The voltage and current exports have been updated to be more consistent and useful when you load them into Excel or another program.

- The output tables now stretch out horizontally instead of vertically. This is also done to fit in better with modifications to the Plot command
- Note: Current exports are keyed on device name
- Note: Voltage exports are keyed on bus name
- Residual values were added (sum of all conductors or nodes at a branch or bus). This is also true for the sequence current and sequence voltage exports because the residuals can be different than the zero-sequence values.

Here are the export options:

- Export Voltages [Filename] (EXP_VOLTAGES.CSV)
- Export SeqVoltages [Filename] (EXP_SEQVOLTAGES.CSV)
- Export Currents [Filename] (EXP_CURRENTS.CSV)
- Export Overloads [Filename] EXP_OVERLOADS.CSV)
- Export Unserved [UEonly] [Filename] EXP_UNSERVED.CSV)
- Export SeqCurrents [Filename] (EXP_SEQCURRENTS.CSV)
- Export Powers [MVA] [Filename](EXP_POWERS.CSV)
- Export SeqPowers [MVA] [Filename](EXP_SEQPOWERS.CSV)
- Export Faultstudy [Filename] (EXP_FAULTS.CSV)
- Export Generators [Filename | /m] (EXP_GENMETERS.CSV)
- Export Loads [Filename] (EXP_LOADS.CSV)
- Export Meters [Filename | /m] (EXP_METERS.CSV)
- Export Monitors monitorname (file name is assigned)

The circuit name is prepended onto the file names above. This is the name you gave the circuit when it was created. You can also change the name that gets prepended by the "Set Casename= …" option.

At the conclusion of an Export operation (as well as any other DSS operation that writes a file) the name of the output file appears in the Result window. You can quickly display/edit the file by:

1. Edit | Result File menu item
2. ctrl-R
3. There is a button on the tool bar for this purpose (see hints by holding the mouse over the buttons)

PLOT MENU COMMANDS

The existing "circuit plot" creates displays on which the thicknesses of LINE elements are varied according to:

- Power

- Current
- Voltage
- Losses
- Capacity (remaining)

This plot is a unicolor plot (specified by color C1). The following example will display the circuit with the line thickness propotional to POWER relative to a max scale of 2000 kW:

```
plot circuit Power Max=2000 dots=n labels=n subs=n C1=$00FF0000
```

The existing "general plot" creates displays on which the color of bus dots are varied according to an arbitrary quantity entered through a CSV file . The following gives a nice yellow-red variation of data field 1 (the bus name is assumed to be in the first column of the csv file) with the min displayed as color C1 (yellow) and the max=0.003 being C2 (red). Values in between are various shades of orange.

```
plot General 1 Max=.003 dots=n labels=n subs=y object=filename.csv  C1=$0080FFFF C2=$000000FF
```

Example results: This plot clearly shows the extent of harmonic resonance involving the residual current (likely, the zero sequence, too) on the circuit.

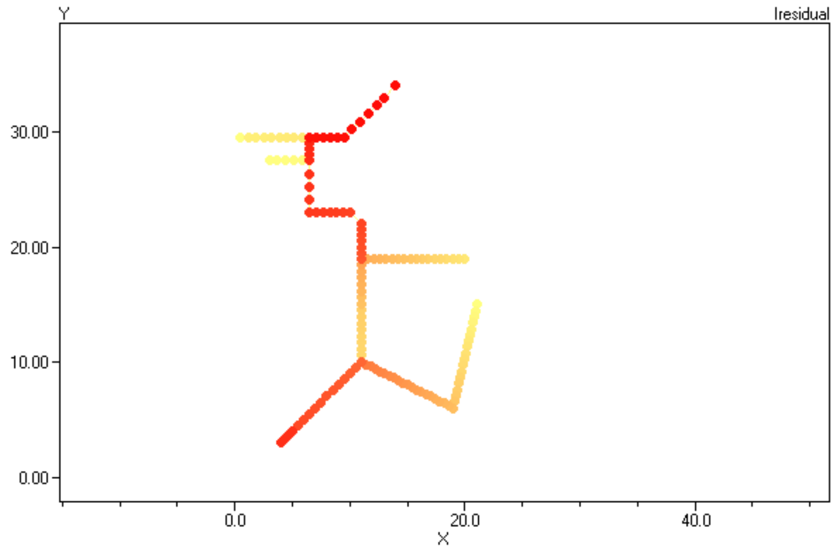


Figure 5: Color-shaded plot of harmonic resonance

There is also an option on the plot circuit command that will permit you to draw the circuit with lines proportional to some arbitrary quantity imported from a CSV file. The first column in the file should be the LINE name specified either by complete specification "line.elementname" or simply "elementname". If the command can find a line by that name in the present circuit, it will plot it.

The usual thing you would do is to export a file that is keyed on branch name, such as

Export currents, or

Export seqcurrents

Then you would issue the plot command, for example:

```
Plot circuit quantity=1 Max=.001  dots=n  labels=n Object=feeder_exp_currents.CSV
```

The DSS differentiates this from the standard circuit plot by:

1. A numeric value for “quantity” (representing the field number) instead of “Power”, etc
2. A non-null specification for “Object” (the specified file must exist for the command to proceed).

The Plot > Circuit Plots > General Line Data menu will guide you through the creation of this command with menus and list prompts (turn the recorder on). The first line in the CSV file is assumed to contain the field names, which is common for CSV files.

For example, the same plot as above with line thicknesses instead of colored dots.

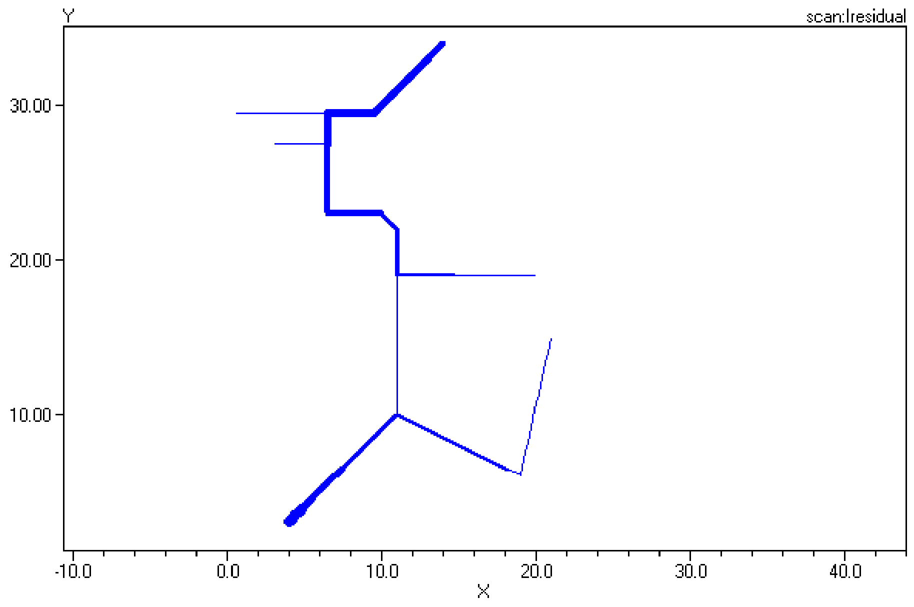


Figure 6: Thickness-weighted plot of harmonic resonance

Overall Circuit Model Concept

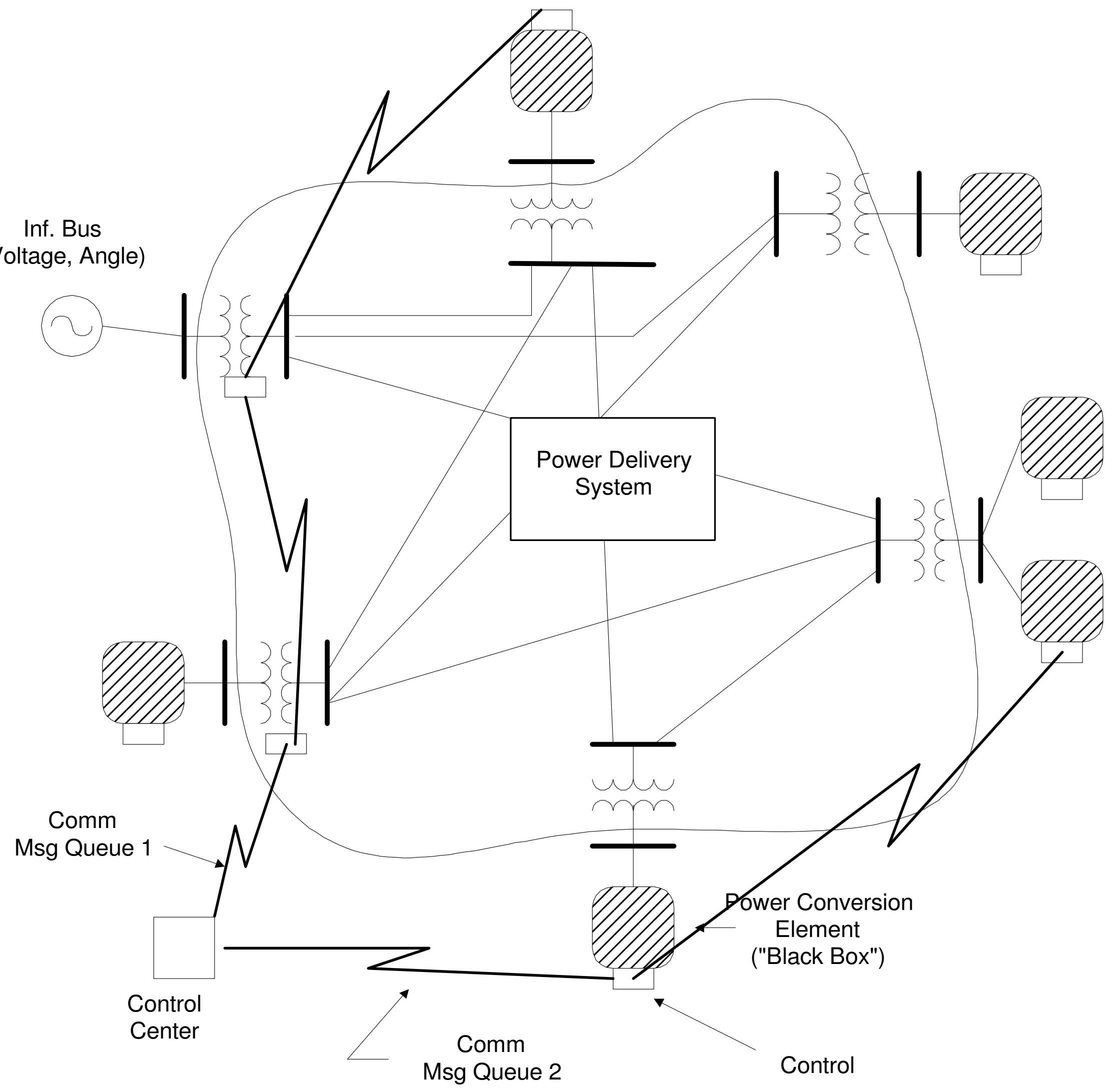


Figure 7: Electrical Circuit with Communications Network

The DSS consists of a model of the electrical system in the rms steady state, overlaid with a communications network that interconnects controls on power delivery elements and on power conversion elements.

[The communications message queues are not completely developed in this version -- one of the control queues is functional and used by the controls that are implemented.]

Bus and Terminal Models

BUS DEFINITION

A bus is a circuit element having [1..N] *nodes*. Buses are the connection point for all other circuit elements.

The main electrical property of a Bus is voltage. Each node has a voltage with respect to the zero voltage reference (remote ground). There is a nodal admittance equation written for every *node* (i.e., the current is summed at each node).

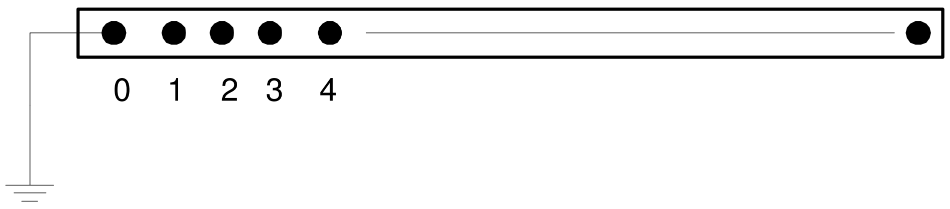


Figure 8: Bus Definition

TERMINAL DEFINITION

Each electrical element in the power system has one or more terminals. Each terminal has one or more conductors. Each conductor conceptually contains a disconnect switch and a TCC (fuse) curve *[The Fuse simulation has been disabled in this version and is being redesigned; a Relay object or a separate Fuse object can be used, if needed, to control the switches in the terminal conductors.]*. The conductors are numbered [1, 2, 3,…].

If the terminal is connected to an N-phase device, the first N conductors are assumed to correspond to the phases, in order. The remaining conductors may be neutrals or other non-power conductors.

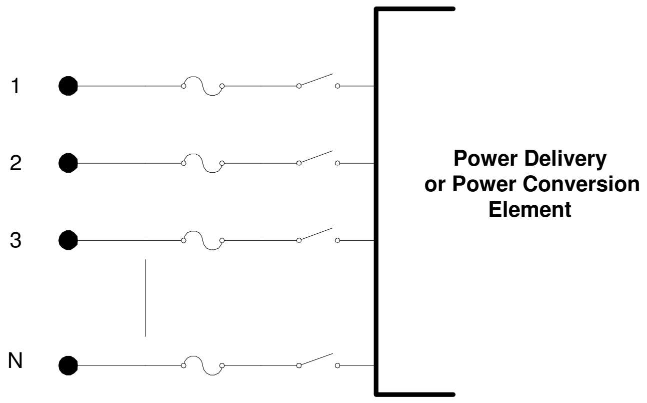


Figure 9: Terminal Definition

The DSS bus is a connecting place with 1 or more nodes for connecting individual phases and other conductors from the terminals of both power delivery elements and power conversion elements.

Buses are named with null-terminated strings of arbitrary length.

Node 0 of each bus is implicitly connected to the voltage reference (i.e., the node's voltage is always zero and is never explicitly included in the Y matrix).

BUS NAMING

Buses are named by an alphanumeric string of ASCII characters. The names may be numbers, but are always treated as strings. Internally, buses will be numbered (actually, each node is numbered), but will be referenced through the COM or command interface by name. The internal index number is subject to change if something in the circuit changes.

It is better if names do not contain blanks, tabs, or other “white space” or control characters. If the name contains a blank, for example, it must be enclosed in quotes (single or double). This can be a source of errors because of the different entry points for the names (user interface, files, COM interface).

Names may be any length and are passed to the DSS through the COM interface as standard null-terminated strings. This is the common way for representing strings in the Windows environment. Thus, it is a simple matter to drive the DSS from most programming languages and from Visual Basic in particular.

BUS INSTANTIATION AND LIFE

One of the features of the DSS that takes some getting used to for users familiar with other power system analysis platforms is that a Bus does not exist until it is needed for a solution or some other purpose. A list of buses is made from the presently enabled devices. Then each bus is instantiated.

The reason for this behavior of the DSS is to avoid having to define all the buses in the problem ahead of time. If you want to add a new bus, simply define a device that is connected to the bus or change the bus connections of an existing device. Since the DSS does not need to know the voltage bases to perform its solution, there is no need to define the base voltages ahead of time.

Bus instantiation can be forced without performing a solution by issuing the “MakeBusList” command.

The bus list is reorganized whenever there is a change in the circuit.

Implications/Side Effects:

- You can’t define the voltage base for a bus until it exists. The CalcVoltageBase command is a solution type that will automatically instantiate all the buses currently defined.

- You can’t define a bus’s coordinates until the bus exists.

Once the bus list exists, properties are copied to new bus objects when a change occurs that requires rebuilding the bus list. However, if there are new buses created by the changes, they will not have any of the special properties defined. That must be done subsequently, if necessary.

It does not hurt to redefine base voltage or coordinates if there is any question whether they have the proper value.

Once instantiated, a bus object will persist until another command forces rebuilding the bus list. Thus, if there are edits performed on the circuit elements, the bus list could be out of synch with the present configuration.

There is also a Node list that is constructed at the same time as the bus list. The order of elements in the system Y matrix is governed by the order of the Node list. The Node list order is basically the order in which the nodes are defined during the circuit model building process.

TERMINAL REFERENCES

Terminals are not named separately from the device. Each device will have a name and a defined number of multiphase terminals. Terminals will be referenced explicitly by number [1, 2, 3 …] or by inference, in the sequence in which they appear. Internally, they will be sequenced by position in a list.

PHASES AND OTHER CONDUCTORS

Each terminal has one or more **phases,** or normal power-carrying conductors and, optionally, a number of other conductors to represent neutral, or grounding, conductors or conductors for any other purpose. By convention, if a device is declared as having N phases, the first N conductors of each terminal are assumed to be the phase conductors, in the same sequence on each terminal. Remaining conductors at each terminal refer to “neutral” or “ground” or "earth" conductors.

All terminals of a device are defined to have the same number of conductors. For most devices, this causes no ambiguity, but for transformers with both delta and wye (star) winding connections, there will be an extra terminal at the delta connection. The neutral is explicit to allow connection of neutral impedances to the wye-connected winding. The extra conductor for the delta connection is simply connected to ground (voltage reference) and the admittances are all set to zero. Thus, the conductor effectively does not appear in the problem; it is ignored. However, you may see it appear in reports that explicitly list all the voltages and currents in a circuit.

The terminal conductors and bus nodes may be combined to form any practical connection.

Bus Nodes: A bus may have any number of *nodes* (places to connect device terminal conductors). The nodes may be arbitrarily numbered, except that the first N are by default reserved for the N phases. Thus, if a bus has 3-phase devices connected to it, connections would be expected to nodes 1, 2, and 3. So the DSS would use these voltages to compute the sequence voltages, for example. Phase 1 would nominally represent the same phase throughout the circuit, although there is nothing to enforce that standard. It is up to the user to maintain a consistent definition. If only the default connections are used, the consistency is maintained automatically.

Any other nodes would simply be points of connection with no special meaning. Each Bus object keeps track of the allocation and designation of its nodes.

Node 0 of a bus is **always the voltage reference** (a.k.a, ground, or earth). That is, it always has a voltage of **exactly zero volts**.

SPECIFYING CONNECTIONS

The user can define how terminals are to be connected to buses in three ways, not just one way:

1. Generically connect a circuit element's *terminal* to a *bus* without specifying node-to-conductor connections. This is the default connection. Normal phase sequence is assumed. Phase 1 of the terminal is connected to Node 1 of the Bus, and so on. Neutrals default to ground (Node 0).

2. Explicitly specify the first phase of the device is connected to node *j* of the bus. The remaining phases are connected in normal 3-phase sequence (1-2-3 rotation). Neutral conductors default to ground (Node 0).

3. Explicitly specify the connection for all phases of each terminal. Using this mode, neutrals (star points) may be left floating. Any arbitrary connection maybe achieved. The syntax is:

   BUSNAME.*i.j.k,* where i, j, k refer to the nodes of a bus.

   This is interpreted as the first conductor of the terminal is connected to node i of the bus designated by BUSNAME; the 2nd conductor is connected to node j, etc.

   The default node convention for a terminal-to-bus connection specification in which the nodes are not explicitly designated is:

   BUSNAME.1.0.0.0.0.0.0. … {Single-phase terminal)
   BUSNAME.1.2.0.0.0.0.0. … {two-phase terminal)
   BUSNAME.1.2.3.0.0.0.0 … {3-phase terminal}

   If the desired connection is anything else, it must be explicitly specified. Note: a bus object "learns" its definition from the terminal specifications. Extra nodes are created on the fly as needed. They are simply designations of places to connect conductors from terminals. For a 3-phase wye-connected capacitor with a neutral reactor, specify the connections as follows:

   BUSNAME.1.2.3.4 {for 3-phase wye-connected capacitor}
   BUSNAME.4 {for 1-phase neutral reactor (2nd terminal defaults to node 0)}

Power Delivery Elements

Power delivery elements usually consist of two or more multiphase terminals. Their basic function is to transport energy from one point to another. On the power system, the most common power delivery elements are lines and transformers. Thus, they generally have more than one terminal (capacitors and reactors can be an exception when shunt-connected rather than series-connected). Power delivery elements are standard electrical elements generally completely defined in the rms steady state by their *impedances*. Thus, they can be represented fully by the primitive **y** matrix (Y_{prim}).

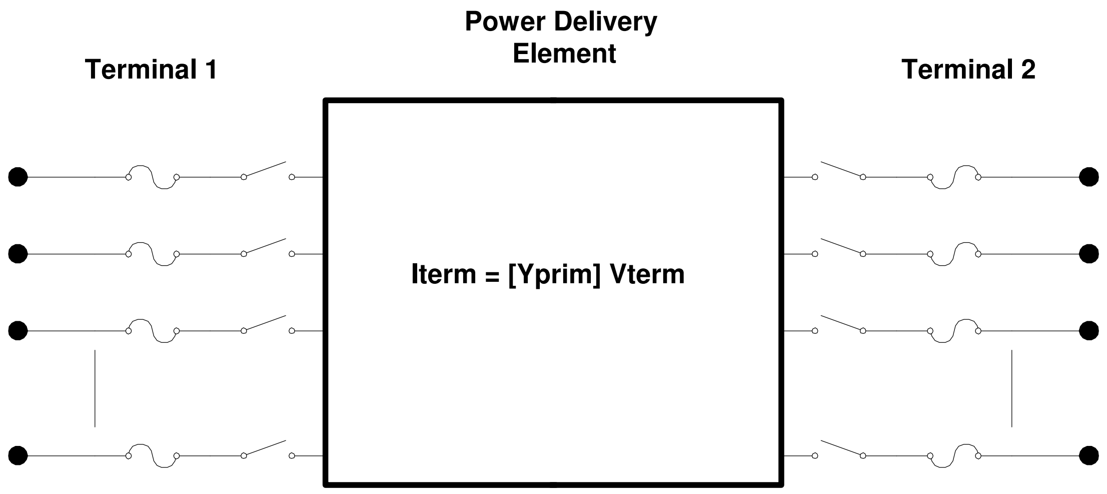


Figure 10: Power Delivery Element Definition

Power Conversion Elements

Power conversion elements convert power from electrical form to some other form, or vice-versa. Some may temporarily store energy and then give it back, as is the case for reactive elements. Most will have only one connection to the power system and, therefore, only one multiphase terminal. The description of the mechanical or thermal side of the power conversion is contained within the "Black box" model. The description may be a simple impedance or a complicated set of differential equations yielding a current injection equation of the form:

$$I_{Term}(t) = \mathbf{F}(V_{Term}, [State], t)$$

The function **F** will vary according to the type of simulation being performed. The power conversion element must also be capable of reporting the partials matrix when necessary:

$$\frac{\partial F}{\partial V}$$

In simple cases, this will simply be the primitive **y** (admittance) matrix; that is, the y matrix for this element alone.

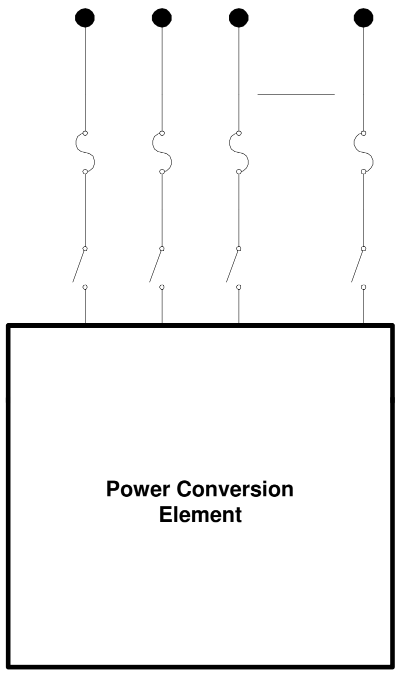


Figure 11: Power Conversion Element Definition

Within the DSS, the typical implementation of a PC element is as shown in Figure 12. Nonlinear elements, in particular Load and Generator elements, are treated as Norton equivalents with a constant Yprim and a “compensation” current or injection current that compensates for the nonlinear portion. This works well for most distribution loads and allows a wide range of models of load variation with voltage. It converges well for the vast majority of typical distribution system conditions. The Yprim matrix is generally kept constant for computational efficiency, although the DSS does not require this. This limits the number of times the system Y matrix has to be rebuilt, which contributes greatly to computational efficiency for long runs, such as annual loading simulations.

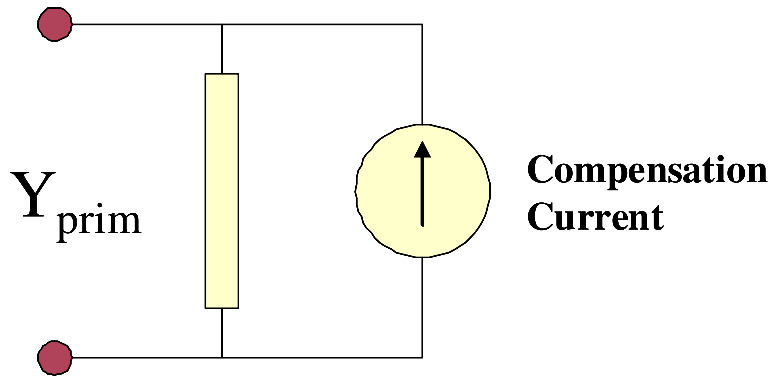


Figure 12. Compensation Current Model of PC Elements (one-line)

The Compensation current is the current that is added into the injection current vector in the main solver (see next section). This model accommodates a large variety of load models quite easily. The load models presently implemented are:

1. Conventional constant P, Q load model. This, and all load models, revert to linear (constant Z) model outside the normal voltage range in an attempt to guarantee convergence even when the voltage drops very low. This is important for performing annual simulations.

2. Constant Z load model. P and Q vary by the square of the voltage. Will usually converge in any condition.

3. Constant P but Q is modeled as a constant reactance.

4. P and Q variation defined by exponential values. For example this is used for Conservation Voltage Reduction (CVR) studies.

5. Constant current magnitude (common in distribution system analysis). P and Q vary linearly with voltage magnitude.

6. Constant P that can be modified by loadshape multipliers, but Q is a fixed value independent of time.

7. Similar to 6, but Q is computed from a fixed reactance (varies with square of voltage)

Except where noted, P and Q can be modified by Loadshape multipliers. Loads can also be exempt from loadshape mulitipliers.

Putting it All Together

Figure 13 illustrates how the DSS puts all the PD elements and PC elements together to perform a solution.

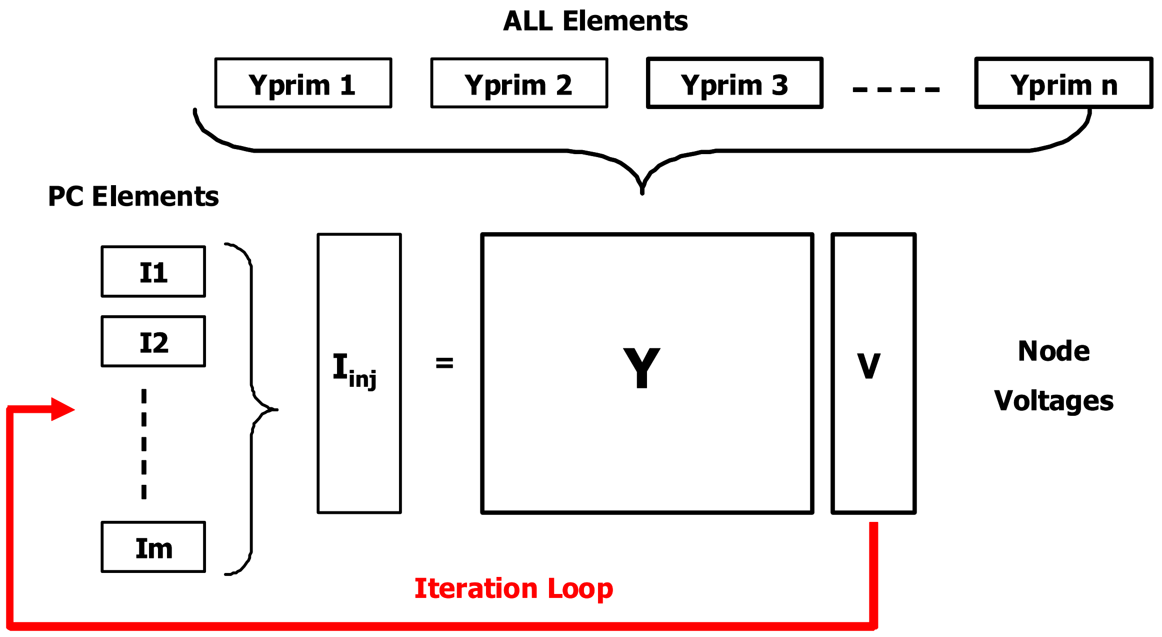


Figure 13. OpenDSS Solution Loop

Basically, the upper structure of the OpenDSS (the part that is written in Delphi) manages the creation and modification of the primitive y matrices (Yprim) for each element in the circuit as well as managing the bus lists, the collection of results through Meter elements, and the execution of Control elements. The Yprim matrices are fed to the sparse matrix solver, which constructs the system Y matrix.

An initial guess at the voltages is obtained by performing a zero load power flow. That is, all shunt elements are disconnected and only the series power delivery elements are considered. This gets all the phase angles and voltage magnitudes in the proper relationship. This is important because the DSS is designed to solve arbitrary n-phase networks in which there can be all sorts of transformer ratios and connections.

The iteration cycle begins by obtaining the injection currents from all the power conversion (PC) elements in the system and adding them into the appropriate slot in the I_{inj} vector. The sparse set is then solved for the next guess at the voltages. The cycle repeats until the voltages converge to typically 0.0001 pu. The system Y matrix is typically not rebuilt during this process so the iterations go quickly.

This simple iterative solution has been found to converge quite well for most distribution systems that have adequate capacity to serve the load. The key is to have a dominant bulk power source, which is the case for most distribution systems. In the DSS, this is the “Normal” solution algorithm. There is also a “Newton” algorithm for more difficult systems (not to be confused with the typical Newton-Raphson power flow method).

When performing Daily or Yearly simulations, the solution at the present time step is used as the starting point for the solution at the next time step. Unless there is a large change in load, the solution will typically converge in 2 iterations – one to do the solution and one to check it to make sure it is converged. Thus, the DSS is able to perform such calculations quite efficiently.

Control iterations are performed outside this loop. That is, a converged solution is achieved before checking to see if control actions are required.

DSS Command Language Syntax

The DSS is designed such that all functions can be carried out through text-based DSS Command Language scripts. The text streams may come from any of these sources:

1. Selecting and executing a script on a Control Panel window,
2. Through the COM interface, or
3. From a standard text file to which the command interpreter may be temporarily redirected (Compile or Redirect commands).

This makes the DSS an easily accessible tool for users who simply want to key in a small circuit and do a quick study as well as to those who perform quite complicated studies. It also makes the DSS more easily adapted by others who have a great deal invested in their own database and would put forth much effort to conform to another. Text files are the most common means of transferring data from one source into another. The DSS scripts can often be configured to be close to various data transfer formats.

Always refer to the **Help** command for the latest commands and property names that are recognized by the DSS.

COMMAND SYNTAX

The command language is of the form:

Command parm1, parm2 parm3 parm 4 ….

Parameters (parm1, etc) may be separated by commas (,) or white space (blank, tab). If a parameter includes a delimiter, enclose it in either double quotes ("), single quotes(') or parentheses (…). While any of these will work, double or single quotes are preferred for strings. Brackets [..] are preferred for arrays, and curly braces {..} are preferred for in-line math.

PARAMETERS

Parameters may be *positional* or *named* (tagged). If named, an "=" sign is expected.

1. *Name=value* (this is the named form)
2. *Value* (value alone in positional form)

For example, the following two commands are equivalent.

```
New   Object="Line.First Line"   Bus1=b1240   Bus2=32   LineCode=336ACSR, …

New   "Line.First Line", b1240   32   336ACSR, …
```

The first example uses named parameters, which are shown in the default order. The second example simply gives the values of the parameters and the parser assumes that they are in the default order. Note that the name of the object contains a blank, which is a standard DSS delimiter character. Therefore, it is enclosed in quotes or parentheses.

You may mix named parameters and positional parameters. Using a named parameter repositions the parser's positional pointer so that subsequent parameters need not be named. The order of parameters is always given in the DSS help command window. The DSS help window allows for display of commands and properties in either alphabetical order or positional order. The properties are by default processed in positional order.

Some commands are interpreted at more than one lexical level inside the DSS. In this example, the main DSS command interpreter interprets the **New** command and essentially passes the remainder of the string to the Executive for adding new circuit elements. It determines the type of element to add (e.g., a Line) and confirms that it is indeed a registered class. It then passes the remainder of the text line on to the module that handles the instantiation and definitions of Line objects. Only the Line model code needs to know how to interpret the parameters it receives. This design allows great flexibility for future modifications to the Line model that might require adding new properties. This happens with regularity and the main DSS executive does not have to be modified; only the Line code module.

For the **New** command, the first two parameters are always required and positional:

1. The New command itself, and
2. The name of the object to add.

For circuit elements, the next one or two parameters are normally the bus connections, which are processed and stored with the circuit element model. Then the definition of the object being created continues, using an editing function expressly devoted to that class of circuit element.

Delimiters and Other Special Characters

The DSS recognizes these delimiters and other special characters:

Array or string delimiter pairs:	[ ] { } ( ) “ “ ‘ ‘
Matrix row delimiter:	|
Value delimiters:	, any white space
Class, Object, Bus, or Node delimiter:	.
Keyword / value separator:	=

Continuation of previous line:	~
Comment line:	//
In-line comment:	!
Query a property:	?

ARRAY PARAMETERS AND QUOTE PAIRS

Array parameters are sequences of numbers. Of necessity, delimiters must be present to separate the numbers. To define an array, simply enclose the sequence in any of the quote pairs. Square brackets, [..], are the preferred method, although any of the other quote pairs: "..", '..', (..), {..} will work. For example, for a 3-winding transformer you could define arrays such as:

```
kvs = [115, 6.6, 22]

kvas=[20000  16000 16000]
```

MATRIX PARAMETERS

Matrix parameters are entered by extending the Array syntax: Simply place a bar (|) character between rows, for example:

```
Xmatrix=[1.2  .3  .3 | .3  1.2  3 | .3  .3  1.2]   !(3x3 matrix)
```

Symmetrical matrices like this example may also be entered in lower triangle form to be more compact.

```
Xmatrix=[ 1.2  | .3 1.2  | .3  .3  1.2 ]   !(3x3 matrix – lower triangle)
```

STRING LENGTH

The DSS Command strings may be as long as can be reasonably passed through the COM interface. This is very long. They must generally NOT be split into separate "lines" since there is no concept of a line of text in the standard DSS COM interface; only separate commands. However, New and Edit commands can be continued on a subsequent text line with the More or ~ command.

In general, string values in DSS scripts can be as long as desired. There are, of course, limitations to what can be reasonably deciphered when printed on reports. Also, the DSS "hashes" bus names, device names and any other strings that are expected to result in long lists for large circuits. This is done for fast searching. At present, hashing is done on only the first 8 characters. Therefore, it is advantageous to make names mostly unique in the first 8 characters if the circuit is large. This does not mean that comparisons are only 8 characters; comparisons are done on full string lengths.

Abbreviations are allowed by default in DSS commands and element property names. However, if the circuit is very large, script processing will proceed more efficiently if the names are spelled out completely. It won't matter on small circuits. The reason is that the hash lists are much shorter than the linear lists. If the DSS does not find the abbreviated name in a hash list, it will then search the whole list from top to bottom. This can slow performance if there are thousands of names in the list.

DEFAULT VALUES

When a New command results in the instantiation of a DSS element, the element is instantiated with reasonable values. Only those properties that need to be changed to adequately define the object need be included in the command string. Commonly, only the bus connections and a few key properties need be defined. Also, a new element need not be defined in one command. It may be edited as many times as desired with subsequent commands (see Edit, More and ~ commands).

When an element is created or selected by a command, it becomes the *Active* element. Thereafter, property edit commands are passed directly to the active element until another element is defined or selected. In that respect, the command language mirrors the basic COM interface.

All changes are persistent. That is, a parameter changed with one command remains as it was defined until changed by a subsequent command. This might be a source of misunderstanding with novices using the program who might expect values to reset to a base case as they do in some other programs. When this is a concern, issue a “clear” command and redefine the circuit from scratch.

IN-LINE MATH

The DSS scripting language uses a form of reverse Polish notation (RPN) to accommodate in-line math. This feature may be used to pre-process input data before simulation. The parser will evaluate RPN expressions automatically if you enclose the expression in any of the quoted formats (quotation marks or any of the matched parentheses, brackets, etc.).

RPN Expressions

Basically, you enter the same keystrokes as you would with an RPN calculator (such as an HP 48). One difference is that you separate numbers, operators, and functions by a space or comma. The RPN calculator in DSS has a "stack" of 10 registers just like an HP calculator. In the function descriptions that follow, the first stack register is referred to as "X" and the next higher as "Y" as on the HP calculator. When a new number is entered, the existing registers are rolled up and the new number is inserted into the X register.

These functions operate on the first two registers, X and Y:

+	Add the last two operands (X and Y registers; result in X; X = X + Y)
-	X = Y - X
*****	X = X * Y

/	X = Y / X
^	(exponentiation) X = Y to the X power

These unitary functions operate on the X register and leave the result in X.

sqr	X=X*X
sqrt	take the square root of X
inv	(inverse of X = 1/X)
ln	(natural log of X)
exp	(e to the X)
log10	(Log base 10 of X)
sin	for X in degrees, take the sine
cos	for X in degrees, take the cosine
tan	for X in degrees, take the tangent
asin	take the inverse sine, result in degrees
acos	take the inverse cosine, result in degrees
atan	take the arc tangent, result in degrees
atan2	take the arc tangent with two arguments, Y=rise and X = run, result in degrees over all four quadrants

The following functions manipulate the stack of registers

Rollup	shift all registers up
Rolldn	shift all registers dn
Swap	swap X and Y

There is currently one constant preprogrammed: **pi**

RPN Examples

For example, the following DSS code will calculate X1 for a 1-mH inductance using inline RPN:

```
// convert 1 mH to ohms at 60 Hz, note the last * operator
line.L1.X1 = (2 pi * 60 * .001 *)
```

The expression in parentheses is evaluated left-to-right. '2' is entered followed by 'pi'.

Then the two are multiplied together yielding 2π. The result is then multiplied by 60 to yield ω (2πf). Finally, the result is multiplied by 1 mH to yield the reactance at 60 Hz. To specify the values of an array using in-line math, simply nest the quotes or parentheses:

```
// Convert 300 kvar to 14.4 kV, 2 steps
Capacitor.C1.kvar = [(14.4 13.8 / sqr 300 *), (14.4 13.8 / sqr 300 *)]
```

The Edit | RPN Evaluator menu command brings up a modal form in which you can enter an RPN expression and compute the result. Clicking the OK button on this form automatically copies the result to the clipboard. The purpose of this feature is to enable you to interpret RPN strings you find in the DSS script or to simplify the above script by computing the result and replacing the RPN string with the final value if you choose:

```
Capacitor.C1.kvar = [ 326.65, 326.65] ! 300 kvar converted to 14.4 kV, 2 steps
```

This next example shows how to use RPN expressions inside an array. Two different delimiter types are necessary to differentiate the array from the expressions.

```
// set the winding kvs to (14.4 20)
New Transformer.t  kvs=("24.9 3 sqrt /" "10 2 *")
```

DSS Command Reference

Nearly all DSS commands and parameter names may be abbreviated. This is for convenience when typing commands in directly. However, there is not necessarily any speed benefit to abbreviating for machine-generated text. (See String Length above.) Thus, commands and parameter names should be spelled out completely when placed in script files that are auto-generated from other computer data sources. Abbreviate only when manually typing commands to the DSS.

Specifying Objects

Any object in the DSS, whether a circuit element or a general DSS object, can be referenced by its complete name:

```
Object=Classname.objname
```

For example,

```
object=vsource.source.
```

In nearly all circumstances, the 'object=' may be omitted as it can for any other command line parameter. The object name is almost always expected to be the first parameter immediately following the command verb.

If the 'classname' prefix is omitted (i.e., no dot in the object name), the previously used class (the active DSS class) is assumed. For example,

```
New line.firstline . . .

New secondline . . .
```

The second command will attempt to create a new line object called 'secondline'.

If there is any chance the active class has been reset, use the fully qualified name. Alternatively, use the Set Class command to establish the active class:

```
Set class=Line

New secondline . . .
```

COMMAND DEFINITIONS

The following documents the command definitions as of this writing. Newer builds of the DSS may have additional properties and commands. Execute the Help command while running the DSS to view the present commands available in your version.

Help

Brings up a help window with a "tree" view of all the commands and property names currently accepted by the DSS. (This is generally more up-to-date than the printed manuals because new features are being continually added.) Clicking on the commands will display a brief description of how to use the particular command or property.

// (comment) and ! (inline comment)

The appearance of "//" in the command position indicates that this statement is a comment line. It is ignored by the DSS. If you wish to place an in-line comment at the end of a command line, use the **"!"** character. The parser ignores all characters following the ! character.

// This is a comment line

New line.line4 linecode=336acsr length=2.0 **! this is an in-line comment**

New [Object] [Edit String]

Adds an element described on the remainder of the line to the active circuit. The first parameter (Object=…) is required for the New command. Of course, "Object=" may be omitted and generally is for aesthetics.

The remainder of the command line is processed by the editing function of the specified element type. All circuit objects are instantiated with a reasonable set of values so that they can likely be included in the circuit and solved without modification. Therefore, the Edit String need only include definitions for property values that are different than the defaults.

Examples:

```
New Object=Line.Lin2    ! Min required

New Line.Lin2              ! Same, sans object= …

!Line from Bs1 to Bs2
New Line.Lin2 Bs1  Bs2  R1=.01 X1=.5 Length=1.3
```

The Edit String does not have to be complete at the time of issuing the New command. The object instantiated may be edited at any later time by invoking the Edit command or by continuing with the next command line (see More or ~).

Immediately after issuing the New command, the instantiated object remains the active object and the Edit command does not have to be given to select the object. You can simply issue the More command, or one of its abbreviations (~), and continue to send editing instructions. Actually, the DSS command interpreter defaults to editing mode and you may simply issue the command

Property=value …(and other editing statements)

The "=" is required. When the DSS parser sees this, it will assume you wish to continue editing and are not issuing a separate DSS Command. To avoid ambiguity, you may specify the element completely:

Class.ElementName.Property = Value

More than one property may be set on the same command, just as if you had issued the New or Edit commands.

Note that all DSS objects have a **Like** property inherited from the base class. Users are somewhat at the mercy of developers to ensure they have implemented the Like feature, but this has been done on DSS objects to date. . When another element of the same class is very similar to a new one being created, use the Like parameter to start the definition then change only the parameters that differ. Issue the Like=nnnn property first. Command lines are parsed from left to right with the later ones taking precedence. Throughout the DSS, the design goal is that a property persistently remains in its last state until subsequently changed.

```
New Line.Lin3 like=Lin2 Length=1.7
```

Edit [Object] [Edit String]

Edits the object specified. The object Class and Name fields are required and must designate a valid object (previously instantiated by a New command) in the problem. Otherwise, nothing is done and an error is posted.

The edit string is passed on to the object named to process. The DSS main program does not attempt to interpret property values for circuit element classes. These can and do change periodically.

More | M | ~ | [Edit String]

The More command continues editing the active object last selected by a New, Edit, or Select command. It simply passes the edit string to the object's editing function (actually, the active object's editing function simply takes control of the parser after it is determined that this is a More command).

The More command may be abbreviated with simply **M** or ~.

Examples:

```
!Line from Bs1 to Bs2
New Line.Lin2 Bs1  Bs2
```

```
More R1=.01 X1=.5 Length=1.3

!Line from Bs1 to Bs2
New Line.Lin2 Bs1  Bs2  R1=.01 X1=.5 Length=1.3
M   C1=3.4  R1 = .02     ! define C1 and re-define R1

!Line from Bs1 to Bs2
New Object=Line.Lin2
~ Bus1=Bs1  Bus2=Bs2.2.3.1    ! Transposition line
~ R1=.01 X1=.5
~ Length=1.3
```

? [Object Property Name]

The "?" command allows you to query the present value of any published property of a DSS circuit element. For example,

? Monitor.mon1.mode

is one way to get the mode defined for the Monitor named Mon1.

The value is returned in the "Result" property of the DSS COM Text interface (See COM interface). It also appears in the Result window of the standalone executable immediately after execution of the command.

[Object Property Name] = value

This syntax permits the setting of any published property value of a DSS circuit element. Simply specify the complete element and property name, "=", and a value. For example,

Monitor.mon1.mode=48

Sets the monitor mode queried in the previous example. The DSS command interpreter defaults to the Edit command. If it does not recognize the command it looks for the "=" and attempts to edit a property as specified. Thus, in this example, this method simply invokes the Monitor object's editor and sets the value. If there is more text on the string, the editor continues editing. For example,

Line.line1.R1=.05   .12 .1   .4

will set the R1, X1, R0, X0 properties of Line.line1 in sequence using positional property rules. This is a convenient syntax to use to change properties in circuit elements that have already been defined.

Disable [Object]

Disables object in active circuit. All objects are **Enabled** when first defined. Use this command if you wish to temporarily remove an object from the active circuit, for a contingency case, for example. If this results in isolating a portion of the circuit, the voltages for those buses will be computed to be zero. (Also see

Open, Close commands.)

Enable [Object]

Cancels a previous **Disable** command. All objects are automatically Enabled when first defined. Therefore, the use of this command is unnecessary until an object has been first disabled. (Also see Open, Close commands.)

Open [Object] [Term] [Cond]

Opens a specified terminal conductor switch. All conductors in the terminals of all circuit elements have an inherent switch. This command can be used to open one or more conductors in a specified terminal. If the 'Cond=' field is 0 or omitted, all *phase* conductors are opened. Any other conductors there might be are unaffected. Otherwise, open one conductor at a time (one per command). For example:

```
Open object=line.linxx  term=1      ! opens all phase conductors of terminal 1 of
linxx

Open line.linxx 2 3     ! opens 3rd conductor of 2nd terminal of linxx line object

Open load.LD3  1 4     ! opens neutral conductor of wye-connected 3-phase load
```

No action is taken if either the terminal or conductor specifications are invalid.

Note, this action disconnects the terminal from the node to which it is normally connected. The node remains in the problem. If it becomes isolated, a tiny conductance is attached to it and the voltage computes to zero. But you don't have to worry about it breaking the DSS.

Close [Object] [Term] [Cond]

Opposite of Open command.

Compile or Redirect [fileName]

The Compile and Redirect commands simply redirect the command interpreter to take input directly from a text file rather than from the Command property of the COM Text interface (the default method of communicating with the DSS). In DSS convention, use Compile when defining a new circuit and Redirect within script files to signify that the output is redirected to another file temporarily for the purpose of nesting script files. Use exactly the same syntax that you would supply to the Command property. Do not span lines; there is no line concept with the DSS commands. Each command must be on its own line in the file. Use the More command or its abbreviation "~" to continue editing a new line.

Set [option1=value1] [option2=value2] (Options)

The Set command sets various global variables and options having to do with solution modes, user interface issues, and the like. Works like the Edit command except that you don't specify object type and name.

Properties that can be set with this command are:

%growth = Set default annual growth rate, percent, for loads with no growth curve specified. Default is 2.5.

%mean = Percent mean to use for global load multiplier. Default is 65%.

%Normal = Sets the Normal rating of all lines to a specified percent of the emergency rating. Note: This action takes place immediately. Only the in-memory value is changed for the duration of the run.

%stddev = Percent Standard deviation to use for global load multiplier. Default is 9%.

Addtype = {Generator | Capacitor} Default is Generator. Type of device for AutoAdd Mode.

Algorithm = {Normal | Newton} Solution algorithm type. Normal is a fixed point current-injection iteration that is a little quicker (about twice as fast) than the Newton iteration. Normal is adequate for most distribution systems. Newton is more robust for circuits that are difficult to solve.

AllocationFactors = Sets all allocation factors for all loads in the active circuit to the value given. Useful for making an initial guess or forcing a particular allocation of load. The allocation factors may be set automatically by the energy meter elements by placing energy meters on the circuit, defining the PEAKCURRENT property, and issuing the ALLOCATELOADS command.

AllowDuplicates = {YES/TRUE | NO/FALSE} Default is No. Flag to indicate if it is OK to have devices of same name in the same class. If No, then a New command is treated as an Edit command but adds an element if it doesn't exist already. If Yes, then a New command will always result in a device being added.

AutoBusList = Array of bus names to include in AutoAdd searches. Or, you can specify a text file holding the names, one to a line, by using the syntax (file=filename) instead of the actual array elements. Default is null, which results in the program using either the buses in the EnergyMeter object zones or, if no EnergyMeters, all the buses, which can make for lengthy solution times.

Examples:

```
Set autobuslist=(bus1, bus2, bus3, ... )
```

Set autobuslist=(file=buslist.txt)

Basefrequency = Default = 60. Set the fundamental frequency for harmonic solution and the default base frequency for all impedance quantities. Side effect: also changes the value of the solution frequency.

Bus = Set Active Bus by name. Can also be done with Select and SetkVBase commands and the "Set Terminal=" option. The bus connected to the active terminal becomes the active bus. See Zsc and Zsc012 commands.

capkVAR = Size of capacitor, kVAR, to automatically add to system. Default is 600.0.

casename = Name of case for yearly simulations with demand interval data. Becomes the name of the subdirectory under which all the year data are stored. Default = circuit name

Side Effect: Sets the prefix for output files

circuit = Set the active circuit by name. (Current version allows only one circuit at a time.)

CktModel = {Multiphase | Positive} Default = Multiphase. Designates whether circuit model is to interpreted as a normal multi-phase model or a positive-sequence only model. If Positive sequence, all power quantities are mulitiplied by 3 in reports and through any interface that reports a power quantity.

Class = Synonym for **Type**=. sets class (type) for the Active DSS Object. This becomes the Active DSS Class.

ControlMode = {OFF | STATIC |EVENT | TIME} Default is "STATIC". Control mode for the solution. Set to OFF to prevent controls from changing.

> STATIC = Time does not advance. Control actions are executed in order of shortest time to act until all actions are cleared from the control queue. Use this mode for power flow solutions which may require several regulator tap changes per solution.
>
> EVENT = solution is event driven. Only the control actions nearest in time are executed and the time is advanced automatically to the time of the event.
>
> TIME = solution is time driven. Control actions are executed when the time for the pending action is reached or surpassed.

Controls may reset and may choose not to act when it comes their time to respond.

Use TIME mode when modeling a control externally to the DSS and a solution mode such as DAILY or DUTYCYCLE that advances time, or set the time (hour and sec) explicitly from the external program.

Datapath = Set the data path for files written or read by the DSS. Defaults to the startup path. May be Null. Executes a CHDIR to this path if non-null. Does not require a circuit defined. You can also use the "cd" command from a script.

DefaultDaily = Default daily load shape name. Default value is "default", which is a 24-hour curve defined when the DSS is started.

DefaultYearly = Default yearly load shape name. Default value is "default", which is a 24-hour curve defined when the DSS is started. If no other curve is defined, this curve is simply repeated when in Yearly simulation mode.

DemandInterval = {YES/TRUE | NO/FALSE} Default = no. Set for keeping demand interval data for daily, yearly, etc, simulations. Side Effect: Resets all meters!!!

DIVerbose = {YES/TRUE | NO/FALSE} Default = FALSE. Set to Yes/True if you wish a separate demand interval (DI) file written for each meter. Otherwise, only the totalizing meters are written.

Editor= Set the command string required to start up the editor preferred by the user. Defaults to Notepad. This is used to display certain reports from the DSS. Note: on many Windows systems, Wordpad.exe is not on the path. However, Write frequently is and will start up Wordpad. Therefore, you may say "Editor=Write" if you wish to use Wordpad. Otherwise, you must locate Wordpad.exe and use the complete path name. Use the complete path name for any other Editor. Does not require a circuit defined.

Element = Sets the active DSS element by name. You can use the complete object spec (class.name) or just the name. if full name is specifed, class becomes the active class, also.

Emergvmaxpu = Maximum permissible per unit voltage for emergency (contingency) conditions. Default is 1.08.

Emergvminpu = Minimum permissible per unit voltage for emergency (contingency) conditions. Default is 0.90.

Frequency = sets the frequency for the next solution of the active circuit.

Genkw = Size of generator, kW, to automatically add to system. Default is 1000.0

GenMult = Global multiplier for the kW output of every generator in the circuit. Default is 1.0. Applies to Snapshot, Daily, and DutyCycle solution modes. Ignored if generator is designated as Status=Fixed.

Genpf = Power factor of generator to assume for automatic addition. Default is 1.0.

H = Alternate name for time step size (see Stepsize).

Harmonics = {ALL | (list of harmonics) } Default = ALL. Array of harmonics for which to perform a solution in Harmonics mode. If ALL, then solution is performed for all harmonics defined in spectra currently being used. Otherwise, specify a more limited list such as:

```
Set Harmonics=(1 5 7 11 13)
```

Hour= sets the hour to be used for the start time of the solution of the active circuit. (See also Time)

KeepList = Array of bus names to keep when performing circuit reductions. You can specify a text file holding the names, one to a line, by using the syntax (file=filename) instead of the actual array elements. Command is cumulative (reset keeplist first). Reduction algorithm may keep other buses automatically.

Examples:

```
Reset Keeplist        (sets all buses to FALSE (no keep))

Set KeepList=(bus1, bus2, bus3, ... )

Set KeepList=(file=buslist.txt)
```

LDcurve = name of the Loadshape object to use for the global circuit Load-Duration curve. Used in solution modes LD1 and LD2 (see below). Must be set before executing those modes. Simply define the load-duration curve as a loadshape object. Default = nil.

LoadModel= {"POWERFLOW" | "ADMITTANCE"} Sets the load model. If POWERFLOW (abbreviated P), loads do not appear in the System Y matrix. For iterative solution types (Mode ≠ Direct) loads (actually all PC Elements) are current injection sources. If ADMITTANCE, all PC elements appear in the System Y matrix and solution mode should be set to Direct (below) because there will be no injection currents.

LoadMult = global load multiplier to be applied to all "variable" loads in the circuit for the next solution. Loads designated as "fixed" are not affected. Note that not all solution modes use this multiplier, but many do, including all snapshot modes. See Mode below. The default LoadMult value is 1.0. Remember that it remains at the last value to which it was set. Solution modes such as Monte Carlo and Load-Duration modes will alter this multiplier. Its value is usually posted on DSS control panels. Loads defined with "status=fixed" are not affected by load multipliers. (The default for loads is "status=variable".)

Log = {YES/TRUE | NO/FALSE} Default = FALSE. Significant solution events are added to the Event Log, primarily for debugging.

LossRegs = Which EnergyMeter register(s) to use for Losses in AutoAdd Mode. May be one or more registers. if more than one, register values are summed together. Array of integer values > 0. Defaults to 13 (for Zone kWh Losses).

For a list of EnergyMeter register numbers, do the "Show Meters" command after defining a circuit.

LossWeight = Weighting factor for Losses in AutoAdd functions. Defaults to 1.0. Autoadd mode minimizes

(Lossweight * Losses + UEweight * UE).

If you wish to ignore Losses, set to 0. This applies only when there are EnergyMeter objects. Otherwise, AutoAdd mode minimizes total system losses.

Markercode = Number code for node marker on circuit plots (SDL MarkAt options).

Maxcontroliter = Max control iterations per solution. Default is 10.

Maxiter = Sets the maximum allowable iterations for power flow solutions. Default is 15.

Mode= specify the solution mode for the active circuit. Mode can be one of (unique abbreviation will suffice, as with nearly all DSS commands):

Snap: Solve a single snapshot power flow for the present conditions. Loads are modified only by the global load multiplier (LoadMult) and the growth factor for the present year (Year).

Daily: Do a series of solutions following the daily load curves. The Stepsize defaults to 3600 sec (1 hr). Set the starting hour and the number of solutions (e.g., 24) you wish to execute. Monitors are reset at the beginning of the solution. The peak of the daily load curve is determined by the global load multiplier (LoadMult) and the growth factor for the present year (Year).

Direct: Solve a single snapshot solution using an admittance model of all loads. This is non-iterative; just a direct solution using the currently specified voltage and current sources.

Dutycycle: Follow the duty cycle curves with the time increment specified. Perform the solution for the number of times specified by the Number parameter (see below).

FaultStudy: Do a full fault study solution, determining the Thevenin equivalents for each bus in the active circuit. Prepares all the data required to produce fault study report under the Show Fault command.

Yearly: Do a solution following the yearly load curves. The solution is repeated as many times as the specified by the Number= option. Each load then follows its yearly load curve. Load is determined solely by the yearly load curve and the growth multiplier. The time step is always 1 hour. Meters and Monitors are reset at the beginning of solution and sampled after each solution. If the yearly load curve is not specified, the daily curve is used and simply repeated if the number of solutions exceeds 24 hrs. This mode is nominally designed to support 8760-hr simulations of load, but can be used for any simulation that uses an hourly time step and needs monitors or meters.

LD1 (Load-Duration Mode 1): Solves for the joint union of a load-duration curve (defined as a Loadshape object) and the Daily load shape. Nominally performs a Daily solution (24-hr) for each point on the Load-duration (L-D) curve. Thus, the time axis of the L-D curve represents *days* at that peak load value. L-D curves begin at zero (0) time. Thus, a yearly L-D curve would be defined for 0..365 days. A monthly L-D curve should be defined for 0..31 days. Energy meters and monitors are reset at the beginning of the solution. At the conclusion, the energy meter values represent the total of all solutions. If the L-D curve represent one year, then the energy will be for the entire year. This mode is intended for those applications requiring a single energy number for an entire year, month, or other time period. Loads are modified by growth curves as well, so set the year before proceeding. Also, set the L-D curve (see Ldcurve option).

LD2 (Load-Duration Mode 2): Similar to LD1 mode except that it performs the Load-duration solution for only a selected hour on the daily load shape. Set the desired hour before executing the Solve command. The meters and monitors are reset at the beginning of the solution. At the conclusion, the energy meters have only the values for that hour for the year, or month, or whatever time period the L-D curve represents. The solver simply solves for each point on the L-D curve, multiplying the load at the selected hour by the L-D curve value. This mode has been used to generate a 3-D plot of energy vs. month and hour of the day.

M1 (Monte Carlo Mode 1): Perform a number of solutions allowing the loads to vary randomly. Executes number of cases specified by the Number option (see below). At each solution, each load is modified by a random multiplier -- a different one for each load. In multiphase loads, all phases are modified simultaneously so that the load remains balanced. The random variation may be uniform or gaussian as specified by the global Random option (see below). If uniform, the load multipliers are between 0 and 1. If gaussian, the multipliers are based on the mean and standard deviation of the Yearly load shape specified for the load. Be sure one is specified for each load.

M2 (Monte Carlo Mode 2): This mode is designed to execute a number of Daily simulations with the global peak load multiplier (LoadMult) varying randomly. Set time step size (h) and Number of solutions to run. "h" defaults to 3600 sec (1 Hr). Number of solutions refers to the number of *DAYS*. For Random = Gaussian, set the global %Mean and %Stddev variables, e.g. "Set %Mean=65 %Stddev=9". For Random=Uniform, it is not necessary to specify %Mean, since the global load multiplier is varied from 0 to 1. For each day, the global peak load multiplier is generated and then a 24 hour Daily solution is performed at the specified time step size.

M3 (Monte Carlo Mode 3): This mode is similar to the LD2 mode except that the global load multiplier is varied randomly rather than following a load-duration curve. Set the Hour of the day first (either Set Time=… or Set Hour=…). Meters and monitors are reset at the beginning of the solution. Energy at the conclusion of the solution represents the total of all random solutions. For example, one might use this mode to estimate the total annual energy at a given hour by running only 50 or 100 solutions at each hour (rather than 365) and ratioing up for the full year.

MF (Monte Carlo Fault mode). One of the faults defined in the active circuit is selected and its resistance value randomized. All other Faults are disabled. Executes number of cases specified by the Number parameter.

Peakdays: Do daily solutions (24-hr) only for those days in which the peak exceeds a specified value.

Nodewidth = Width of node marker in circuit plots. Default=1.

Normvmaxpu = Maximum permissible per unit voltage for normal conditions. Default is 1.05.

Normvminpu = Minimum permissible per unit voltage for normal conditions. Default is 0.95.

Number= specify the number of time steps or solutions to run or the number of Monte Carlo cases to run.

Object (or **Name)=** sets the name of the Active DSS Object. Use the complete object specification (*classname.objname*), or simply the *objname,* to designate the active object which will be the target of the next command (such as the More command). If 'classname' is omitted, you can set the class by using the Class= field.

Overloadreport = {YES/TRUE | NO/FALSE} Default = FALSE. For yearly solution mode, sets overload reporting on/off. DemandInterval must be set to true for this to have effect.

PriceCurve = Sets the curve to use to obtain for price signal. Default is none (null string). If none, price signal either remains constant or is set by an external process. Curve is defined as a loadshape (not normalized) and should correspond to the type of analysis being performed (daily, yearly, load-duration, etc.).

PriceSignal = Sets the price signal ($/MWh) for the circuit. Initial value is 25.

Random= specify the mode of random variation for Monte Carlo studies: One of [Uniform | Gaussian | Lognormal | None] for Monte Carlo Variables May abbreviate value to "G", “L”, or "U" where **G** = gaussian (using mean and std deviation for load shape); **L** = lognormal (also using mean and std dev); **U** = uniform (varies between 0 and 1 randomly). Anything else: randomization disabled.

Recorder = {YES/TRUE | NO/FALSE} Default = FALSE. Opens DSSRecorder.DSS in DSS install folder and enables recording of all commands that come through the text command interface. Closed by either setting to NO/FALSE or exiting the program. When closed by this command, the file name can be found in the Result. Does not require a circuit defined.

ReduceOption = { Default or [null] | Stubs [Zmag=nnn] | MergeParallel | BreakLoops | Switches | TapEnds [maxangle=nnn] | Ends} Strategy for reducing feeders. Default is to eliminate all dangling end buses and buses without load, caps, or taps.

"Stubs [Zmag=0.02]" merges short branches with impedance less than Zmag (default = 0.02 ohms

"MergeParallel" merges lines that have been found to be in parallel

"Breakloops" disables one of the lines at the head of a loop.

"Tapends [maxangle=15]" eliminates all buses except those at the feeder ends, at tap points and where the feeder turns by greater than maxangle degrees.

"Ends" eliminates dangling ends only.

"Switches" merges switches with downline lines and eliminates dangling switches.

Marking buses with "Keeplist" will prevent their elimination.

Sec = sets the seconds from the hour for the start time for the solution of the active circuit. (See also Time)

Stepsize (or **h**)= sets the time step size (in sec) for the solution of the active circuit. Normally specified for dynamic solution, but also used for duty-cycle load following solutions. Yearly simulations typically go hour-by-hour and daily simulations follow the smallest increment of the daily load curves. (Note: defaults to 0.001 sec! This is appropriate for dynamics, but not for duty cycle)

Terminal = Set the active terminal of the active circuit element. May also be done with Select command.

Time= Specify the solution start time as an array : time="hour, sec" or time = (hour, sec): e.g., time = (23, 370) designate 6 minutes, 10 sec past the 23rd hour.

tolerance= Sets the solution tolerance. Default is 0.0001.

TraceControl = {YES/TRUE | NO/FALSE} Set to YES to trace the actions taken in the control queue. Creates a file named TRACE_CONTROLQUEUE.CSV in the default directory. The names of all circuit elements taking an action are logged.

Trapezoidal = {YES/TRUE | NO/FALSE} Default is "No". Specifies whether to use trapezoidal integration for accumulating energy meter registers. Applies to EnergyMeter and Generator objects. Default method simply multiplies the present value of the registers times the width of the interval. Trapezoidal is more accurate when there are sharp changes in a load shape or unequal intervals. Trapezoidal is automatically used for some load-duration curve simulations where the interval size varies considerably. Keep in mind that for Trapezoidal, you have to solve one more point than the number of intervals. That is, to do a Daily simulation on a 24-hr load shape, you would set Number=25 to force a solution at the first point again to establish the last (24th) interval.

Type=Sets the active DSS class type. Same as **Class**=...

Ueregs = Which EnergyMeter register(s) to use for UE in AutoAdd Mode. May be one or more registers. if more than one, register values are summed together. Array of integer values > 0. Defaults to 11 (for Load EEN).

For a list of EnergyMeter register numbers, do the "Show Meters" command after defining a circuit.

Ueweight= Weighting factor for UE/EEN in AutoAdd functions. Defaults to 1.0.

Autoadd mode minimizes

```
(Lossweight * Losses + UEweight * UE).
```

If you wish to ignore UE, set to 0. This applies only when there are EnergyMeter objects. Otherwise, AutoAdd mode minimizes total system losses.

Voltagebases=Define legal bus voltage bases for this circuit. Enter an array of the legal voltage bases, in phase-to-phase voltages, for example:

```
set voltagebases=[.208, .480, 12.47, 24.9, 34.5, 115.0, 230.0]
```

When the CalcVoltageBases command is issued, a snapshot solution is performed with no load injections and the bus base voltage is set to the nearest legal voltage base. The defaults are as shown in the example above.

The DSS does not use per unit values in its solution. You only need to set the voltage bases if you wish to see per unit values on the reports or if you intend to use the AutoAdd feature.

Voltexceptionreport={YES/TRUE | NO/FALSE} Default = FALSE. For yearly solution mode, sets voltage exception reporting on/off. DemandInterval must be set to true for this to have effect.

Year= sets the Year to be used for the next solution of the active circuit; the base case is year 0. Used to determine the growth multiplier for each load. Each load may have a unique growth curve (defined as a Growthshape object).

ZoneLock = {YES/TRUE | NO/FALSE} Default is No. if No, then meter zones are recomputed each time there is a change in the circuit. If Yes, then meter zones are not recomputed unless they have not yet been computed. Meter zones are normally recomputed on Solve command following a circuit change.

There are new Set command values added periodically. Check the Help on the DSS while it is running.

Get [Opt1] [opt2] etc.

Basically, the opposite of the SET command. Returns DSS property values set using the Set command. Result is returned in the Result property of the Text interface.

VBA Example:

```
DSSText.Command = "Get mode"

Answer = DSSText.Result
```

Multiple properties may be requested on one get. The results are appended and the individual values separated by commas. Array values are returned separated by commas.

See help on Set command for property names.

Reset {Meters | Monitors }

{Monitors | Meters | (no argument) } Resets all Monitors or Energymeters as specified. If no argument specified, resets meters and monitors.

Show <Quantity>

Writes a text file report of the specified quantity for the most recent solution and opens a viewer (the default Editor -- e.g., Notepad or some other editor) to display the file. Defaults to Show Voltages

Quantity can be one of:

Currents - Shows the currents into each device terminal.

Monitor <monitor name> - Shows a text (CSV) file with the voltages and currents presently stored in the specified monitor.

Faults - Shows results of Faultstudy mode solution: all-phase, one-phase, and adjacent 2-phase fault currents at each bus.

Elements - Shows all the elements in the active circuit.

Buses Shows all buses in the active circuit.

Panel - Same as Panel Command. Opens the internal DSS control panel.

Meter - shows the present values in the energy meter registers in the active circuit.

Generators - Each generator has its own energy meter. Shows the present values in each generator energy meter register in the active circuit.

Losses Loss summary.

Powers [MVA|kVA*] [Seq* | Elements] Show the power flow in various units. Default (*) is sequence power in kVA.

Voltages [LL |LN*] [Seq* | Nodes | Elements]. Shows the voltages in different ways. Default (*) is sequence quantities line-to-neutral.

Zone EnergyMeterName [Treeview] Different ways to show the selected energymeter zone.

AutoAdded (see AutoAdd solution mode)

Taps shows the taps on regulated transformers

Overloads Overloaded PD elements report

Unserved [UEonly] Unserved energy report. Loads that are unserved.

EVentlog Show the event log (capacitor switching, regulator tap changes)

Variables Show state variable values in present circuit.

Isolated Show isolated buses and circuit sections

Ratings Device rating report.

Loops Shows where loops occur in present circuit

Yprim (shows Yprim for active ckt element)

Y (shows the system Y matrix)

BusFlow busname [MVA|kVA*] [Seq* | Elements]. Power flow report centered on specified bus. Can show it by sequence quantities or by elements (detailed report).

LineConstants [frequency] [none|mi|km|kft|m|ft|in|cm] Show line constants results, at the specified frequency and normalized length, for all the presently-defined LineGeometry objects.

Export <Quantity> [Filename or switch]

Writes a text file (.CSV) of the specified quantity for the most recent solution Defaults to Export Voltages. The purpose of this command is to produce a file that is readily readable by other programs such as those written in VB, spreadsheet programs, or database programs.

The first record is a header record providing the names of the fields. The remaining records are for data. For example, the voltage export looks like this:

```
Bus, Node Ref., Node, Magnitude, Angle, p.u., Base kV
sourcebus   ,   1,    1,  6.6395E+0004,    0.0,    1.000,   115.00
sourcebus   ,   2,    2,  6.6395E+0004, -120.0,    1.000,   115.00
sourcebus   ,   3,    3,  6.6395E+0004,  120.0,    1.000,   115.00
subbus      ,   4,    1,  7.1996E+0003,   30.0,    1.000,    12.47
subbus      ,   5,    2,  7.1996E+0003,  -90.0,    1.000,    12.47
subbus      ,   6,    3,  7.1996E+0003,  150.0,    1.000,    12.47
```

This format is common for many spreadsheets and databases, although databases may require field types and sizes for direct import. The columns are aligned for better readability.

Valid syntax for the command can be one of the following statement prototypes in bold. If the Filename is omitted, the file name defaults to the name shown in italics in parantheses.

Export Voltages [Filename] ***(EXP_VOLTAGES.CSV)***. Exports voltages for every bus and active node in the circuit. (Magnitude and angle format).

Export SeqVoltages [Filename] ***(EXP_SEQVOLTAGES.CSV)*** Exports the sequence voltage magnitudes and the percent of negative- and zero-sequence to positive sequence.

Export Currents [Filename] ***(EXP_CURRENTS.CSV)*** Exports currents in magnitude and angle for each phase of each terminal of each device.

Export Overloads [Filename] ***EXP_OVERLOADS.CSV)*** Exports positive sequence current for each device and the percent of overload for each power delivery element that is overloaded.

Export SeqCurrents [Filename] ***(EXP_SEQCURRENTS.CSV)*** Exports the sequence currents for each terminal of each element of the circuit.

Export Powers [MVA] [Filename] ***(EXP_POWERS.CSV)*** Exports the powers for each terminal of each element of the circuit. If the MVA switch is specified, the result are specified in MVA. Otherwise, the results are in kVA units.

Export Faultstudy [Filename] ***(EXP_FAULTS.CSV)*** Exports a simple report of the 3-phase, 1-phase and max L-L fault at each bus.

Export Loads [Filename] ***(EXP_LOADS.CSV)*** Exports the follow data for each load object in the circuit: Connected KVA, Allocation Factor, Phases, kW, kvar, PF, Model.

Export Monitors monitorname ***(file name is assigned)*** Automatically creates a separate filename for each monitor. Exports the monitor record corresponding the monitor's mode. This will vary for different modes.

Export Meters [Filename | /multiple] ***(EXP_METERS.CSV)***
Export Generators [Filename | /multiple] ***(EXP_GENMETERS.CSV)***
EnergyMeter and Generator object exports are similar. Both export the time and the values of the energy registers in the two classes of objects. In contrast to the other Export options, each invocation of these export commands **appends** a record to the file. For Energymeter and Generator, specifying the switch "/multiple" (or /m) for the file name will cause a separate file to be written for each meter or generator. The default is for a single file containing all meter or generator elements.

Export Yprims [Filename] (EXP_Yprims.CSV) . Exports all primitive Y matrices for the present circuit to a CSV file.

Export Y [Filename] (EXP_Y.CSV) . Exports the present system Y matrix to a CSV file. Useful for importing into another application.

Solve [see set command options ...]

Executes the solution mode specified by the Set Mode = command. It may execute a single solution or hundreds of solutions. The Solution is a DSS object associated with the active circuit. It has several properties that you may set to define which solution mode will be performed next. This command invokes the Solve method of the Solution object, which proceeds to execute the designated mode. You may also specify the Mode and Number options directly on the Solve command line if you wish: Solve Mode=M1 Number=1000, for example. Note that there is also a Solve method in the Solution interface in the DSS COM implementation. This is generally faster when driving the DSS from user-written code for a custom solution algorithm that must execute many solutions.

You may use the same options on the Solve command line as you can with the Set command. In fact, the Solve command is simply the Set command that performs a solution after it is done setting the options. For example:

```
Solve mode=daily   algorithm=newton
```

Dump <Circuit Element> [Debug]

Writes a text file showing all the properties of the circuit object and displays it with the DSS text editor. You would use this command to check the definition of elements. It is also printed in a format that would allow it to be fed back into the DSS with no, or only minor, editing. If the "dump" command is used without an object reference, all elements in the active circuit are dumped to a file, which could be quite voluminous.

If the Debug option is specified, considerably more information is dumped including the primitive Y matrices and other internal DSS information.

You can limit the dump to all elements of a specific class by using the wildcard, *, character: For example,

```
Dump Transformer.* debug
```

Will dump all information on all transformers.

Clear

Clears all circuit element definitions from the DSS. This statement is recommended at the beginning of all master files for defining DSS circuits.

About

Displays the "About" box. The Result string is set to the version string.

CalcVoltageBases

Calculates the voltage base for each bus based on the array of voltage bases defined with "Set Voltagebases=..." command. Performs a zero-current power flow considering only the series power-delivery elements of the system. No loads, generators, or shunt elements are included in the solution. The voltage base for the bus is then set to the nearest voltage base specified in the voltage base array. Alternatively, you may use the SetkVBase command to set the voltage base for each bus individually. The DSS does not need the voltage base for most calculations, but uses it for reporting. The exception is the AutoAdd solution mode where the program needs to specify the voltage rating of capacitors and generators it is adding to the system. Also, some controls may need the base voltage to work better.

It is useful to show the bus voltages after the execution of this command. This will confirm that everything in the circuit is connected as it should be.

SetkVBase

Command to explicitly set the base voltage for a bus. Bus must be previously defined. This will override the definitions determined by CalcVoltageBases. When there are only a few voltage bases in the problem, and they are very distinct, the CalcvoltageBases command will work nearly every time. Problems arise if there are two voltage bases that are close together, such as 12.47 kV and 13.2 kV. Use a script composed of SetkVBase commands to remove ambiguity.

Parameters in order are:

Bus = {bus name}

kVLL = (line-to-line base kV)

kVLN = (line-to-neutral base kV)

kV base is normally given in line-to-line kV (phase-phase). However, it may also be specified by line-to-neutral kV. The following exampes are equivalent:

```
setkvbase Bus=B9654 kVLL=13.2

setkvbase B9654 13.2

setkvbase B9654 kvln=7.62
```

BuildY

Forces rebuild of Y matrix upon next Solve command regardless of need. The usual reason for doing this would be to reset the matrix for another load level when using LoadModel=PowerFlow (the default), if the system is difficult to solve when the load is far from its base value. Some of the load elements will recompute their primitive Y to facilitate convergence. Works by invalidating the Y primitive matrices for all the Power Conversion elements.

Init

This command forces reinitialization of the solution for the next Solve command. To minimize iterations, most solutions start with the previous solution unless there has been a circuit change. However, if the previous solution is bad, it may be necessary to re-initialize. In most cases, a re-initialization results in a zero-load power flow solution with only the series power delivery elements considered.

Fileedit [filename]

Edit specified file in default text file editor (see Set Editor= option).

```
Fileedit EXP_METERS.CSV     (brings up the meters export file)
```

"FileEdit" may be abbreviated to a unique character string.

Voltages

Returns the voltages for the ACTIVE TERMINAL ONLY of the active circuit element in the Result string. For setting the active circuit element, see the Select command or the Set Terminal= property. Returned as magnitude and angle quantities, comma separated, one set per conductor of the terminal.

Currents

Returns the currents for each conductor of ALL terminals of the active circuit element in the Result string. (See Select command.) Returned as comma-separated magnitude and angle.

Powers

Returns the powers (complex) going into each conductors of ALL terminals of the active circuit element in the Result string. (See Select command.) Returned as comma-separated kW and kvar.

SeqVoltages

Returns the sequence voltages at all terminals of the active circuit element (see Select command) in Result string. Returned as comma-separated magnitude only values.Order of returned values: 0, 1, 2 (for each terminal).

SeqCurrents

Returns the sequence currents into all terminals of the active circuit element (see Select command) in Result string. Returned as comma-separated magnitude only values. Order of returned values: 0, 1, 2 (for each terminal).

SeqPowers

Returns the sequence powers into all terminals of the active circuit element (see Select command) in Result string. Returned as comma-separated kw, kvar pairs.Order of returned values: 0, 1, 2 (for each terminal).

Losses

Returns the **total** losses for the active circuit element (see Select command) in the Result string in kW, kvar.

PhaseLosses

Returns the losses for the active circuit element (see Select command) for each PHASE in the Result string in comma-separated kW, kvar pairs.

CktLosses

Returns the total losses for the active circuit in the Result string in kW, kvar.

AllocateLoads

Estimates the allocation factors for loads that are defined using the XFKVA property. Requires that energymeter objects be defined with the PEAKCURRENT property set. Loads that are not in the zone of an energymeter cannot be allocated. This command adjusts the allocation factors for the appropriate loads until the best match possible to the meter values is achieved. Loads are adjusted by phase. Therefore all single-phase loads on the same phase will end up with the same allocation factors.

If loads are not defined with the XKVA property, they are ignored by this command.

Plot (options …)

Plot is a rather complex command that displays a variety of results in a variety of manners on graphs. You should use the control panel to execute the plot command with the recorder on to see examples of how to construct the plot command. Implemented options (in order):

Type = {Circuit | Monitor | Daisy | Zones | AutoAdd | General (bus data) } A Circuit plot requires that the bus coordinates be defined. By default the thickness of the circuit lines is drawn proportional to power. A Daisy plot is a special circuit plot that shows a unique symbol for generators. (When there are many generators at the same bus, the plot resembles a daisy.) The Monitor plot plots one or more channels from a Monitor element. The Zones plot draws the energymeter zones. Autoadd shows autoadded elements on the circuit plot. General expects a CSV file of bus data with bus name and a number of values. Specify which value to plot in Quantity= property. Bus colors are interpolated based on the specification of C1, C2, and C3.

Quantity = {Voltage | Current | Power | Losses | Capacity | (Value Index for General, AutoAdd, or Circuit[w/ file]) } Specify quantity or value index to be plotted.

Max = {0 | value corresponding to max scale or line thickness}

Dots = {Y | N} Turns on/off the dot symbol for bus locations on the circuit plot.

Labels = {Y | N} Turns on/off the bus labels on the circuit plot (with all bus labels displayed, the plot can get cluttered – zoom in to see individual names)

Object = [metername for Zone plot | Monitor name | File Name for General bus data or Circuit branch data]. Specifies what object to plot: meter zones, monitor or CSV file.

ShowLoops = {Y | N} (default=N). Shows the loops in meter zone in red. Note that the DSS has no problem solving loops; This is to help detect unintentional loops in radial circuits.

R3 = pu value for tri-color plot max range [.85] (Color C3)

R2 = pu value for tri-color plot mid range [.50] (Color C2)

C1, C2, C3 = {RGB color number}. Three color variables used for various plots.

Channels = (array of channel numbers for monitor plot). More than one monitor channel can be plotted on the same graph.

Bases = (array of base values for each channel for monitor plot). Default is 1.0 for each. Set Base= **after** defining channels.

Subs = {Y | N} (default=N) (show substations) See Transformer definition

Thickness = max thickness allowed for lines in circuit plots (default=7). Useful for controlling aesthetics on circuit plots.

Loadshapes may be plotted from the control panel of the user interface.

Power and Losses are specified in kW. C1 used for default color. C2, C3 used for gradients, tri-color plots. Scale determined automatically of Max = 0 or not specified. Examples:

```
Plot daisy power 5000 dots=N
Plot circuit quantity=7 Max=.010 dots=Y Object=branchdata.csv
Plot General Quantity=2 Object=valuefile.csv
```

AutoAdd and General Plots

You can make plots that show values computed for buses in a couple of different ways. If you are auto adding generators, the DSS keeps an "DSS_autoaddlog.csv" file. You can plot the values in the columns of this file by saying something like this (the DSS user interface will help you construct this command if you turn on the recorder):

```
plot type=Auto quantity=3 Max=2000 dots=n labels=n C1=16711680 C2=8421376 C3=255 R3=0.75 R2=0.5
```

Max doesn't mean anything here, but the DSS will throw it in.

Quantity=3: Column 3 is % improvement in losses. The DSS will put colored marker circles on the node points, with one of the 3 colors indicated. First the min and max values in the file is determined to set the range of the plot. Then the file is read in again and the node markers are added to the plot in the order they are encountered in the file. If the value is greater than fraction of the total range designated by R3, color C3 is used. If between R2 and R3, then C2 is used, else C1 is used. In this example, C1 is blue, c2 is green and C3 is red.

If you have a lot of overlapping nodes, you may wish to sort the CSV file (use Excel) on the column of interest before plotting. Usually, you will want to sort the column ascending. That way, in the example, the C3 nodes will show up clearly on a background of C2 and C1 respectively. That's usually what you want.

The Autoadd plot is a special case of the General plot. Here is an example of a General plot.

```
plot General quantity=1 Max=2000 dots=n labels=n object=(C:\Projects\PosSeqModelNoSw2\16-500KW-amps_Fault.csv) C1=16777088 C2=255
```

Note that this only uses two colors and an object must be specified. This is the file name containing at least two columns: the bus name followed by one or more columns of values. In this case, the first value is used. Again, the max= is superfluous.

For a General plot, the values are depicted as a transition from color C1 to C2. C2 represents the high end of the range and C1 is the low end. The colors are interpolated maintaining their RGB proportions. Generally, you'll want C2 to be some dark color such as solid red and C1 a lighter color. And, if the node markers overlap, it is generally best to sort the file you are plotting in ascending order before executing this command.

General Circuit Plots

If you wish to generate a circuit plot with the width of the lines controlled by something other than voltage, current, losses, power, or capacity, create a csv file with the first column being the full name of the line and one or more columns of values. Set the Quantity= property to the value column of interest. Specify the filename in the Object= property.

The program will generate a circuit plot of the quantity specified with the width being proportional to the value compared to the Max= property.

Interpolate {All | MeterName}

Default is "All". Interpolates coordinates for missing bus coordinates in meter zones. Fills in coordinates evenly spaced between existing coordinates.

CD Directoryname

Change to the specified directory.

CloseDI

Close all DemandInterval (DI) files. This must be issued at the end of a yearly solution where DI files are left open. Reset and Set Year=nnn will also close the DI files. See Set DemandInterval option. DI files remain open once the yearly simulation begins to allow for changes and interactions from outside programs during the simulation. They must be closed before viewing the results.

DI_plot

[case=]casename [year=]yr [registers=](reg1, reg2,...) [peak=]y/n [meter=]metername

Plots demand interval (DI) results from yearly simulation cases. Plots selected registers from selected meter file (default = DI_Totals.CSV). Peak defaults to NO. If YES, only daily peak of specified registers is plotted. Example:

```
 DI_Plot basecase year=5 registers=(9,11) no
```

Totals

Total of all EnergyMeter objects in the circuit. Reports register totals in the result string.

Comparecases

[Case1=]casename [case2=]casename [register=](register number) [meter=]{Totals* | SystemMeter | metername}.

Compares yearly simulations of two specified cases with respect to the quantity in the designated register from the designated meter file. Defaults: Register=9 meter=Totals. Example:

```
Comparecases base pvgens 10
```

YearlyCurves

[cases=](case1, case2, ...) [registers=](reg1, reg2, ...) [meter=]{Totals* | SystemMeter | metername}

Plots yearly curves for specified cases and registers.

Default: meter=Totals. Example:

```
yearlycurves cases=(basecase, pvgens) registers=9
```

Ysc

Returns full short circuit admittance, Ysc, matrix for the ACTIVE BUS in comma-separated complex number form G + jB.

Zsc

Returns full Short circuit impedance, Zsc, matrix for the ACTIVE BUS in comma-separated complex number form.

Zsc10

Returns symmetrical component short-circuit impedances, Z1, Z0 for the ACTIVE BUS in comma-separated R+jX form.

ZscRefresh

Refreshes Zsc matrix for the ACTIVE BUS.

DOScmd /c *...command string ...*

Execute a DOS command from within the DSS. Closes the window when the command is done. To keep the DOS window open, use /k switch.

Varnames

Returns variable names for active element if PC element. Otherwise, returns null.

VarValues

Returns variable values for active element if PC element. Otherwise, returns null.

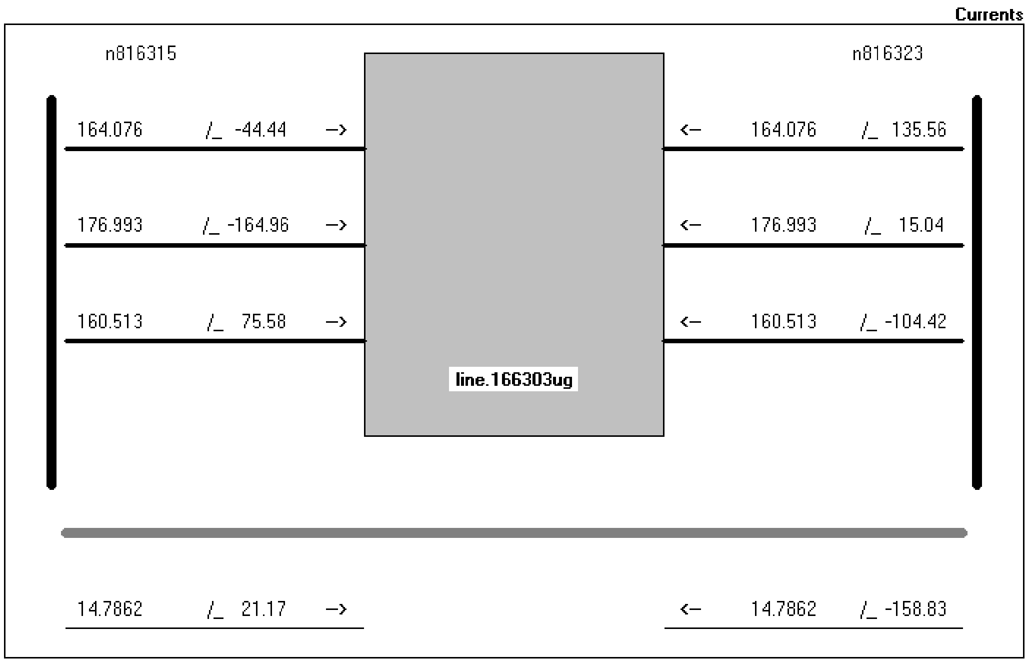
Vdiff

Displays the difference between the present solution and the last one saved using the SAVE VOLTAGES command.

Visualize

[What=] {Currents* | Voltages | Powers} [element=]full_element_name (class.name).

Shows the currents, voltages, or powers for selected element on a drawing in phasor quantities. For example:



Check the on-line Help while the DSS is running for information on additional commands.

General DSS Object Property Descriptions

These are objects common to all circuits in the DSS (there may be several circuits loaded into memory at any one time). Any circuit can reference the data contained in the General DSS Objects

The following describes each object class and the parameters that may be defined through the command interface.

Note that all DSS objects have a Like parameter. When another element of the same class is very similar to a new one being created, use the Like parameter to start the definition then change the parameters that differ. Issue the Like=nnnn parameter first. Command lines are parsed and executed from left to right.

LINECODE

Linecodes are objects that contain impedance characteristics for lines and cables. The term "line code" is an old term that simply refers to a code that was made up by programmers to describe a line construction. In most distribution analysis programs, one can describe a line by its linecode and its length. Linecodes were defined in a separate file. This collection of objects emulates the old linecode files, except that the concept is slightly more powerful.

Ultimately, the impedance of a line is described by its series impedance matrix and nodal capacitive admittance matrix These matrices may be specified directly or they can be generated by specifying the symmetrical component data. Note that the impedances of lines may be specified directly and one does not need to use a line code, although the linecode will be more convenient most of the time. There may be hundreds of lines, but only a few different kinds of line constructions.

LineCode also performs a Kron reduction, reducing out the last conductor in the impedance matrices, which is assumed to be a neutral conductor. This applies only if the impedance is specified as a matrix. If the impedance is defined as symmetrical components, this function does not apply because symmetrical component values already assume the reduction.

By specifying the values of Rg, Xg, and rho, the DSS will take the base frequency impedance matrix values and adjust the earth return component for frequency. Skin effect in the conductors is not modified. To represent skin effect, you have to define the geometry.

This assumes the impedance matrix is constructed as follows:

$$Z = R + jX = \begin{bmatrix} Z_{11} + Zg & Z_{12} + Zg & Z_{13} + Zg \\ Z_{21} + Zg & Z_{22} + Zg & Z_{23} + Zg \\ Z_{31} + Zg & Z_{32} + Zg & Z_{33} + Zg \end{bmatrix}$$

Where, using the 1st term of Carson's formulation,

$$Rg = \mu_0 \frac{\omega}{8} \quad \Omega / m$$

$$Xg = \mu_0 \frac{\omega}{2\pi} \ln\left(658.5\sqrt{\frac{\rho}{f}} \right) \quad \Omega / m$$

$$Xii = \mu_0 \frac{\omega}{2\pi} \ln\left(\frac{1}{GMRi} \right) \quad \Omega / m$$

$$Xij = \mu_0 \frac{\omega}{2\pi} \ln\left(\frac{1}{dij} \right) \quad \Omega / m$$

$$\mu_0 = 4\pi \times 10^{-7}$$

(All dimensional values in meters.)

The LineCode editing parameters, in order, are: (not case sensitive)

Nphases= Number of phases. Default = 3.

R1 = Positive-Sequence resistance, ohms per unit length.

X1 = Positive-Sequence reactance, ohms per unit length.

R0 = Zero-Sequence resistance, ohms per unit length.

X0 = Zero-Sequence reactance, ohms per unit length.

C1= Positive-Sequence capacitance, nanofarads per unit length

C0= Zero-Sequence capacitance, nanofarads per unit length

Units= {mi | km | kft | m | ft | in | cm} Length units. If not specified, it is assumed that the units correspond to the length being used in the Line models.

If any of the Symmetrical Component data is specified, the impedance matrices are determined from that data. You may also enter the matrices directly using the following three parameters. The order of matrices expected is the number of phases. The matrices may be entered in lower triangle form or full matrix. The result is always symmetrical. Matrices are specified by the syntax shown below. The "|" separates rows. Extraneous numbers on each row are ignored.

```
Rmatrix="11 | 21 22 | 31 32 33"
```

Rmatrix= Series resistance matrix, ohms per unit length..

Xmatrix= Series reactance matrix, ohms per unit length.

Cmatrix= Shunt nodal capacitance matrix, nanofarads per unit length.

BaseFreq= Base Frequency at which the impedance values are specified. Default = 60.0 Hz.

Normamps= Normal ampacity, amps.

Emergamps= Emergency ampacity, amps.

Faultrate= Number of faults per year per unit length. This is the default for this general line construction.

Pctperm= Percent of the fault that become permanent (requiring a line crew to repair and a sustained interruption).

Kron= Y/N. Default=N. Perform Kron reduction on the impedance matrix after it is formed, reducing order by 1. Do this only on initial definition after matrices are defined. Ignored for symmetrical components.

Rg= Carson earth return resistance per unit length used to compute impedance values at base frequency. For making better frequency adjustments. Default=0

Xg= Carson earth return reactance per unit length used to compute impedance values at base frequency. For making better frequency adjustments. Default=0

Rho= Earth resistivity used to compute earth correction factor. Default=100 meter ohms.

Like= Name of an existing LineCode object to build this like.

WIREDATA

This class of data defines the raw conductor data that is used to compute the impedance for a line geometry.

Note that you can use whatever units you want for any of the dimensional data – be sure to declare the units. Otherwise, the units are all assumed to match, which would be very rare for conductor data. Conductor data is usually supplied in a hodge-podge of units. Everything is converted to meters internally to the DSS

Rdc= dc Resistance, ohms per unit length (see Runits). Defaults to Rac if not specified.

Rac= Resistance at 60 Hz per unit length. Defaults to Rdc if not specified.

Runits= Length units for resistance: ohms per {mi|kft|km|m|Ft|in|cm } Default=none.

GMRac= GMR at 60 Hz. Defaults to .7788*radius if not specified.

GMRunits= Units for GMR: {mi|kft|km|m|Ft|in|cm } Default=none.

Radius= Outside radius of conductor. Defaults to GMR/0.7788 if not specified.

Radunits= Units for outside radius: {mi|kft|km|m|Ft|in|cm } Default=none.

Normamps= Normal ampacity, amperes. Defaults to Emergency amps/1.5 if not specified.

Emergamps= Emergency ampacity, amperes. Defaults to 1.5 * Normal Amps if not specified.

Diam= Diameter; Alternative method for entering radius.

Like= Make like another object, e.g.:

New Capacitor.C2 like=c1 ...

LINEGEOMETRY

This class of data is used to define the positions of the conductors.

Nconds= Number of conductors in this geometry. Default is 3. Triggers memory allocations. Define first!

Nphases= Number of phases. Default =3; All other conductors are considered neutrals and might be reduced out.

Cond= Set this to number of the conductor you wish to define. Default is 1.

Wire= Code from WireData. MUST BE PREVIOUSLY DEFINED. no default.

X= x coordinate.

H= Height of conductor.

Units= Units for x and h: {mi|kft|km|m|Ft|in|cm } Initial default is "ft", but defaults to last unit defined

Normamps= Normal ampacity, amperes for the line. Defaults to first conductor if not specified.

Emergamps= Emergency ampacity, amperes. Defaults to first conductor if not specified.

Reduce= {Yes | No} Default = no. Reduce to Nphases (Kron Reduction). Reduce out neutrals.

Like= Make like another object, e.g.:

New Capacitor.C2 like=c1 ...

LINE CONSTANT EXAMPLES

Define the wire data:

```
New Wiredata.ACSR336  GMR=0.0255000 DIAM=0.7410000 RAC=0.3060000
~  NormAmps=530.0000
~  Runits=mi radunits=in gmrunits=ft
New Wiredata.ACSR1/0  GMR=0.0044600 DIAM=0.3980000 RAC=1.120000
~   NormAmps=230.0000
~   Runits=mi radunits=in gmrunits=ft
```

Define the Geometry data:

```
New Linegeometry.HC2_336_1neut_0Mess  nconds=4 nphases=3
~ cond=1 Wire=acsr336  x=-1.2909 h=13.716 units=m
~ cond=2 Wire=acsr336  x=-0.502    h=13.716 !units=m
~ cond=3 Wire=acsr336  x=0.5737   h=13.716 !units=m
~ cond=4 Wire= ACSR1/0  x=0           h=14.648 ! units=m  ! neutral
```

Define a 300-ft line section:

```
New Line.Line1 Bus1=xxx   Bus2=yyy
~   Geometry= HC2_336_1neut_0Mess
~    Length=300   units=ft
```

Check out the line constants at 60 and 600 Hz in per km values. This command shows line constants for all defined geometries:

```
Show lineconstants freq=60 units=km
Show lineconstants freq=600 units=km
```

If the number of conductors = 3, this Show command will also give you the sequence impedances. If your geometry has more than 3 conductors and you want to see the sequence impedance, define

```
nconds = whatever
nphases = 3
Reduce=Yes
```

This will force the impedance matrices to be reduced to 3x3 and the Show LineConstants command will automatically give the sequence impedances. Note: make sure the phase conductors are defined first in the geometry definition.

No automatic assembly of bundled conductors is available yet. However, you can specifically define the position of each conductor in the geometry definition and connect them up explicitly in the DSS, for example, for a two conductor bundle:

```
New Line.2Bundled bus1=FromBus.1.1.2.2.3.3  ToBus.1.1.2.2.3.3
~ Geometry=2BundleGeometry  Length= etc. etc.
```

LOADSHAPE

A LoadShape object consists of a series of multipliers, nominally ranging from 0.0 to 1.0 that are applied to the base kW values of the load to represent variation of the load over some time period.

Load shapes are generally fixed interval, but may also be variable interval. For the latter, both the time, hr, and the multiplier must be specified.

All loadshapes, whether they be daily, yearly, or some arbitrary duty cycle, are maintained in this class. Each load simply refers to the appropriate shape by name.

The loadshape arrays may be entered directly in command line, or the load shapes may be stored in one of three different types of files from which the shapes are loaded into memory.

The command parameters for LoadShape objects are:

Npts= Number of points to expect when defining the curve

Interval= time interval of the data, Hr. Default=1.0. If the load shape has non-uniformly spaced points, specify the interval as 0.0.

Mult= Array of multiplier values. Looking for Npts values. To enter an array, simply enclose a series of numbers in double quotes "…", single quotes '…', or parentheses(..). Omitted values are assumed to be zero. Extra values are ignored. You may also use the syntax: **mult=(file=filename.ext)** in which the array values are entered one per line in the text file referenced.

Hour= Array of hour values corresponding to the multipliers. Not required if Interval>0. You may also use the syntax: **hour=(file=filename.ext)** in which the hour array values are entered one per line in the text file referenced. Again, this is not required for fixed interval load shape curves.

Mean= Mean of the multiplier array. The mean and standard deviation <u>are always computed</u> after an array of points are entered or normalized (see below). However, if you are doing only parametric load studies using the Monte Carlo solution mode, only the Mean and Std Deviation are required to define a loadshape. These two values may be defined directly rather than by supplying the curve. Of course, the multiplier points are not generated.

Stddev= Standard Deviation (see Mean, above).

The next three parameters instruct the LoadShape object to get its data from a file. Three different formats are allowed. If Interval>0 then only the multiplier is entered. For variable interval data, set Interval=0.0 and enter both the time (in hours) and multiplier, in that order for each interval.

Csvfile= Name of a CSV file containing load shape data, one interval to a line. For variable interval data enter one (hour, multiplier) point to a line with the values separated by a comma. Otherwise there is simply one value to a line.

Sngfile= Name of a binary file of single-precision floating point values containing the load shape data. The file is packed. For fixed interval data, the multipliers are packed in order. For variable interval data, start with the first hour point and alternate with the multiplier value.

Dblfile= Name of a binary file of double-precision floating point values containing the load shape data. The file is packed. For fixed interval data, the multipliers are packed in order. For variable interval data, start with the first hour point and alternate with the multiplier value.

Action= { Normalize} Many times the raw load shape data is in actual kW or some other unit. The load shapes normally will have a maximum value of 1.0. Specifying this parameter as "Action=N" *after* the load shape multiplier data are imported will force the normalization of the data in memory and recalculation of the mean and standard deviation.

Like= Name of an existing loadshape object to base this one on.

GrowthShape

A GrowthShape object is similar to a Loadshape object. However, it is intended to represent the growth in load year-by-year and the way the curve is specified is entirely different. You must enter the growth for the first year. Thereafter, only the years where there is a change must be entered. Otherwise it is assumed the growth stays the same.

Growth rate is specified by specifying the *multiplier* for the previous year's load. Thus, if the load grows 2.5% in 1999, the multiplier for that year will be specified as 1.025.

(If no growth shape is specified, the load growth defaults to the circuit's default growth rate -- see **Set %Growth** command).

The parameters are:

Npts= Number of points to expect when defining the curve.

Year= Array of year values corresponding to the multiplier values. Enter only those years in which the multiplier changes. Year data may be any integer sequence -- just so it's consistent for all growth curves. Setting the global solution variable Year=0 causes the growth factor to default to 1.0, effectively neglecting growth. This is what you would do for all base year analyses.

You may also use the syntax: **year=(file=filename.ext)** in which the array values are entered one per line in the text file referenced.

Mult= Array of Multiplier values corresponding to the year values. Enter multiplier by which the load will grow in this year.

Example:

```
New growthshape.name npts=3 year="1999 2003 2007" mult="1.10  1.05  1.025"
```

This defines a growthshape the a 10% growth rate for 1999-2002. Beginning in 2003, the growth tapers off to 5% and then to 2.5% for 2007 and beyond.

Normally, only a few points need be entered and the above parameters will be quite sufficient. However, provision has been made to enter the (year, multiplier) points from files just like the LoadShape objects. You may also use the syntax: **mult=(file=filename.ext)** in which the array values are entered one per line in the text file referenced.

Csvfile= Name of a csv file containing one (year, mult) point per line. Separate the year from the multiplier by a comma.

Sngfile= Name of a file of single-precision numbers containing (year, mult) points packed.

Dblfile= Name of a file of single-precision numbers containing (year, mult) points packed.

Like= Name of an existing GrowthShape object to base this one on.

TCC_CURVE

A TCC_Curve object is defined similarly to Loadshape and Growthshape objects in that they all are defined by curves consisting of arrays of points. Intended to model time-current characteristics for overcurrent relays, TCC_Curve objects are also used for other relay types requiring time curves. Both the time array and the C array must be entered.

The parameters are:

Npts= Number of points to expect when defining the curve.

C_Array= Array of current (or voltage or whatever) values corresponding to time values in T_Array (see T_Array).

T_Array= Array of time values in sec. Typical array syntax:

t_array = (1, 2, 3, 4, ...)

You may also substitute a file designation: **t_array = (file=filename).** The specified file has one value per line.

Like= Name of an existing GrowthShape object to base this one on.

DSS Circuit Element Object Descriptions

The following are descriptions of key DSS circuit elements. This should be enough to get you started with DSS simulations. New circuit element classes may be added at any time. Check the on-line help for brief help on new elements.

VSOURCE OBJECT

Voltage source. This is a special power conversion element. It is special because voltage sources must be identified to initialize the solution with all other injection sources set to zero.

A Vsource object is simply a multi-phase Thevenin equivalent with data specified as it would commonly be for a power system source equivalent: Line-line voltage (kV) and short circuit MVA.

The properties are, in order:

Bus1= Name of bus to which the source's one terminal is connected. Remember to specify the node order if the terminals are connected in some unusual manner.

basekv= base or rated Line-to-line kV.

pu= Actual per unit at which the source is operating. Assumed balanced for all phases.

Angle= Base angle, degrees, of the first phase.

Frequency= frequency of the source.

Phases= Number of phases. Default = 3.0.

Mvasc3= 3-phase short circuit MVA= $kVBase^2 / Z_{SC}$

Mvasc1- 1-phase short circuit MVA. There is some ambiguity concerning the meaning of this quantity For the DSS, it is defined as $kVBase^2 / Z_{1\text{-phase}}$ where

$$Z_{1\text{-phase}} = 1/3\ (2Z_1+Z_0)$$

Thus, unless a neutral reactor is used, it should be a number on the same order of magnitude as Mvasc3.

x1r1= Ratio of X1/R1. Default = 4.0.

x0r0= Ratio of X0/R0. Default = 3.0.

Isc3 = Alternate method of defining the source impedance. 3-phase short circuit current, amps. Default is 10000.

Isc1 = Alternate method of defining the source impedance. single-phase short circuit current, amps. Default is 10500.

R1 = Alternate method of defining the source impedance. Positive-sequence resistance, ohms. Default is 1.65.

X1 = Alternate method of defining the source impedance. Positive-sequence reactance, ohms. Default is 6.6.

R0 = Alternate method of defining the source impedance. Zero-sequence resistance, ohms. Default is 1.9.

X0 = Alternate method of defining the source impedance. Zero-sequence reactance, ohms. Default is 5.7.

BaseFreq = Base Frequency for impedance specifications. Default is 60 Hz.

like= Name of an existing Vsource object on which to base this one.

LINE OBJECT

Multi-phase, two-port line or cable. Pi model. Power delivery element described by its impedance. Impedances may be specified by symmetrical component values or by matrix values. Alternatively, you may simply refer to an existing LineCode object from which the impedance values will be copied. Then you need only specify the length.

You can define the line impedance at a base frequency directly in a Line object definition or you can import the impedance definition from a LineCode object. Both of these definitions of impedance are quite similar except that the LineCode object can perform Kron reduction. (See LineCode Object).

If the geometry property is specified all previous definitions are ignored. The DSS will compute the impedance matrices from the specified geometry each time the frequency changes.

Whichever definition is the most recent applies, as with nearly all DSS functions.

Note the units property; you can declare any length measurement in whatever units you please. Internally, everything is converted to meters. Just be sure to declare the units. Otherwise, they are assumed to be compatible with other data or irrelevant.

The properties, in order, are:

bus1= Name of bus for terminal 1. Node order definitions optional.

bus2= Name of bus for terminal 2.

Linecode= name of an existing LineCode object containing impedance definitions.

Length= Length multiplier to be applied to the impedance data.

Phases= No. of phases. Default = 3. A line has the same number of conductors per terminal as it has phases. Neutrals are not explicitly modeled unless declared as a “phase”, and the impedance matrices must be augmented accordingly. For example, a three-phase line has a 3x3 Z matrix with the neutral reduced, or a 4x4 Z matrix with the neutral retained.

R1= positive-sequence resistance, ohms per unit length.

X1= positive-sequence reactance, ohms per unit length.

R0= zero-sequence resistance, ohms per unit length.

X0= zero-sequence reactance, ohms per unit length.

C1= positive-sequence capacitance, nanofarads per unit length.

C0= zero-sequence capacitance, nanofarads per unit length.

Normamps= Normal ampacity, amps.

Emergamps= Emergency ampacity, amps. Usually the one-hour rating.

Faultrate= Number of faults per year per unit length. This is the default for this general line construction.

Pctperm= Percent of the faults that become permanent (requiring a line crew to repair and a sustained interruption).

Repair= Hours to repair.

BaseFreq= Base Frequency at which the impedance values are specified. Default = 60.0 Hz.

Rmatrix= Series resistance matrix, ohms per unit length. See Command Language for syntax. Lower triangle form is acceptable.

Xmatrix= Series reactance matrix, ohms per unit length.

Cmatrix= Shunt nodal capacitance matrix, nanofarads per unit length.

Switch= {y/n | T/F} Default= no/false. Designates this line as a switch for graphics and algorithmic purposes. SIDE EFFECT: Sets R1=0.001 X1=0.0. You must reset if you want something different.

Rg= Carson earth return resistance per unit length used to compute impedance values at base frequency. For making better frequency adjustments. Default=0

Xg= Carson earth return reactance per unit length used to compute impedance values at base frequency. For making better frequency adjustments. Default=0

Rho= Earth resistivity used to compute earth correction factor. Overrides Line geometry definition if specified. Default=100 meter ohms.

Geometry= Geometry code for LineGeometry Object. Supercedes any previous definition of line impedance. Line constants are computed for each frequency change or rho change. CAUTION: may alter number of phases.

Units= Length Units = {none | mi|kft|km|m|Ft|in|cm } Default is None - assumes length units match impedance units.

Like= Name of an existing Line object to build this like.

LOAD OBJECT

A Load is a complicated Power Conversion element that is at the heart of many analyses. It is basically defined by its nominal kW and PF or its kW and kvar. Then it may be modified by a number of multipliers, including the global circuit load multiplier, yearly load shape, daily load shape, and a dutycycle load shape.

The default is for the load to be a current injection source. Thus, its primitive Y matrix contains only the impedance that might exist from the neutral of a wye-connected load to ground. However, if the load model is switched to Admittance from PowerFlow (see Set LoadModel command), the load is converted to an admittance and included in the system Y matrix. This would be the model used for fault studies where convergence might not be achieved because of low voltages.

Loads are assumed balanced for the number of phases specified. If you would like unbalanced loads, enter separate single-phase loads.

There are three legal ways to specify the base load:

1. kW, PF
2. kw, kvar
3. kVA, PF

If you sent these properties in the order shown, the definition should work. If you deviate from these procedures, the result may or may not be what you want. (To determine if it has accomplished the desired effect, execute the Dump command for the desired load(s) and observe the settings.)

The properties, in order, are:

bus1= Name of bus to which the load is connected. Include node definitions if the terminal conductors are connected abnormally. 3-phase Wye-connected loads have 4 conductors; Delta-connected have 3. Wye-connected loads, in general, have one more conductor than phases. 1-phase Delta has 2 conductors; 2-phase has 3. The remaining Delta, or line-line, connections have the same number of conductors as phases.

Phases= No. of phases this load.

Kv= Base voltage for load. For 2- or 3-phase loads, specified in phase-to-phase kV. For all other loads, the actual kV across the load branch. If wye (star) connected, then specify phase-to-neutral (L-N). If delta or phase-to-phase connected, specify the phase-to-phase (L-L) kV.

Kw= nominal kW for load. Total of all phases.

Pf= nominal Power Factor for load. Negative PF is leading. Specify either PF or kvar (see below). If both are specified, the last one specified takes precedence.

Model= Integer defining how the load will vary with voltage. Presently defined models are:

1: Normal load-flow type load: constant P and Q.
2: Constant impedance load
3: Constant P, Quadratic Q (somewhat like a motor)
4: Linear P, Quadratic Q (Mixed resistive, motor)
5: Rectifier load (Constant P, constant current)
6: Constant P; Q is fixed at nominal value
7: Constant P; Q is fixed impedance at nominal value

"Constant" values may be modified by loadshape multipliers. "Fixed" values are always the same -- at nominal, or base, value.

Yearly= Name of Yearly load shape.

Daily= Name of Daily load shape.

Duty= name of Duty cycleload shape. Defaults to Daily load shape if not defined.

Growth= Name of Growth Shape Growth factor defaults to the circuit's default growth rate if not defined. (see Set %Growth command)

Conn= {wye | y | LN} for Wye (Line-Neutral) connection; {delta | LL} for Delta (Line-Line) connection. Default = wye.

Kvar= Base kvar. If this is specified, supercedes PF. (see PF)

Rneut= Neutral resistance, ohms. If entered as negative, non-zero number, neutral is assumed open, or ungrounded. Ignored for delta or line-line connected loads.

Xneut= Neutral reactance, ohms. Ignored for delta or line-line connected loads. Assumed to be in series with Rneut value.

Status= {fixed| variable}. Default is variable. If fixed, then the load is not modified by multipliers; it is fixed at its defined base value.

Class= Integer number segregating the load according to a particular class.

Vminpu = Default = 0.95. Minimum per unit voltage for which the MODEL is assumed to apply. Below this value, the load model reverts to a constant impedance model.

Vmaxpu = Default = 1.05. Maximum per unit voltage for which the MODEL is assumed to apply. Above this value, the load model reverts to a constant impedance model.

VminNorm = Minimum per unit voltage for load EEN evaluations, Normal limit. Default = 0, which defaults to system "vminnorm" property (see Set Command under Executive). If this property is specified, it ALWAYS overrides the system specification. This allows you to have different criteria for different loads. Set to zero to revert to the default system value.

VminEmerg = Minimum per unit voltage for load UE evaluations, Emergency limit. Default = 0, which defaults to system "vminemerg" property (see Set Command under Executive). If this property is specified, it ALWAYS overrides the system specification. This allows you to have different criteria for different loads. Set to zero to revert to the default system value.

XfkVA = Default = 0.0. Rated kVA of service transformer for allocating loads based on connected kVA at a bus. Side effect: kW, PF, and kvar are modified. See PeakCurrent property of EnergyMeter. See also AllocateLoads Command.

AllocationFactor = Default = 0.5. Allocation factor for allocating loads based on connected kVA at a bus. Side effect: kW, PF, and kvar are modified by multiplying this factor times the XFKVA (if > 0). See also AllocateLoads Command.

kVA = Definition of the Base load in kVA, total all phases. This is intended to be used in combination with the power factor (PF) to determine the actual load.

%mean = Percent mean value for load to use for monte carlo studies if no loadshape is assigned to this load. Default is 50.

%stddev = Percent Std deviation value for load to use for monte carlo studies if no loadshape is assigned to this load. Default is 10.

CVRwatts = Percent reduction in active power (watts) per 1% reduction in voltage from 100% rated. Default=1. Typical values range from 0.4 to 0.8. Applies to Model=4 only. Intended to represent conservation voltage reduction or voltage optimization measures.

CVRvars = Percent reduction in reactive power (vars) per 1% reduction in voltage from 100% rated. Default=2. Typical values range from 2 to 3. Applies to Model=4 only. Intended to represent conservation voltage reduction or voltage optimization measures.

spectrum = Name of harmonic current spectrum for this load. Default is "defaultload", which is defined when the DSS starts.

Basefreq = Base frequency for which this load is defined. Default is 60.0.

Like = Name of another Load object on which to base this one.

GENERATOR OBJECT

A Generator is a Power Conversion element similar to a Load object. Its rating is basically defined by its nominal kW and PF or its kW and kvar. Then it may be modified by a number of multipliers, including the global circuit load multiplier, yearly load shape, daily load shape, and a dutycycle load shape.

The generator is essentially a negative load that can be dispatched.

If the dispatch value (DispValue property) is 0, the generator always follows the appropriate dispatch curve, which is simply a Loadshape object. If DispValue>0 then the generator only comes on when the global circuit load multiplier exceeds DispValue. When the generator is on, it always follows the dispatch curve appropriate for the type of solution being performed.

If you want to model a generator that is fully on whenever it is dispatched on, simply designate "Status=Fixed". The default is "Status=Variable" (i.e., it follows a dispatch curve. You could also define a dispatch curve that is always 1.0.

Generators have their own energy meters that record:

1. Total kwh
2. Total kvarh
3. Max kW
4. Max kVA
5. Hours in operation
6. $ (Price signal * energy generated)

Generator meters reset with the circuit energy meters and take a sample with the circuit energy meters as well. The Energy meters also used trapezoidal integration so that they are compatible with Load-Duration simulations.

Generator power models are:

1. Constant P, Q (* dispatch curve, if appropriate).
2. Constant Z (For simple, approximate solution)
3. Constant P, |V| somwhat like a standard power flow
4. Constant P, fixed Q. P follows dispatch; Q is always the same.
5. Constant P, fixed reactance. P follows dispatch, Q is computed as if it were a fixed reactance.

Most of the time you will use #1 for planning studies.

The default is for the generator to be a current injection source. Thus, its primitive Y matrix is similar to a Load object with a nominal equivalent admittance included in the primitive Y matrix and the injection current representing the amount required to compensate. However, if the generator model is switched to Admittance from PowerFlow (see Set Mode command), the generator is converted to strictly an eqiuivalent admittance and included in the system Y matrix.

Generators are assumed balanced for the number of phases specified. If you would like unbalanced generators, enter separate single-phase generators.

The properties, in order, are:

bus1= Name of bus to which the generator is connected. Include node definitions if the terminal conductors are connected unusually. 3-phase Wye-connected generators have 4 conductors; Delta-connected have 3. Wye-connected generators, in general, have one more conductor than phases. 1-phase Delta has 2 conductors; 2-phase has 3. The remaining Delta, or line-line, connections have the same number of conductors as phases.

Phases= No. of phases this generator.

Kv= Base voltage for generator. For 2- or 3-phase generators, specified in phase-to-phase kV. For all other generators, the actual kV across the generator branch. If wye (star) connected, specify the phase-to-neutral (L-N) kV. If delta or phase-to-phase connected, specify the phase-to-phase (L-L) kV.

Kw= nominal kW for generator. Total of all phases.

Pf= nominal Power Factor for generator. Negative PF is leading (absorbing vars). Specify either PF or kvar (see below). If both are specified, the last one specified takes precedence.

Model= Integer defining how the generator will vary with voltage. Presently defined models are:

1: Generator injects a constant kW at specified power factor.
2: Generator is modeled as a constant admittance.
3: Const kW, constant kV. Somewhat like a conventional transmission power flow P-V generator.
4: Const kW, Fixed Q (Q never varies)
5: Const kW, Fixed Q(as a constant reactance)
6: Compute load injection from User-written Model.(see usage of Xd, Xdp)
7: Constant kW, kvar, but current limited below Vminpu

Yearly= Name of Yearly load shape.

Daily= Name of Daily load shape.

Duty= name of Duty cycle load shape. Defaults to Daily load shape if not defined.

Dispvalue= Dispatch value. If = 0.0 then Generator follows dispatch curves. If > 0 then Generator is ON only when the global load multiplier exceeds this value. Then the generatorfollows dispatch curves (see also Status)

Conn= {wye | y | LN} for Wye (Line-Neutral) connection; {delta | LL} for Delta (Line-Line) connection. Default = wye.

Kvar= Base kvar. If this is specified, will supercede PF. (see PF)

Rneut= Neutral resistance ohms. If entered as negative, non-zero number, neutral is assumed open, or ungrounded. Ignored for delta or line-line connected generators. Default is 0.

Xneut= Neutral reactance, ohms. Ignored for delta or line-line connected generators. Assumed to be in series with Rneut value.

Status= {fixed | variable}. If Fixed, then dispatch multipliers do not apply. The generator is always at full power when it is ON. Default is Variable (follows curves).

Class= Integer number segregating the generator according to a particular class.

Maxkvar = Maximum kvar limit for Model = 3. Defaults to twice the specified load kvar. Always reset this if you change PF or kvar properties.

Minkvar = Minimum kvar limit for Model = 3. Enter a negative number if generator can absorb vars. Defaults to negative of Maxkvar. Always reset this if you change PF or kvar properties.

Pvfactor = Convergence deceleration factor for P-V generator model (Model=3). Default is 0.1. If the circuit converges easily, you may want to use a higher number such as 1.0. Use a lower number if solution diverges. Use Debugtrace=yes to create a file that will trace the convergence of a generator model.

Debugtrace = {Yes | No } Default is no. Turn this on to capture the progress of the generator model for each iteration. Creates a separate file for each generator named "GEN_name.CSV".

Vminpu = Default = 0.95. Minimum per unit voltage for which the Model is assumed to apply. Below this value, the generator model reverts to a constant impedance model.

Vmaxpu = Default = 1.05. Maximum per unit voltage for which the Model is assumed to apply. Above this value, the generator model reverts to a constant impedance model.

ForceON = {Yes | No} Forces generator ON despite requirements of other dispatch modes. Stays ON until this property is set to NO, or an internal algorithm cancels the forced ON state.

kVA = kVA rating of electrical machine. Defaults to 1.2* kW if not specified. Applied to machine or inverter definition for Dynamics mode solutions.

MVA = MVA rating of electrical machine. Alternative to using kVA=.

Xd = Per unit synchronous reactance of machine. Presently used only for Thevinen impedance for power flow calcs of user models (model=6). Typically use a value from 0.4 to 1.0. Default is 1.0

Xdp = Per unit transient reactance of the machine. Used for Dynamics mode and Fault studies. Default is 0.27.For user models, this value is used for the Thevinen/Norton impedance for Dynamics Mode.

Xdpp = Per unit subtransient reactance of the machine. Used for Harmonics. Default is 0.20.

H = Per unit mass constant of the machine. MW-sec/MVA. Default is 1.0.

D = Damping constant. Usual range is 0 to 4. Default is 1.0. Adjust to get damping

UserModel = Name of DLL containing user-written model, which computes the terminal currents for Dynamics studies, overriding the default model. Set to "none" to negate previous setting.

UserData = String (in quotes or parentheses) that gets passed to user-written model for defining the data required for that model.

ShaftModel = Name of user-written DLL containing a Shaft model, which models the prime mover and determines the power on the shaft for Dynamics studies. Models additional mass elements other than the single-mass model in the DSS default model. Set to "none" to negate previous setting.

ShaftData = String (in quotes or parentheses) that gets passed to user-written shaft dynamic model for defining the data for that model.

spectrum = Name of harmonic voltage or current spectrum for this generator. Voltage behind Xd" for machine - default. Current injection for inverter. Default value is "default", which is defined when the DSS starts.

Basefreq= Base frequency for which this generator is defined. Default is 60.0.

Like= Name of another Generator object on which to base this one.

ENERGYMETER OBJECT

An EnergyMeter object is an intelligent meter connected to a terminal of a circuit element. It simulates the behavior of an actual energy meter. However, it has more capability because it can access values at other places in the circuit rather than simply at the location at which it is installed. It measures not only power and energy values at its location, but losses and overload values within a defined region of the circuit.

The operation of the object is simple. It has several *registers* that accumulate certain values. At the beginning of a study, the registers are cleared (reset) to zero. At the end of each subsequent solution, the meter is instructed to take a sample. Energy values are then integrated using the interval of time that has passed since the previous solution.

Registers

There are two types of registers:

1. Energy Accumulators (for energy values)

2. Maximum power values ("drag hand" registers).

The energy registers use trapezoidal integration, which allows for somewhat arbitrary time step sizes between solutions with less integration error. This is important for using load duration curves approximated with straight lines, for example.

The present definitions of the registers are:

1. **KWh at the meter location.**
2. **Kvarh at the meter location.**
3. **Maximum kW at the meter location.**
4. **Maximum kVA at the meter location.**
5. **KWh in the meter zone.**
6. **Kvarh in the meter zone.**
7. **Maximum kW in the meter zone.**
8. **Maximum kVA in the meter zone.**
9. **Overload kWh in the meter zone, normal ratings.**
10. **Overload kWh in the meter zone, emergency ratings.**
11. **Energy Exceeding Normal (EEN) in the loads in the meter zone.**
12. **Unserved Energy (UE) in the loads in the meter zone.**
13. **Losses (kWh) in power delivery elements in the meter zone.**
14. **Reactive losses (kvarh) in power delivery elements in the meter zone.**
15. **Maximum losses (kW) in power delivery elements in the meter zone.**
16. **Maximum reactive losses (kvar) in power delivery elements in the meter zone.**
17. **Load Losses kWh. I^2R Losses in power delivery elements**
18. **Load Losses kvarh. I^2X Losses in power delivery elements**
19. **No Load Losses kWh in shunt elements, principally transformers.**
20. **No Load Losses kvarh in shunt elements.**
21. **Max kW Load Losses during the simulation**
22. **Max kW No Load Losses during the simulation**
23. **Line Losses: Losses in LINE elements.**
24. **Transformer Losses: Losses in TRANSFORMER elements.**
25. **Line Mode Line Losses (3X Pos and neg seq losses)**
26. **Zero Mode Line Losses (3X zero sequence losses)**
27. **Gen kWh**

28. **Gen kvarh**
29. **Gen Max kW**
30. **Gen Max kVA**
31. **Aux1 (used for segregating losses by voltage level)**
32. **Aux2**
33. **Aux3**
34. **Aux4**
35. **Aux5**
36. **Aux6**
37. **Aux7**

Registers are frequently added for various purposes. You can view the present meters simply by solving and taking a sample. Then do Show Meters.

Zones

The EnergyMeter object uses the concept of a *zone.* This is an area of the circuit for which the meter is responsible. It can compute energies, losses, etc for any power delivery object and Load object in its zone (Generator objects have their own intrinsic meters).

A zone is a collection of circuit elements "downline" from the meter. This concept is nominally applicable to radial circuits, but also has some applicability to meshed circuits. The zones are automatically determined according to the following rules:

1. Start with the circuit element in which the meter is located. Ignore the terminal on which the meter is connected. This terminal is the start of the zone. Begin tracing with the other terminal(s).

2. Trace out the circuit, finding all other circuit elements (loads and power delivery elements) connected to the zone. Continue tracing out every branch of the circuit. Stop tracing a branch when:

    - The end of the circuit branch is reached
    - A circuit element containing another EnergyMeter object is encountered
    - A OPEN terminal is encountered. (all phases in the terminal are open.)
    - A disabled device is encountered.
    - A circuit element already included in another zone is encountered.
    - There are no more circuit elements to consider.

Zones are automatically updated after a change in the circuit unless the ZONELOCK option (Set command) is set to true (Yes). Then zones remain fixed after initial determination.

.Zones on Meshed Networks

While the concept of zones nominally applies to radial circuits, judicious placement of energy meters can make the concept useful for meshed networks as well. Keep in mind that there can be many EnergyMeter objects defined in the circuit. Their placement does not necessarily have to represent reality; they are for the reporting of power and energy quantities throughout the system.

The automatic algorithm for determining zones will determine zones consistently for meshed networks, although the zones themselves may not be radial. If there are several meters on the network that could be monitoring the same zone, the first one defined will have access to all the elements except the ones containing the other meters. The others will have only one element in their zone, as in Figure 14.

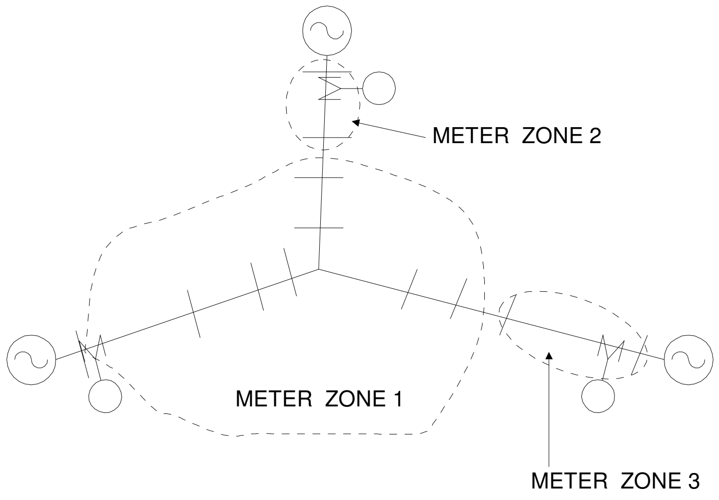


Figure 14. Default Meter Zones for a Simple Network

Don't put any load at tie point or take care in processing meter information so that it isn't counted more than once. All three of the meters added would see the center-most bus in Figure 15.

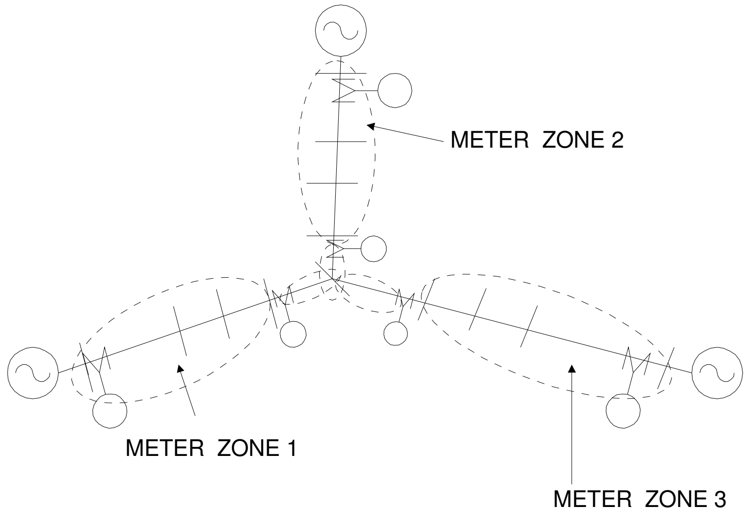


Figure 15. Using Additional Meters to Control the Definition of Meter Zones

Sampling

The sampling algorithms are as follows:

1. <u>Local Energy and Power Values</u>: Simply compute the power into the terminal on which the meter is installed and integrate using the interval between the present solution and the previous solution. This operation uses the voltage and current computed from the present solution.

2. <u>Losses in Zone</u>: Accumulate the kW losses in each power delivery element in the zone.

3. <u>Load in Zone</u>: While sampling the losses in each power delivery element, accumulate the power in all loads connected to the downline bus(es) of the element.

4. <u>Overload Energy in Zone</u>: For each power delivery element in the zone, compute the amount of power exceeding the rating of the element compared to both normal and emergency ratings.

5. <u>EEN and UE in Zone</u>: For each load in the zone marked as exceeding normal or unserved, compute the present power. Integrate to get energies.

EEN and UE Definitions

EEN refers to load energy considered unserved because the current or voltage exceeds Normal ratings. UE refers to load energy considered unserved because the power (actually the current) exceeds Emergency, or maximum, ratings.

On radial systems (default), a load is marked as unserved with respect to either normal or emergency ratings if either:

1. The voltage at the load bus is below minimum ratings

2. The current in any power delivery element supplying the load exceeds the current ratings.

Either the entire load or just the portion above rating at the bus that is considered unserved is counted as unserved, depending on whether the Excess option or the Total option have been specified.

EnergyMeter EEN and UE Registers

Register 9 through 12 on the EnergyMeter are confusing to both new and experienced OpenDSS users. Hopefully, this explanation will help shed some light on the contents of these registers.

EEN = Energy Exceeding Normal: The energy served over a selected period of time above the "Normal" rating of power delivery devices (lines, transformers, switches, etc.). It is assumed that there is sufficient engineering margin in the power system to continue to operate, but that a failure (1st contingency) will require curtailment of load. This is the primary quantity used for measure of risk.

UE = Unserved Energy: The energy over a selected period of time projected to be above the Emergency, or Maximum, rating of power delivery equipment. It is assumed that some load will have to be curtailed to bring the power down to a manageable level.

These concepts have evolved since 1994 as we began to look for some means to measure risk in planning, particularly as it applies to distributed generation. We quickly learned that distribution planners were not comfortable with pure probabilistic planning. They prefer methods based on concrete limits. Therefore, we adapted some traditional planning concepts to achieve a measure of risk. Many traditional methods used two limits for power delivery equipment: Normal and Emergency. Planning studies are triggered when the peak demand exceeds the Normal limits. Exceeding the Emergency limit requires immediate curtailment. The usual intent was to get something built before the Emergency limit was projected to be exceeded. By using power demand alone, new construction is frequently very conservative. Distributed generation muddies the waters for planning with demand alone because it is often unclear how much capacity is actually achieved by DG.

Our extension of the concept was to use *energy* in addition to simply *power* to determine the risk. One would defer investing in new infrastructure until the energy exceeding the limits was sufficient to justify it.

For more on this subject, see the following papers for reference on this subject:

R. C. Dugan, "Computing Incremental Capacity Provided By Distributed Resources For Distribution Planning," IEEE PES General Meeting, 2007.

Roger Dugan and Marek Waclawiak, “Using Energy as a Measure of Risk in Distribution Planning,” Paper 0822, CIRED 2007, Vienna.

Registers 9 and 10: Overload EEN and UE

As the Energymeter element sweeps through its zone, it queries each series power delivery element encountered for the kW above Normal and Emergency limits. The value recorded in these two registers is the largest kW amount encountered. In other words, the value is the maximum overload in terms of actual kW.

Most of the time, this is what you want. However, there are cases where data errors can skew the results. For example, if the data show a 15 kVA transformer serving an apartment building with over 150 kVA load, this will consistently show up as a 135 kVA overload, which might be the largest overload in the problem. However, you would not build a new feeder because a small transformer is overloaded. You would change out the transformer (or correct the data error). Such errors are common in distribution data where it may take a while for transformer changeouts to get properly entered.

Excess or Total: There are two ways these registers can record the results. They can simply record the excess kW over the limits. This is the default behavior. However, they can also record the total kW flowing through the element. The latter method is for cases where it is assumed that the feeder branch in question must be switched off completely if an overload occurs. This reflects a more conservative planning approach.

Registers 11 and 12: Load EEN and UE

These two registers record a different approach to compute EEN and UE values: They ask each Load element in the zone if it is “unserved.” The nominal criterion is undervoltage.

If you set Option=Voltage for an Energymeter object, only the voltage is used. The global options NormVminpu and EmergVminpu are used for this value. If the voltage is between NormVminpu and EmergVminpu, the EEN is proportioned to how far below the NormVminpu it is. If the voltage is less than EmergVminpu, the UE value for the load is computed in proportion to the degee it is below EmergVminpu, continuing on the same slope as the EEN calculation. In multi-phase elements, the lowest phase voltage is used.

The default behavior (Option=Combined) is to consider both line overload and undervoltage. If a line, or other power delivery element, serving the load is overloaded, the load is considered unserved in the same percentage that the line is overloaded. This actually takes precedence over the voltage criteria, which assumes that any undervoltage is due to the line overload. A line is considered to serve the load if it is between the EnergyMeter and the load. Note that some inaccuracies can occur if the meter zone is not properly defined, such as if loops exist.

Properties

The properties, in order, are:

Element= Name of an existing circuit element to which the monitor is to be connected.

Note that there may be more than one circuit element with the same name (not wise, but it is allowed). The monitor will be placed at the first one found in the list.

Terminal= No. of the terminal to which the monitor will be connected.

Action= Optional action to execute. One of

1. Clear = reset all registers to zero
2. Save = Saves (appends) the present register values to a file. File name is MTR_*metername*.CSV, where *metername* is the name of the energy meter.
3. Take = Takes a sample at the present solution.

Option = Options: Enter a string ARRAY of any combination of the following. Options processed left-to-right:

(**E**)xcess : (default) UE/EEN is estimate of only energy exceeding capacity

(**T**)otal : UE/EEN is total energy after capacity exceeded.

(**R**)adial : (default) Treats zone as a radial circuit

(**M**)esh : Treats zone as meshed network (not radial).

Example: option=(E, R)

In a meshed network, the overload registers represent the total of the power delivery element overloads and the load UE/EEN registers will contain only those loads that are "unserved", which are those with low voltages. In a radial circuit, the overload registers record the max overload (absolute magnitude, not percent) in the zone. Loads become unserved either with low voltage or if any line in their path to the source is overloaded.

KWNorm = Upper limit on kW load in the zone, *Normal* configuration. Default is 0.0 (ignored). If specified, overrides limits on individual lines for overload EEN. KW above this limit for the entire zone is considered EEN.

KWEmerg = Upper limit on kW load in the zone, *Emergency* configuration. Default is 0.0 (ignored). If specified, overrides limits on individual lines for overload UE. KW above this limit for the entire zone is considered UE.

Peakcurrent = ARRAY of current magnitudes representing the peak currents measured at this location for the load allocation function (for loads defined with **xfkva=**). Default is (400, 400, 400). Enter one current for each phase.

Zonelist = ARRAY of full element names for this meter's zone. Default is for meter to find it's own zone. If specified, DSS uses this list instead. It can access the names in a single-column text file. Examples:

```
zonelist=[line.L1, transformer.T1, Line.L3]
```

zonelist=(file=branchlist.txt)

LocalOnly = {Yes | No} Default is NO. If Yes, meter considers only the monitored element for EEN and UE calcs. Uses whole zone for losses.

Mask = Mask for adding registers whenever all meters are totalized. Array of floating point numbers representing the multiplier to be used for summing each register from this meter. Default = (1, 1, 1, 1, ...). You only have to enter as many as are changed (positional). Useful when two meters monitor same energy, etc.

Losses = {Yes | No} Default is YES. Compute Zone losses. If NO, then no losses at all are computed.

LineLosses = {Yes | No} Default is YES. Compute Line losses. If NO, then none of the line losses are computed.

XfmrLosses = {Yes | No} Default is YES. Compute Transformer losses. If NO, transformers are ignored in loss calculations.

SeqLosses = {Yes | No} Default is YES. Compute Sequence losses in lines and segregate by line mode losses and zero mode losses.

VbaseLosses = {Yes | No} Default is YES. Compute losses and segregate by voltage base. If NO, then voltage-based tabulation is not reported. Make sure the voltage bases on the buses are assigned BEFORE defining the EnergyMeter to ensure that it will automatically pick up the voltage bases.

Like= Name of another EnergyMeter object on which to base this one.

MONITOR OBJECT

A monitor is a benign circuit element that is associated with a terminal of another circuit element. It takes a sample when instructed, recording the time and the complex values of voltage and current, or power, at all phases. The data are saved in a file stream (separate one for each monitor) at the conclusion of a multistep solution (e.g., daily or yearly, or harmonics) or each solution in a Monte Carlo calculation. In essence, it works like a real power monitor. The data in the file may be converted to csv form and, for example, brought into Excel. You may accomplish this by either the Show Monitor command or the Export Monitor command.

For Monte Carlo runs, the hour is set to the number of the solution and seconds is set to zero. For Harmonic solutions, the first two fields are changed to frequency and harmonic.

Monitors may be connected to both power delivery elements and power conversion elements.

Parameters, in order, are:

Element= Name of an existing circuit element to which the monitor is to be connected. Note that there may be more than one circuit element with the same name (not wise, but it is allowed). The monitor will be placed at the first one found in the list.

Terminal= No. of the terminal to which the monitor will be connected.

Mode= Integer bitmask code to describe what it is that the monitor will save. Monitors can save two basic types of quantities: 1) Voltage and current; 2) Power. The Mode codes are defined as follows:

1. 0: Standard mode - V and I,each phase, complex
2. 1: Power each phase, complex (kw and kvars)
3. Transformer taps
4. State variables

+16: Sequence components: V_{012}, I_{012}
+32: Magnitude only
+64: Pos Seq only or Average of phases, if not 3 phases

For example, Mode=33 will save the magnitude of the power (kVA) only in each phase. Mode=112 saves Positive sequence voltages and currents, magnitudes only.

Action= {clear | save} parsing of this parameter forces clearing of the monitor's buffer, or saving to disk.

Check the Result window for the name of the file created.

Capacitor Object

The capacitor model is basically implemented as a two-terminal power delivery element. However, if you don't specify a connection for the second bus, it will default to the 0 node (ground reference) of the same bus to which the first terminal is connected. That is, it defaults to a grounded wye (star) shunt capacitor bank.

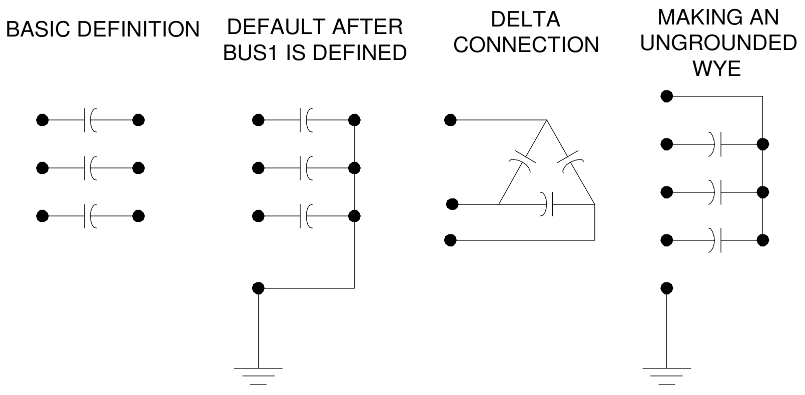


Figure 16: Definition of Capacitor Object

If you specify the connection to be "delta" then the second terminal is eliminated.

If you wish a series capacitor, simply specify a second bus connection.

If you wish an ungrounded wye capacitor, set all the second terminal conductors to an empty node on the first terminal bus, e.g.:

Bus1=B1 bus2 = B1.4.4.4 ! for a 3-phase capacitor

Of course, any other connection is possibly by explicitly specifying the nodes.

While the Capacitor object may be treated as if it were a single capacitor bank, it is actually implemented as a multistep tuned filter bank. Many of the properties can be specified as arrays. See Numsteps property.

If you wish to represent a filter bank, specify either the XL or Harm property. When a Capcontrol object switches a capacitor, it does so by incrementing or decrementing the active step.

Parameters, in order, are:

Bus1= Definition for the connection of the first bus. When this is set, Bus2 is set to the same bus name, except with all terminals connected to node 0 (ground reference). Set Bus 2 at some time later if you wish a different connection.

Bus2= Bus connection for second terminal. Must always be specified after Bus1 for series capacitor. Not necessary to specify for delta or grd-wye shunt capacitor. Must be specified to achieve an ungrounded neutral point connection.

Phases= Number of phases. Default is 3.

Kvar= Most common of three ways to define a power capacitor. Rated kvar at rated kV, total of all phases. Each phase is assumed equal. Normamps and Emergamps automatically computed. Default is 600.0 kvar. Total kvar, if one step, or **ARRAY** of kvar ratings for each step. Evenly divided among phases. See rules for NUMSTEPS.

Kv= Rated kV of the capacitor (not necessarily same as bus rating). For Phases=2 or Phases=3, enter line-to-line (phase-to-phase) rated voltage. For all other numbers of phases, enter actual can rating. (For Delta connection this is always line-to-line rated voltage). Default is 12.47 kV.

Conn= Connection of bank. One of {wye | ln} for wye connected banks or {delta | ll} for delta (line-line) connected banks. Default is wye (or straight-through for series capacitor).

Cmatrix= Alternate method of defining a capacitor bank. Enter nodal capacitance matrix in µf. Can be used to define either series or shunt banks. Form should be:

Cmatrix=[c_{11} | -c_{21} c_{22} | -c_{31} -c_{32} c_{33}]

All steps are assumed the same if this property is used.

Cuf= Alternate method of defining a capacitor bank. Enter a value or ARRAY of values for C in µf. ARRAY of Capacitance, each phase, for each step, microfarads. See Rules for NumSteps.

R = ARRAY of series resistance in each phase (line), ohms. Default is 0.0

XL = ARRAY of series inductive reactance(s) in each phase (line) for filter, ohms at base frequency. Use this OR "h" property to define filter. Default is 0.0.

Harm = ARRAY of harmonics to which each step is tuned. Zero is interpreted as meaning zero reactance (no filter). Default is zero.

Numsteps = Number of steps in this capacitor bank. Default = 1. Forces reallocation of the capacitance, reactor, and states array. Rules: If this property was previously =1, the value in the kvar property is divided equally among the steps. The kvar property does not need to be reset if that is correct. If the Cuf or Cmatrix property was used previously, all steps are set to the value of the first step. The states property is set to all steps on. All filter steps are initially set to the same harmonic. If this property was previously >1, the arrays are reallocated, but no values are altered. You must SUBSEQUENTLY assign all array properties.

states = ARRAY of integers {1|0} states representing the state of each step (on|off). Defaults to 1 when reallocated (on). Capcontrol will modify this array as it turns steps on or off.

Normamps= Normal current rating. Automatically computed if kvar is specified. Otherwise, you need to specify if you wish to use it.

Emergamps= Overload rating. Defaults to 135% of Normamps.

Faultrate= Annual failure rate. Failure events per year. Default is 0.0005.

Pctperm= Percent of faults that are permanent. Default is 100.0.

Basefreq= Base frequency, Hz. Default is 60.0

Like= Name of another Capacitor object on which to base this one.

TRANSFORMER OBJECT

The Transfomer is implemented as a multi-terminal (two or more) power delivery element.

A transfomer consists of two or more *Windings*, connected in somewhat arbitrary fashion (with a default Wye-Delta connection). You can specify the parameters one winding at a time or use arrays to set all the winding values at once. Use the "wdg=…" parameter to select a winding for editing.

Transformers have one or more *phases.* The number of conductors per terminal is always one more than the number of phases. For wye- or star-connected windings, the extra conductor is the neutral point. For delta-connected windings, the extra terminal is open internally (you normally leave this connected to node 0).

Parameters, in order, are:

Phases= Number of phases. Default is 3.

Windings= Number of windings. Default is 2.

For defining the winding values one winding at a time, use the following parameters. Always start the winding definition with "wdg = …" when using this method of defining transformer parameters. The remainder of the tags are optional as usual if you keep them in order.

Wdg= Integer representing the winding which will become the active winding for subsequent data.

Bus= Definition for the connection of this winding (each winding is connected to one terminal of the transformer and, hence, to one bus).

Conn= Connection of this winding. One of {wye | ln} for wye connected banks or {delta | ll} for delta (line-line) connected banks. Default is wye.

Kv= Rated voltage of this winding, kV. For transformers designated 2- or 3-phase, enter phase-to-phase kV. For all other designations, enter actual winding kV rating. Two-phase transfomers are assumed to be employed in a 3-phase system. Default is 12.47 kV.

Kva= Base kVA rating (OA rating) of this winding.

Tap = Per unit tap on which this winding is set.

%r = Percent resistance of this winding on the rated kVA base. (Reactance is *between* two windings and is specified separately -- see below.)

rneut = Neutral resistance to ground in ohms for this winding. Ignored if delta winding.

For open ungrounded neutral, set to a **negative** number. Default is –1 (capable of being ungrounded). The DSS defaults to connecting the neutral to node 0 at a bus, so it will still be ground when the system Y is built. To make the neutral floating, explicitly connect it to an unused node at the bus, e.g., Bus=Busname.1.2.3.4, when node 4 will be the explicit neutral node.

xneut = Neutral reactance in ohms for this winding. Ignored if delta winding. Assumed to be in series with neutral resistance. Default is 0.

Use the following parameters to set the winding values using arrays (setting of wdg= … is ignored).

Buses = Arrayof bus definitions for windings [1, 2. …].

Conns = Array of winding connections for windings [1, 2. …].

KVs = Array of kV ratings following rules stated above for the kV field for windings [1,2,…].

KVAs = Array of base kVA ratings for windings [1,2,…].

Taps = Array of per unit taps for windings [1,2,…].

Use the following parameters to define the reactances of the transformer. For 2- and 3-winding transformers, you may use the conventional XHL, XLT, and XHT parameters. You may also put the values in an array (xscarray), which is required for higher phase order transformers. There are always n*(n-1)/2 different short circuit reactances, where n is the number of windings. *Always use the kVA base of the first winding for entering impedances.* Impedance values are entered in percent.

XHL = Percent reactance high-to-low (winding 1 to winding 2).

XLT = Percent reactance low-to-tertiary (winding 2 to winding 3).

XHT = Percent reactance high-to-tertiary (winding 1 to winding 3).

XscArray = Array of n*(n-1)/2 short circuit reactances in percent on the first winding's kVA base. "n" is number of windings. Order is (12, 13, 14, …1n, 23, 24, … 34, …)

General transformer rating data:

Thermal = Thermal time constant, hrs. Default is 2.

n = Thermal exponent, n, from IEEE/ANSI C57. Default is 0.8.

m = Thermal exponent, m, from IEEE/ANSI C57. Default is 0.8.

flrise = Full-load temperature rise, degrees centigrade. Default is 65.

hsrise = Hot-spot temperatire rise, degrees centigrade. Default is 15.

%Loadloss = Percent Losses at rated load.. Causes the **%r** values to be set for windings 1 and 2.

%Noloadloss = Percent No load losses at nominal voltage. Default is 0. Causes a resistive branch to be added in parallel with the magnetizing inductance.

%imag = Percent magnetizing current. Default is 0. An inductance is used to represent the magnetizing current. This is embedded within the transformer model as the primitive Y matrix is being computed.

Ppm_Antifloat = Parts per million for anti floating reactance to be connected from each terminal to ground. Default is 1. That is, the diagonal of the primitive Y matrix is increased by a factor of 1.000001. Prevents singular matrix if delta winding left floating. Set to zero if you don't need it and the resulting impedance to ground is affecting the results. Is inconsequential for most cases.

NormHKVA = Normal maximum kVA rating for H winding (1). Usually 100 - 110% of maximum nameplate rating.

EmergHKVA = Emergency maximum kVA rating for H winding (1). Usually 140 - 150% of maximum nameplate rating. This is the amount of loading that will cause 1% loss of life in one day.

Faultrate = Failure rate for transformer. Defaults to 0.007 per year. All are considered permanent.

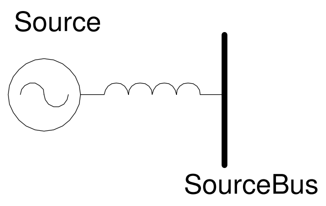
Basefreq= Base frequency, Hz. Default is 60.0

Like= Name of another Transformer object on which to base this one.

Sub = Yes/No. Designates whether this transformer is to be treated as a substation. Default is No.

Default Circuit

When a new circuit is instantiated, it is created as a 3-phase voltage source named "Source" connected to a bus named "SourceBus" with a reasonable short circuit strength for transmission systems feeding distribution substations.



The default values are (see Vsource object definition):

115 kV

3000 MVA short circuit

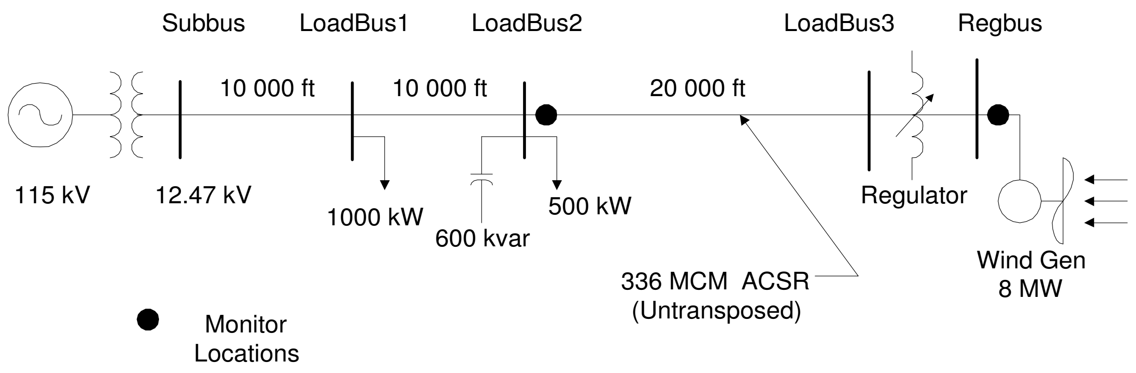
Thus, the circuit can be immediately solved, albeit with a trivial result.

The circuit is also immediately available for adding a substation and/or lines for a quick manual circuit modeling task.

The default circuit model does not include a substation transformer because the user may wish to study more than one substation connected by a non-trivial transmission or subtransmission network.

Examples

EXAMPLE CIRCUIT 1

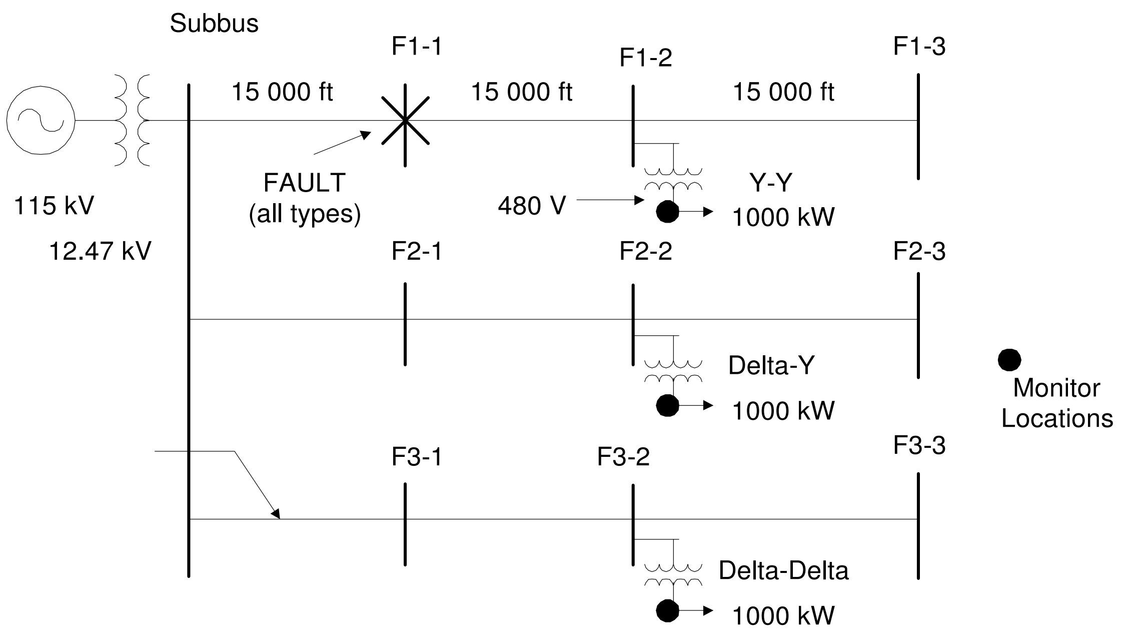


DSS Circuit Description Script

```
new object=circuit.DSSLLibtestckt
~ basekv=115  1.00 0.0 60.0 3 20000 21000 4.0 3.0  !edit the voltage source

new loadshape.day 24 1.0
~   mult=(.3 .3 .3 .35 .36 .39 .41 .48 .52 .59 .62 .94 .87 .91 .95 .95 1.0 .98 .94
.92 .61 .60 .51 .44)
new loadshape.year 24 1.0  ! same as day for now
~   mult=".3 .3 .3 .35 .36 .39 .41 .48 .52 .59 .62 .94 .87 .91 .95 .95 1.0 .98 .94
.92 .61 .60 .51 .44"
new loadshape.wind 2400 0.00027777   ! unit must be hours 1.0/3600.0 = .0002777
~   csvfile=zavwind.csv action=normalize  ! wind turbine characteristi

! define a linecode for the lines - unbalanced 336 MCM ACSR connection
new linecode.336matrix nphases=3   ! horizontal flat construction
~  rmatrix=(0.0868455 |  0.0298305 0.0887966 | 0.0288883 0.0298305  0.0868455) !
ohms per 1000 ft
~  xmatrix=(0.2025449 |  0.0847210 0.1961452 | 0.0719161 0.0847210  0.2025449)
~  cmatrix=(2.74 | -0.70 2.96| -0.34 -0.71 2.74)  !nf per 1000 ft
~ Normamps = 400  Emergamps=600

! Substation transformer
new transformer.sub phases=3 windings=2 buses=(SourceBus subbus) conns='delta wye'
kvs="115 12.47 " kvas="20000 20000" XHL=7

! define the lines
new line.line1 subbus   loadbus1 linecode=336matrix length=10
new line.line2 loadbus1 loadbus2 336matrix 10
new line.line3 Loadbus2 loadbus3 336matrix 20

! define a couple of loads
new load.load1 bus1=loadbus1 phases=3 kv=12.47 kw=1000.0 pf=0.88 model=1 class=1
yearly=year daily=day status=fixed
new load.load2 bus1=loadbus2 phases=3 kv=12.47 kw=500.0 pf=0.88 model=1 class=1
yearly=year daily=day conn=delta status=fixed

! Capacitor with control
new capacitor.C1  bus1=loadbus2  phases=3 kvar=600 kv=12.47
new capcontrol.C1 element=line.line3 1 capacitor=C1 type=current ctratio=1
ONsetting=60 OFFsetting=55 delay=2
```

```
! regulated transformer to DG bus
new transformer.reg1 phases=3 windings=2
~           buses=(loadbus3 regbus)
~           conns='wye wye'
~           kvs="12.47 12.47"
~           kvas="8000 8000"
~           XHL=1              !tiny reactance for a regulator

! Regulator Control definitions
new regcontrol.sub  transformer=sub  winding=2 vreg=125 band=3 ptratio=60 delay=10
new regcontrol.reg1 transformer=reg1 winding=2 vreg=122 band=3 ptratio=60 delay=15

! define a wind generator of 8MW
New generator.gen1   bus1=regbus kV=12.47 kW=8000 pf=1 conn=delta duty=wind
Model=1


! Define some monitors so's we can see what's happenin'

New Monitor.gen1a element=generator.gen1 1   mode=48
New Monitor.line3 element=line.line3 1     mode=48
New Monitor.gen1  element=generator.gen1 1 mode=32

! Define voltage bases so voltage reports come out in per unit
Set voltagebases="115 12.47 .48"
Calcv

Set controlmode=time
Set mode=duty number=2400  hour=0  h=1.0 sec=0  ! Mode resets the monitors
```

EXAMPLE CIRCUIT 2



DSS Circuit Description Script

```
new object=circuit.DSSLLibtestckt
~ basekv=115  1.00 0.0 60.0 3 20000 21000 4.0 3.0  !edit the voltage source

! define a linecode for the lines - unbalanced 336 MCM ACSR connection
new linecode.336matrix nphases=3   ! horizontal flat construction
~  rmatrix=(0.0868455 |  0.0298305 0.0887966 | 0.0288883 0.0298305  0.0868455) !
ohms per 1000 ft
~  xmatrix=(0.2025449 |  0.0847210 0.1961452 | 0.0719161 0.0847210  0.2025449)
~  cmatrix=(2.74 | -0.70 2.96| -0.34 -0.71 2.74)  !nf per 1000 ft
~  Normamps = 400  Emergamps=600

! Substation transformer
new transformer.sub phases=3 windings=2 buses=(SourceBus subbus) conns='delta wye'
kvs="115 12.47 " kvas="20000 20000" XHL=7

! define the lines  (Make sure they have unique names!)

! Feeder 1
new line.line1-1 subbus   F1-1 336matrix 15
new line.line1-2 F1-1     F1-2 336matrix 15
new line.line1-3 F1-2     F1-3 336matrix 15

! Feeder 2
new line.line2-1 subbus   F2-1 336matrix 15
new line.line2-2 F2-1     F2-2 336matrix 15
new line.line2-3 F2-2     F2-3 336matrix 15

! Feeder 3
new line.line3-1 subbus   F3-1 336matrix 15
new line.line3-2 F3-1     F3-2 336matrix 15
new line.line3-3 F3-2     F3-3 336matrix 15
```

```
! Define 3 transformer of different connections on each feeder midpoint
new transformer.TR1 phases=3 windings=2
~           buses=(F1-2 Genbus1)
~           conns='wye wye'
~           kvs="12.47 0.48"
~           kvas="5000 5000"
~           XHL=6

new transformer.TR2 phases=3 windings=2
~           buses=(F2-2 Genbus2)
~           conns='delta wye'
~           kvs="12.47 0.48"
~           kvas="5000 5000"
~           XHL=6

new transformer.TR3 phases=3 windings=2
~           buses=(F3-2 Genbus3)
~           conns='delta delta'
~           kvs="12.47 0.48"
~           kvas="5000 5000"
~           XHL=6

! put some loads on the transformers
new load.load1 bus1=Genbus1 phases=3 kv=0.48 kw=1000.0 pf=0.88 model=1 class=1
status=fixed
new load.load2 bus1=Genbus2 phases=3 kv=0.48 kw=1000.0 pf=0.88 model=1 class=1
status=fixed
new load.load3 bus1=Genbus3 phases=3 kv=0.48 kw=1000.0 pf=0.88 model=1 class=1
status=fixed


! Define monitors at the secondary buses to pick up voltage magnitudes

New Monitor.TR1 element=Transformer.TR1 2  mode=32
New Monitor.TR2 element=Transformer.TR2 2  mode=32
New Monitor.TR3 element=Transformer.TR3 2  mode=32


! Define voltage bases so voltage reports come out in per unit
Set voltagebases="115 12.47 .48"
Calcv

! define all kinds of faults at one bus (will be moved later)
! in Monte Carlo Fault mode, only one will be chosen at a time
New Fault.F1   bus1=F1-1.1 phases=1 r=2
New Fault.F2   bus1=F1-1.2 phases=1 r=2
New Fault.F3   bus1=F1-1.3 phases=1 r=2
New Fault.FALL bus1=F1-1   phases=3 r=2
New Fault.F12  bus1=F1-1.1 bus2=F1-1.2 phases=1 r=2
New Fault.F23  bus1=F1-1.2 Bus2=F1-1.3 phases=1 r=2

Set loadmodel=a
Set Toler=0.001 Random=uniform
```

COM Interface Reference

OpenDSSEngine.DSS Interface Definition

Enumerations

This section lists enumerations exposed by OpenDSSEngine.

```
Public Enum MonitorModes
        dssVI=0
        dssPower=1
        dssSequence=16
        dssMagnitude=32
        dssPosOnly=64
End Enum
Public Enum Options
        dssPowerFlow=1
        dssAdmittance=2
        dssNormalSolve=0
        dssNewtonSolve=1
        dssStatic=0
        dssEvent=1
        dssTime=2
        dssMultiphase=0
        dssPositiveSeq=1
        dssGaussian=1
        dssUniform=2
        dssLogNormal=3
        dssAddGen=1
        dssAddCap=2
End Enum
Public Enum SolveModes
        dssSnapShot=0
        dssDutyCycle=6
        dssDirect=7
        dssDaily=1
        dssMonte1=3
        dssMonte2=10
        dssMonte3=11
        dssFaultStudy=9
        dssYearly=2
        dssMonteFault=8
        dssPeakDay=5
        dssLD1=4
        dssLD2=12
        dssAutoAdd=13
End Enum
```

Interfaces

This section lists the Classes exposed by OpenDSSEngine. For each class, the methods and events are listed.

IBus {4CE105CD-E76E-11D2-B339-00A024DB8C85}

<u>Methods</u>
Property Get Name() As String
Name of Bus_F
Property Get NumNodes() As Long
Number of Nodes this bus._F
Property Get Voltages() As Variant
Complex array of voltages at this bus._F
Property Get SeqVoltages() As Variant
Double Array of sequence voltages at this bus._F
Property Get Nodes() As Variant
Integer Array of Node Numbers defined at the bus._F
Property Get Voc() As Variant
Open circuit voltage; Complex array.*#4ß_
Property Get Isc() As Variant
Short circuit currents at bus; Complex Array.
<u>Events</u>
None

ICircuit {4CE105CA-E76E-11D2-B339-00A024DB8C85}

<u>Methods</u>
Property Get Name() As String
Name of the active circuit.*#4*#a*#4*#e_*#10_*#12_*#1
Property Get NumCktElements() As Long
Number of CktElements in the circuit.*#4*#a*#4*#e_*#10_*#12_*#1
Property Get NumBuses() As Long
Total number of Buses in the circuit.*#4*#a*#4*#e_*#10_*#12_*#1
Property Get NumNodes() As Long
Total number of nodes in the circuit.*#4*#a*#4*#e_*#10_*#12_*#1
Property Get Buses(ByVal Index As Variant) As IBus
Collection of Buses in the circuit
Property Get CktElements(ByVal Idx As Variant) As ICktElement
Collection of CktElements in Circuit
Property Get Losses() As Variant
Total losses in active circuit, complex number (two-element array of double).
Property Get LineLosses() As Variant
Complex total line losses in the circuit
Property Get SubstationLosses() As Variant
Complex losses in all transformers designated to substations.
Property Get TotalPower() As Variant
Total power, watts delivered to the circuit
Property Get AllBusVolts() As Variant

Complex array of all bus, node voltages from most recent solution
Property Get AllBusVmag() As Variant
Array of magnitudes (doubles) of voltages at all buses
Property Get AllElementNames() As Variant
Array of strings containing all element names.
Property Get ActiveElement() As ICktElement
Return an interface to the active circuit element
Sub Disable(ByVal Name As String)
Disable a circuit element by name (removes from circuit but leave in database)
Sub Enable(ByVal Name As String)
Activate (enable) a disabled device.
Property Get Solution() As ISolution
Return an interface to the Solution object.
Property Get ActiveBus() As IBus
Return an interface to the active bus.
Function FirstPCElement() As Long
Sets the first Power Conversion (PC) element to be the active element.
Function NextPCElement() As Long
Gets next PC Element. Returns 0 if no more.
Function FirstPDElement() As Long
Sets the first Power Delivery (PD) element to be the active element.
Function NextPDElement() As Long
Gets next PD Element. Returns 0 if no more.
Property Get AllBusNames() As Variant
Array of strings containing names of all buses in circuit.
Property Get AllElementLosses() As Variant
Array of total losses (complex) in each circuit element
Sub Sample
Force all Meters and Monitors to take a sample.
Sub SaveSample
Force all meters and monitors to save their current buffers.
Property Get Monitors() As IMonitors
Returns a Monitors object interface.
Property Get Meters() As IMeters
Returns a Meters object interface (Energy Meters)
Property Get Generators() As IGenerators
Returns a Generators Object interface
Property Get Settings() As ISettings
Returns interface to Settings interface.
Property Get Lines() As ILines
Returns Interface to Line elements interface.
Function SetActiveElement(ByVal FullName As String) As Long
Sets the Active Circuit Element using the full object name (e.g. "generator.g1"). Returns 0 if not found. Else positive integer.
Events
None

ICktElement {4CE105C6-E76E-11D2-B339-00A024DB8C85}

Methods
Property Get Name() As String
Name of Circuit Element*#4*#2ß_
Property Get NumTerminals() As Long
Number of Terminals this Circuit Element*#4*#2ß_
Property Get NumConductors() As Long
Number of Conductors per Terminal*#4*#2ß_
Property Get NumPhases() As Long
Number of Phases*#4*#2ß_
Property Get BusNames() As Variant
Set Names of the Buses to which each terminal is connected.*#2ß_
Property Let BusNames(RHS As Variant)
Set Names of the Buses to which each terminal is connected.*#2ß_
Property Get Properties(ByVal Indx As Variant) As IDSSProperty
Collection of Properties for this Circuit Element (0 based index, if numeric)
Property Get Voltages() As Variant
Complex array of voltages at terminals
Property Get Currents() As Variant
Complex array of currents into each conductor of each terminal
Property Get Powers() As Variant
Complex array of powers into each conductor of each terminal
Property Get Losses() As Variant
Total losses in the element: two-element complex array
Property Get PhaseLosses() As Variant
Complex array of losses by phase
Property Get SeqVoltages() As Variant
Double array of symmetrical component voltages at each 3-phase terminal
Property Get SeqCurrents() As Variant
Double array of symmetrical component currents into each 3-phase terminal
Property Get SeqPowers() As Variant
Double array of sequence powers into each 3-phase teminal
Property Get Enabled() As Boolean
Boolean indicating that element is currently in the circuit.
Property Let Enabled(RHS As Boolean)
Boolean indicating that element is currently in the circuit.
Property Get NormalAmps() As Double
Normal ampere rating
Property Let NormalAmps(RHS As Double)
Normal ampere rating
Property Get EmergAmps() As Double
Emergency Ampere Rating
Property Let EmergAmps(RHS As Double)
Emergency Ampere Rating
Sub Open(ByVal Term As Long, ByVal Phs As Long)
Open the specified terminal and phase, if non-zero. Else all conductors at terminal.
Sub Close(ByVal Term As Long, ByVal Phs As Long)
Close the specified terminal and phase, if non-zero. Else all conductors at terminal.
Function IsOpen(ByVal Term As Long, ByVal Phs As Long) As Boolean
Boolean indicating if the specified terminal and, optionally, phase is open.

Property Get NumProperties() As Long
Number of Properties this Circuit Element.
Property Get AllPropertyNames() As Variant
Variant array containing all property names of the active device.
Events
None

IDSS {4CE105CF-E76E-11D2-B339-00A024DB8C85}

Methods
Property Get NumCircuits() As Long
Number of Circuits currently defined*#4*#8*#0*#6_*#16ß_
Property Get Circuits(ByVal Idx As Variant) As ICircuit
Collection of Circuit objects*#4*#8*#0*#6_*#16ß_
Property Get ActiveCircuit() As ICircuit
Returns interface to the active circuit.*#4*#8*#0*#6_*#16ß_
Property Get Text() As IText
Returns the DSS Text (command-result) interface.*#4*#8*#0*#6_*#16ß_
Property Get Error() As IError
Returns Error interface.*#4*#8*#0*#6_*#16ß_
Function NewCircuit(ByVal Name As String) As ICircuit
Make a new circuit and return interface to active circuit.
Sub ClearAll
Clears all circuit definitions.
Sub ShowPanel
Shows control panel for DSS.
Function Start(ByVal code As Long) As Boolean
Validate the user and start the DSS. (old method) Returns TRUE if successful.
Property Get Version() As String
Get version string for the DSS.
Property Get DSSProgress() As IDSSProgress
Gets interface to the DSS Progress Meter
Events
None

IDSSProgress {FAA79841-525D-11D3-AD84-444553540000}

Methods
Property Let PctProgress(RHS As Long)
Percent progress to indicate [0..100]*#4ß_
Property Let Caption(RHS As String)
Caption to appear on the bottom of the DSS Progress form.
Sub Show
Shows progress form with null caption and progress set to zero.
Sub Close
Closes (hides) DSS Progress form.
Events
None

IDSSProperty {4CE105C4-E76E-11D2-B339-00A024DB8C85}

Methods
Property Get Name() As String
Name of Property*#4ß_
Property Get Description() As String
Description of the property.*#4ß_
Property Get Val() As String

Property Let Val(RHS As String)

Events
None

IError {4CE105C8-E76E-11D2-B339-00A024DB8C85}

Methods
Property Get Number() As Long
Error Number*#4ß_
Property Get Description() As String
Description of error for last operation*#4ß_
Events
None

IGenerators {4FFBDD30-0474-11D3-B339-00A024DB8C85}

Methods
Property Get AllNames() As Variant
Array of names of all Generator objects.*#4ß_
Property Get RegisterNames() As Variant
Array of Names of all generator energy meter registers
Property Get RegisterValues() As Variant
Array of valus in generator energy meter registers.
Property Get First() As Long
Sets first Generator to be active. Returns 0 if none.
Property Get Next() As Long
Sets next Generator to be active. Returns 0 if no more.
Property Get ForcedON() As Boolean
Indicates whether the generator is forced ON regardles of other dispatch criteria.
Property Let ForcedON(RHS As Boolean)
Indicates whether the generator is forced ON regardles of other dispatch criteria.
Property Get Name() As String
Sets a generator active by name.
Property Let Name(RHS As String)
Sets a generator active by name.
Events
None

ILines {083E2420-7B74-11D4-B33A-00A024DB8C85}

Methods

Property Get Name() As String
Specify the name of the Line element to set it active.
Property Let Name(RHS As String)
Specify the name of the Line element to set it active.
Property Get AllNames() As Variant
Names of all Line Objects
Property Get First() As Long
Invoking this property sets the first element active. Returns 0 if no lines. Otherwise, index of the line element.
Property Get Next() As Long
Invoking this property advances to the next Line element active. Returns 0 if no more lines. Otherwise, index of the line element.
Function New(ByVal Name As String) As Long
Creates a new Line and makes it the Active Circuit Element.__
Property Get Bus1() As String
Name of bus for terminal 1.
Property Let Bus1(RHS As String)
Name of bus for terminal 1.
Property Get Bus2() As String
Name of bus for terminal 2.
Property Let Bus2(RHS As String)
Name of bus for terminal 2.
Property Get LineCode() As String
Name of LineCode object that defines the impedances.
Property Let LineCode(RHS As String)
Name of LineCode object that defines the impedances.
Property Get Length() As Double
Length of line section in units compatible with the LineCode definition.
Property Let Length(RHS As Double)
Length of line section in units compatible with the LineCode definition.
Property Get Phases() As Long
Number of Phases, this Line element.
Property Let Phases(RHS As Long)
Number of Phases, this Line element.
Property Get R1() As Double
Positive Sequence resistance, ohms per unit length.
Property Let R1(RHS As Double)
Positive Sequence resistance, ohms per unit length.
Property Get X1() As Double
Positive Sequence reactance, ohms per unit length.
Property Let X1(RHS As Double)
Positive Sequence reactance, ohms per unit length.
Property Get R0() As Double
Zero Sequence resistance, ohms per unit length.
Property Let R0(RHS As Double)
Zero Sequence resistance, ohms per unit length.
Property Get X0() As Double
Zero Sequence reactance ohms per unit length.
Property Let X0(RHS As Double)

Zero Sequence reactance ohms per unit length.
Property Get C1() As Double
Positive Sequence capacitance, nanofarads per unit length.
Property Let C1(RHS As Double)
Positive Sequence capacitance, nanofarads per unit length.
Property Get C0() As Double
Zero Sequence capacitance, nanofarads per unit length.
Property Let C0(RHS As Double)
Zero Sequence capacitance, nanofarads per unit length.
Property Get Rmatrix() As Variant
Resistance matrix (full), ohms per unit length. Variant array of doubles.
Property Let Rmatrix(RHS As Variant)
Resistance matrix (full), ohms per unit length. Variant array of doubles.
Property Get Xmatrix() As Variant

Property Let Xmatrix(RHS As Variant)

Property Get Cmatrix() As Variant

Property Let Cmatrix(RHS As Variant)

Property Get NormAmps() As Double
Normal ampere rating of Line.
Property Let NormAmps(RHS As Double)
Normal ampere rating of Line.
Property Get EmergAmps() As Double
Emergency (maximum) ampere rating of Line.
Property Let EmergAmps(RHS As Double)
Emergency (maximum) ampere rating of Line.'E
Events
None

IMeters {4FFBDD2C-0474-11D3-B339-00A024DB8C85}

Methods
Property Get AllNames() As Variant
Array of all energy Meter names
Property Get First() As Long
Set the first energy Meter active. Returns 0 if none.
Property Get Next() As Long
Sets the next energy Meter active. Returns 0 if no more.
Property Get RegisterNames() As Variant
Array of strings containing the names of the registers.
Property Get RegisterValues() As Variant
Array of all the values contained in the Meter registers for the active Meter.
Sub Reset
Resets registers of active Meter.
Sub ResetAll
Resets registers of all Meter objects.

Sub Sample
Forces active Meter to take a sample._
Sub Save
Saves meter register values._
Property Get Name() As String
Set a meter to be active by name._
Property Let Name(RHS As String)
Set a meter to be active by name._
Events
None

IMonitors {4FFBDD28-0474-11D3-B339-00A024DB8C85}

Methods
Property Get AllNames() As Variant
Array of all Monitor Names
Property Get First() As Long
Sets the first Monitor active. Returns 0 if no monitors.
Property Get Next() As Long
Sets next monitor active. Returns 0 if no more.
Sub Reset
Resets active Monitor object.
Sub ResetAll
Resets all Monitor Objects
Sub Sample
Causes active Monitor to take a sample.
Sub Save
Causes active monitor to save its current buffer to disk.
Sub Show
Converts monitor file to text and displays with text editor
Property Get FileName() As String
Name of disk file associated with active Monitor.
Property Get Mode() As Long
Set Monitor mode (bitmask integer - see DSS Help)
Property Let Mode(RHS As Long)
Set Monitor mode (bitmask integer - see DSS Help)
Property Get Name() As String
Sets the active Monitor object by name
Property Let Name(RHS As String)
Sets the active Monitor object by name
Events
None

ISettings {A7ED88D0-735A-11D4-B33A-00A024DB8C85}

Methods
Property Get AllowDuplicates() As Boolean
True | False* Designates whether to allow duplicate names of objects
Property Let AllowDuplicates(RHS As Boolean)
True | False* Designates whether to allow duplicate names of objects

Property Get ZoneLock() As Boolean
True | False* Locks Zones on energy meters to prevent rebuilding if a circuit change occurs.
Property Let ZoneLock(RHS As Boolean)
True | False* Locks Zones on energy meters to prevent rebuilding if a circuit change occurs.
Property Let AllocationFactors(RHS As Double)
Sets all load allocation factors for all loads defined by XFKVA property to this value.
Property Get AutoBusList() As String
List of Buses or (File=xxxx) syntax for the AutoAdd solution mode.
Property Let AutoBusList(RHS As String)
List of Buses or (File=xxxx) syntax for the AutoAdd solution mode.
Property Get CktModel() As Long
dssMultiphase * | dssPositiveSeq IIndicate if the circuit model is positive sequence.
Property Let CktModel(RHS As Long)
dssMultiphase * | dssPositiveSeq IIndicate if the circuit model is positive sequence.
Property Get NormVminpu() As Double
Per Unit minimum voltage for Normal conditions.
Property Let NormVminpu(RHS As Double)
Per Unit minimum voltage for Normal conditions.
Property Get NormVmaxpu() As Double
Per Unit maximum voltage for Normal conditions.
Property Let NormVmaxpu(RHS As Double)
Per Unit maximum voltage for Normal conditions.
Property Get EmergVminpu() As Double
Per Unit minimum voltage for Emergency conditions.
Property Let EmergVminpu(RHS As Double)
Per Unit minimum voltage for Emergency conditions.
Property Get EmergVmaxpu() As Double
Per Unit maximum voltage for Emergency conditions.
Property Let EmergVmaxpu(RHS As Double)
Per Unit maximum voltage for Emergency conditions.
Property Get UEweight() As Double
Weighting factor applied to UE register values.
Property Let UEweight(RHS As Double)
Weighting factor applied to UE register values.
Property Get LossWeight() As Double
Weighting factor applied to Loss register values.
Property Let LossWeight(RHS As Double)
Weighting factor applied to Loss register values.
Property Get UEregs() As Variant
Array of Integers defining energy meter registers to use for computing UE
Property Let UEregs(RHS As Variant)
Array of Integers defining energy meter registers to use for computing UE
Property Get LossRegs() As Variant
Integer array defining which energy meter registers to use for computing losses
Property Let LossRegs(RHS As Variant)
Integer array defining which energy meter registers to use for computing losses
Property Get Trapezoidal() As Boolean

True | False * Gets value of trapezoidal integration flag in energy meters.
Property Let Trapezoidal(RHS As Boolean)
True | False * Gets value of trapezoidal integration flag in energy meters.
Property Get VoltageBases() As Variant
Array of doubles defining the legal voltage bases in kV L-L
Property Let VoltageBases(RHS As Variant)
Array of doubles defining the legal voltage bases in kV L-L
Property Get ControlTrace() As Boolean
True | False* Denotes whether to trace the control actions to a file.
Property Let ControlTrace(RHS As Boolean)
True | False* Denotes whether to trace the control actions to a file.
Property Get PriceSignal() As Double
Price Signal for the Circuit
Property Let PriceSignal(RHS As Double)
Price Signal for the Circuit
Property Get PriceCurve() As String
Name of LoadShape object that serves as the source of price signal data for yearly simulations, etc.
Property Let PriceCurve(RHS As String)
Name of LoadShape object that serves as the source of price signal data for yearly simulations, etc.
Events
None

ISolution {4CE105D1-E76E-11D2-B339-00A024DB8C85}

Methods
Sub Solve
Execute solution for present solution mode.
Property Get Mode() As Long
Set present solution mode (by a text code - see DSS Help)
Property Let Mode(RHS As Long)
Set present solution mode (by a text code - see DSS Help)
Property Get Frequency() As Double
Set the Frequency for next solution
Property Let Frequency(RHS As Double)
Set the Frequency for next solution
Property Get Hour() As Long
Set Hour for time series solutions.
Property Let Hour(RHS As Long)
Set Hour for time series solutions.
Property Get Seconds() As Double
Seconds from top of the hour._
Property Let Seconds(RHS As Double)
Seconds from top of the hour.
Property Get StepSize() As Double
Time step size in sec
Property Let StepSize(RHS As Double)
Time step size in sec

Property Get Year() As Long
Set year for planning studies
Property Let Year(RHS As Long)
Set year for planning studies
Property Get LoadMult() As Double
Default load multiplier applied to all non-fixed loads
Property Let LoadMult(RHS As Double)
Default load multiplier applied to all non-fixed loads
Property Get Iterations() As Long
Number of iterations take for last solution.
Property Get MaxIterations() As Long
Max allowable iterations.
Property Let MaxIterations(RHS As Long)
Max allowable iterations.
Property Get Tolerance() As Double
Solution convergence tolerance.
Property Let Tolerance(RHS As Double)
Solution convergence tolerance.
Property Get Number() As Long
Number of solutions to perform for Monte Carlo and time series simulations
Property Let Number(RHS As Long)
Number of solutions to perform for Monte Carlo and time series simulations
Property Get Random() As Long
Randomization mode for random variables "Gaussian" or "Uniform"
Property Let Random(RHS As Long)
Randomization mode for random variables "Gaussian" or "Uniform"
Property Get ModeID() As String
ID (text) of the present solution mode
Property Get LoadModel() As Long
Load Model: dssPowerFlow (default) | dssAdmittance
Property Let LoadModel(RHS As Long)
Load Model: dssPowerFlow (default) | dssAdmittance
Property Get LDCurve() As String
Load-Duration Curve name for LD modes
Property Let LDCurve(RHS As String)
Load-Duration Curve name for LD modes
Property Get pctGrowth() As Double
Percent default annual load growth rate
Property Let pctGrowth(RHS As Double)
Percent default annual load growth rate
Property Get AddType() As Long
Type of device to add in AutoAdd Mode: dssGen (Default) | dssCap
Property Let AddType(RHS As Long)
Type of device to add in AutoAdd Mode: dssGen (Default) | dssCap
Property Get GenkW() As Double
Generator kW for AutoAdd modedssGen (Default) | dssCap
Property Let GenkW(RHS As Double)
Generator kW for AutoAdd modedssGen (Default) | dssCap
Property Get GenPF() As Double

PF for generators in AutoAdd modedssGen (Default) | dssCap
Property Let GenPF(RHS As Double)
PF for generators in AutoAdd mode‘E
Property Get Capkvar() As Double
Capacitor kvar for adding capacitors in AutoAdd mode‘E
Property Let Capkvar(RHS As Double)
Capacitor kvar for adding capacitors in AutoAdd mode
Property Get Algorithm() As Long
Base Solution algorithm: dssNormalSolve | dssNewtonSolve
Property Let Algorithm(RHS As Long)
Base Solution algorithm: dssNormalSolve | dssNewtonSolve
Property Get ControlMode() As Long
dssStatic* | dssEvent | dssTime Modes for control devices
Property Let ControlMode(RHS As Long)
dssStatic* | dssEvent | dssTime Modes for control devices
Property Get GenMult() As Double
Default Multiplier applied to generators (like LoadMult)
Property Let GenMult(RHS As Double)
Default Multiplier applied to generators (like LoadMult)
Property Get DefaultDaily() As String
Default daily load shape (defaults to "Default")
Property Let DefaultDaily(RHS As String)
Default daily load shape (defaults to "Default")
Property Get DefaultYearly() As String
Default Yearly load shape (defaults to "Default")
Property Let DefaultYearly(RHS As String)
Default Yearly load shape (defaults to "Default")
Events
None

IText {4CE105C1-E76E-11D2-B339-00A024DB8C85}

Methods
Property Get Command() As String
Input command string for the DSS.
Property Let Command(RHS As String)
Input command string for the DSS.
Property Get Result() As String
Result string for the last command.
Events
None