
# Reference Guide

# The Open Distribution System Simulator™ (OpenDSS)

Roger C. Dugan
Davis Montenegro
Electric Power Research Institute, Inc.
June 2021

Other Contributors:
Andrea Ballanti (2016)


ELECTRIC POWER
RESEARCH INSTITUTE

# License

Copyright (c) 2008-2021, Electric Power Research Institute, Inc.

All rights reserved.

**Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:**

**Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.**
**Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.**
**Neither the name of the Electric Power Research Institute, Inc., nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.**

**THIS SOFTWARE IS PROVIDED BY Electric Power Research Institute, Inc., "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL Electric Power Research Institute, Inc., BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.**

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 refers to the open-source implementation of the DSS. It is implemented in three formsL

1. A stand-alone executable program. (OpenDSS.exe)
2. An in-process COM server DLL designed to be driven from a variety of existing software platforms. (OpenDSSEngine.DLL)
3. A Stdcall DLL that provides all the functions of the COM server but can be used from languages that do not support COM or that require Threadsafe execution such as on a Cloud server. (OpenDSSDirect.DLL)

The executable version has a basic text-based user interface on the solution engine to assist users in developing scripts and viewing solutions.

The program supports nearly all rms steady-state (i.e., frequency domain) analyses commonly performed for utility distribution systems planning and analysis. 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 utilities worldwide and the advent of the "smart grid". 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, smart grid applications, 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
- Risk-based Distribution Planning Studies
- Probabilistic Planning Studies
- Solar PV System Simulation
- Wind Plant Simulations
- Nuclear Plant Station Auxiliary Transformer Modeling
- Distribution Automation Control Assessment
- Protection System Simulation
- Storage Modeling
- Distribution Feeder Simulation with AMI Data
- Distribution State Estimation
- Ground Voltage Rise on Transmission Systems
- Geomagnetically-Induced Currents (GIC)
- EV Impacts Simulations
- Co-simulation of Power and Communications Networks
- Analysis of Unusual Transformer Configurations
- Harmonic and Interharmonic Distortion Analysis

- Neutral-to-earth Voltage Simulations
- Development of IEEE Test feeder cases
- Phase Shifter Simulation
- Arc Furnace Simulation
- Impulse Loads (car crushers, etc.)
- And more ….

The program has several built-in solution modes, such as

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

These modes were added as the program evolved to meet the analysis needs of specific projects the authors were involved with. However, the program was designed with the recognition that developers would never be able to anticipate everything users will want to do with it. A Component Object Model (COM) interface was implemented on the in-process server DLL version of the program to allow knowledgable users to use the features of the program to perform new types of studies. The direct function call DLL was added later to provide the features of the COM interface to computer languages or platforms that do not support COM, which is primarily for Microsoft Windows.

Through the COM interface, the user can design and execute custom solution modes and features from an external program 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, Python, C#, R, and other languages. This provides powerful external analytical capabilities as well as excellent graphics for displaying results.

Many users find the text scripting interface of the stand-alone executable version enough for nearly all their work. As users find themselves repeatedly needing a feature for their work, the feature is implemented within the built-in solution control module and connected to the text-based command interface.

The COM interface and direct-call DLL interface also provide direct access to the text-based command interface as well as access to numerous methods and properties for accessing many of the properties of the simulator's models. Through the text-based 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 in programming environments

and provide some database-like characteristics (although the program does not technically have a database). Many of the results can be retrieved through the COM interface as well as from various output files. Many output or export files are written in Comma-Separated Value (CSV) format that be imported easily into other tools such as Microsoft Excel or MATLAB® 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. 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 nominal OpenDSS structure is depicted in Figure 1

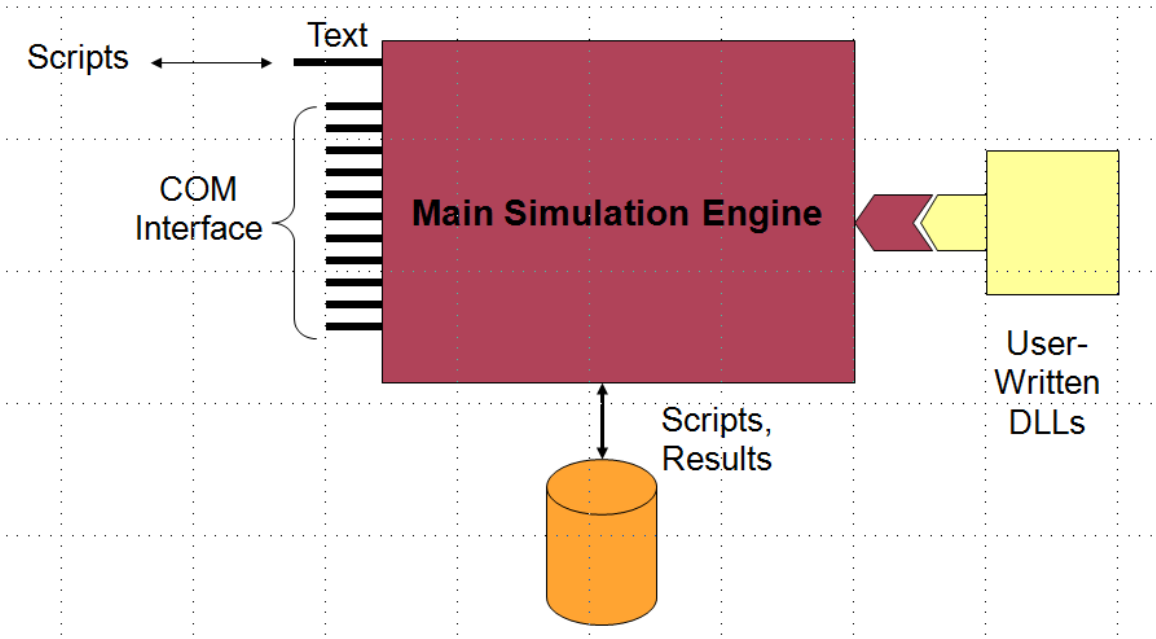



**Figure 1. OpenDSS Structure**

At Version 7.6, a 64-bit version of the program was introduced to accommodate the needs of users needing to access the program from a 64-bit program or to run very large models. Both the 32-bit and 64-bit versions can coexist on the same computer to support users who need both versions. For example, MATLAB is frequently installed as a 64-bit app while Microsoft Office programs are frequently installed as 32-bit applications. Windows will automatically select the proper version of OpenDSS.

# Brief History and Objectives of OpenDSS

Development of the OpenDSS program began in April 1997 at Electrotek Concepts, Inc. At that time the program was simply called "DSS" for Distribution System Simulator. 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 an open source license 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 functional spec for a simulator that would support most of the 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 provide a useful tool to support the needs of a consulting company that was involved in leading-edge research in distribution system analysis.

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 frequently having to adjust our models and add new features to our tools. Thus, we needed a tool that would not lock us in to 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 different strengths and various modeling limitations. One design goal was to develop an object-oriented circuit description language that minimized the conversion effort.
3. 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 approach gets around much of the limited-dialog issue and allows the user to better adapt the analysis process to the problem at hand.
4. 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. This is the concept of "distribution" used by much of the rest of the world outside North America.

5. A key capability desired for the tool was to capture both 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 to support sequential-time simulations.
6. There are many instances where it becomes necessary to model elements with many phases – not just one, two, or 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 with many traditional distribution system analysis tools.
7. 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 enough for DG interconnection evaluation. Simple dynamics models are presently built into the program and this feature continues to be developed.
8. 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.
9. 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.
10. We wanted a program that would simulate the behavior of devices on the distribution system as they would 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 most 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. New features are continually being added to support ongoing research into distribution system planning and analysis at EPRI.

The OpenDSS 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 analyses 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). It does perform *electromechanical* transients, or dynamics analysis.

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. 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 mostly for power flow studies, but it gives the tool great modeling flexibility, particularly for accommodating all sorts of load models and unusual circuit configurations. It is easier to make a harmonics flow simulation program solve the power flow problem than the opposite.

## 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 Embarcadero.com). The various sparse matrix solvers employed by the DSS over the years have been written in C and C++.

Since Pascal is seldom taught in engineering and computer science education in the US anymore, 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. Another reason stems from the desire to have a COM interface to allow easy access from other programs. Delphi provided an easy way to build these interfaces early on. 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 as well as many other languages 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 entire program takes only 3 seconds on a Dell Precision 5510 running Delphi 10.2 Tokyo compiler. 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 that it was not. You should find that the OpenDSS processes large circuit description scripts with relative ease. The math processing speed appears to be comparable to any natively compiled programming language. This has improved over the years as the Delphi environment has introduced function in-lining, which has been exploited in the program for several years now.

4. Writing COM interfaces and DLLs are relatively easy tasks for a programmer.
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 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, there are open source Pascal compilers available on the Web, such as the Free Pascal Compiler (FPC) (Lazarus IDE), for which we strive to maintain as much compatibility with as possible. There are many conditional compiles in the code to permit compatible compilation with Delphi and FPC.

## NOTES ON THE FUTURE …

OpenDSS incorporates parallel processing capabilities since year 2016. This version was the birth of version 8, which was used to perfection the parallel processing suite, now part of the base in OpenDSS.

Nowadays, OpenDSS has capabilities for parallel execution of

1. Long time series simulations by dividing the time (year, for example) into segments and assigning each temporal segment to a separate CPU.
2. Diakoptic parallelization of large circuits by splitting the circuits into several smaller circuits and assigning each to a separate CPU.
3. Multiple circuits at the same time.

You can manage the parallel processes directly from the OpenDSS scripting language. There is no need for 3rd party software to manage the parallel processing.

# Installation

This installation procedure applies to Version 7.6 and later versions.

The 7.6 version was the first to be delivered in both 32-bit (X86) and 64-bit (X64) versions. The OpenDSSInstaller download includes both, along with documentation and examples.

If you have 64-bit Windows, you may install both the 64-bit and 32-bit versions. The 32-bit version is required if you plan to automate OpenDSS from Excel or any other 32-bit program. The 64-bit version is required to automate OpenDSS from 64-bit MatLab or Python on a 64-bit system.

The installer will give you a choice to install the executables and optional files under a target directory of your choice, such as C:\Program Files\OpenDSS. Files that are specific to the 32-bit version will be written to an x86 subdirectory, such as c:\Program Files\opendss\x86. Files that are specific to the 64-bit version will be written to an x64 subdirectory, such as c:\Program Files\opendss\x64. The EXE and DLL files should not be moved after installation but may be updated in place with newer versions.

Due to corporate IT restrictions you may find it difficult to run OpenDSS out of the Program Files folder. This is usually a privileges issue. If so, simply install to another location where you have enough privileges.

On a 64-bit system, you may install and use both the 32-bit and 64-bit versions with no conflict between them.

Short-cuts to the program and manual are created under Start Menu/OpenDSS.

The most up-to-date reference information on objects and their properties will always be found through the software's "Help / DSS Help" menu command.

## COM Automation

The COM Server in OpenDSSEngine.DLL may be automated. The installer will register either or both versions, depending on your selection. Even though the file names and registration commands match, they are in separate locations and Windows will activate the correct version required by the calling program. For example, 64-bit MATLAB will call the 64-bit OpenDSSEngine.DLL and 32-bit Microsoft Excel will call the 32-bit version.

(Note: In corporate IT implementations of Microsoft Office, it is common for only the 32-bit version to be installed.)

## Files

The OpenDSS Installer will automatically register the COM servers during the installation process. However, you can still do it manually if you prefer. Although it has gotten a little more complicated, installation of the OpenDSS program is still 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 or C:\Users\MyUserName\OpenDSS) and start the program.

The following are the key program files:

1. OpenDSS.exe (Stand alone executable)
2. OpenDSSEngine.DLL (The in-process COM server version)
3. KLUSolve.DLL (Sparse matrix solver)
4. DSSView.exe (Viewer for DSS graphics output)
5. OpenDSSDirect.DLL (Direct call DLL – alternative to the COM interface)

The **OpenDSSDirect.DLL** file is a relatively new option for driving OpenDSS without using the COM interface. It is a stdcall DLL that mimics the COM interface in many aspects but has special functions as well. You can use this to basically build OpenDSS into your application or use OpenDSS to do parallel processing on systems with this capability (such as with the Julia language, for example, which does not support COM). For documentation, see **OpenDSS_Direct_DLL.pdf** in the ..\Doc directory where OpenDSS was installed.

The **OpenDSSEngine.DLL** is an *in-process* COM server that will have to be registered if you intend to access it from other programs/languages such as MATLAB and VBA in MS Office. This will occur automatically if you use the installer from the download.

### *Manual Registration of COM Server*

If you do not intend to automate OpenDSS, and simply use the OpenDSS.EXE stand-alone version, you could skip this step and simply copy the EXE and DLL files to some convenient location on your disk. Afterward, you can manually register the server by issuing the following command to the (DOS) command prompt when in the folder you have placed the files:

```
Regsvr32  OpenDSSEngine.DLL
```

The **RegisterDSSEngine.BAT** file provided within the standard zip file download to perform this action. Registration need only be done once. You may simply double-click on the .BAT file and it will execute the registration.

**Note**: You may need Administrator privileges on your computer to do this. This applies more to Windows 10, 8, 7 and Vista than previous versions of Windows.

On Windows Vista, 7, 8, and 10 you will have to execute the .BAT file from an elevated Admin status. One way to do this is right click on the Windows buttonand select **Command Prompt (Admin)**. Then run the .BAT file from the command window, or DOS window. Some corporate IT policies may require you to go through additional steps. It may be necessary to have an administrator install the intial copy of OpenDSS. Once installed, you can simply update by copying the newer EXE or DLL over the older one in most cases.

After registration, if you start the Windows registry editor (Type 'regedit' in the command box on the start menu) you will find the OpenDSSEngine listed under Classes as shown in Figure 2. If you then look up the GUID in the registry, it should point to the OpenDSSEngine.DLL file in the folder where you installed it (Figure 3).

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');
```

When both the 32-bit and 64-bit versions are available, Windows will automagically load the appropriate one for your application.

Except for the 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 OpenDSS through the text-based window interface provided. Many users find this mode adequate 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 file to your start menu (right-click on the EXE file and look at your options).

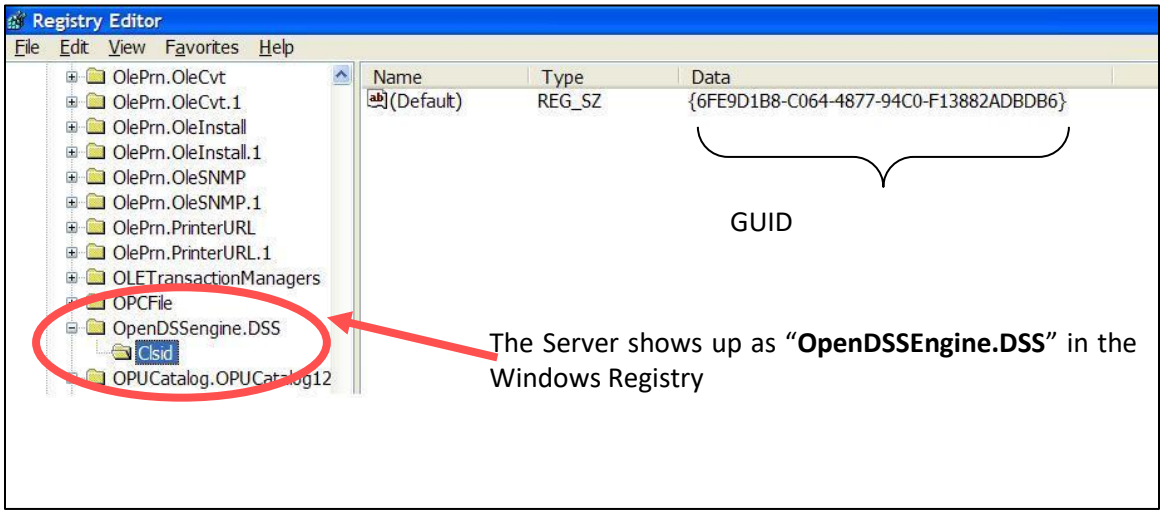



**Figure 2. OpenDSSEngine in the Windows Registry**

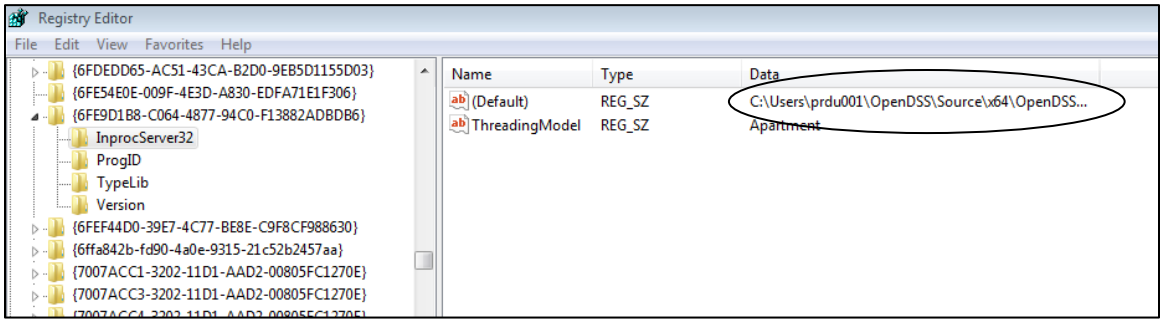



**Figure 3. GUID points to the In-Process Server File (OpenDSSEngine.DLL)**

The key shown is for the 64-bit version CLSID on Windows pointing to the server in the x64 folder. The full registry key is:

HKEY_CLASSES_ROOT\CLSID\{6FE9D1B8-C064-4877-94C0-F13882ADBDB6}

Note that on 64-Bit Windows systems, the 32-bit server version CLSID will be registered under the

*Wow6432Node* key with a key like this:

HKEY_CLASSES_ROOT\Wow6432Node\CLSID\{6FE9D1B8-C064-4877-94C0-F13882ADBDB6}

This is the registry key for the 32-bit "Windows on Windows" (WoW) part of the registry. This is how Windows knows which version of the server to load even though the GUIDs are the same.

Although there are many interfaces in the OpenDSS COM interface, only one is registered: the DSS interface. The program automatically instantiates the rest of the interfaces when it starts. Then you can define some variables in your program to the various interfaces to make it convenient to access the intefaces. For example, our common startup routine in Excel VBA is:

```
Public Sub StartDSS()

' Create a new instance of the DSS
    Set DSSobj = New OpenDSSEngine.DSS

' Start the DSS
    If Not DSSobj.Start(0) Then
        MsgBox "DSS Failed to Start"
    Else
        ' MsgBox "DSS Started successfully"
        ' Assign a variable to each of the  interfaces for easier access
        Set DSSText = DSSobj.Text
        Set DSSCircuit = DSSobj.ActiveCircuit
        Set DSSSolution = DSSCircuit.Solution
        Set DSSControlQueue = DSSCircuit.CtrlQueue
        Set DSSCktElement = DSSCircuit.ActiveCktElement
        Set DSSPDElement = DSSCircuit.PDElements
        Set DSSMeters = DSSCircuit.Meters
        Set DSSBus = DSSCircuit.ActiveBus
        Set DSSCmath = DSSobj.CmathLib
        Set DSSParser = DSSobj.Parser
        Set DSSIsources = DSSCircuit.ISources
        Set DSSMonitors = DSSCircuit.Monitors
        Set DSSLines = DSSCircuit.Lines


        Range("DSSVersion").Value = "Version:  " + DSSobj.Version
        Beep
    End If


End Sub
```

## OTHER REGISTRY ENTRIES

Note: The ".ini" files that were created in early versions of OpenDSS no longer exist on Windows. The Windows Registry is now used for this purpose. The values are stored under the *HKEY_USERS/Software/OpenDSS* key. See Figure 4.

The purpose of saving these values is to remember the contents of the user control panels and some global settings such as the default Base Frequency and the path for the user's preferred editor. Thus, the program will start the next time with the same script window contents and defaults as at the end of the previous session that is successfully closed.

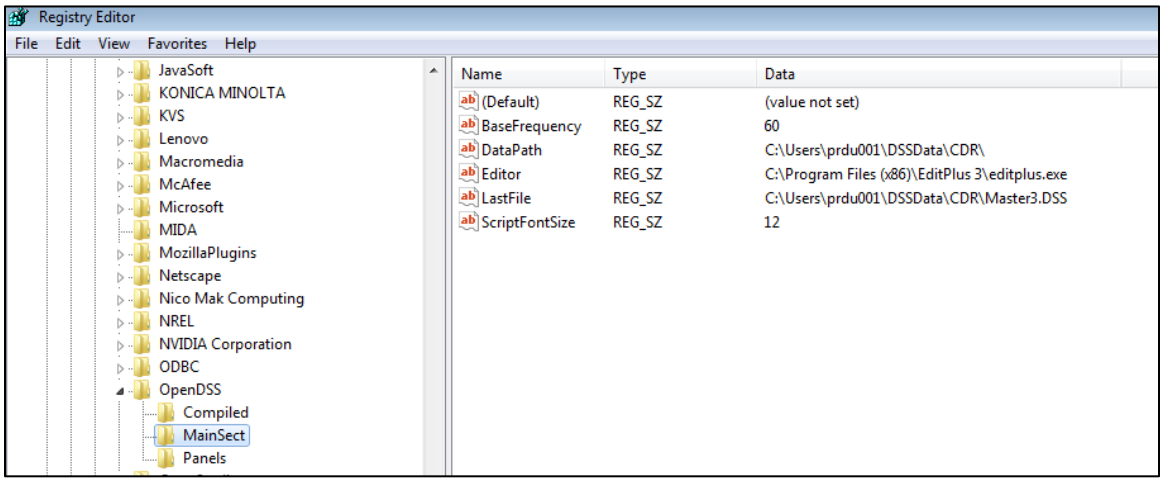



**Figure 4. OpenDSS Values in Registry**

## OTHER FILES

*DSSView.EXE* is a custom 32-bit program that can read special output files from the OpenDSS Plot command and display simple charts and graphs. The program is automatically started by the OpenDSS Plot command or you may start it at any time and access previously created .DSV files. OpenDSS does not have to be running to reproduce these plots. Many of these same charts can be reproduced in MATLAB and other program from data exported using the Export command or by accessing the values through the COM interface. See the Python examples.

*DSS Visualization Tool* is a program developed in 2017 to replace and supplement DSSView with advanced graphics. It has a separate installer. It is activated by the **Set DSSVisualizationTool = TRUE** command. The tool is described in the "Advance graphics Module for OpenDSS.PDF" file in the Doc folder distributed with the program.

*IndMach012.DLL* is a DLL file containing a special symmetrical component (012) model of an induction machine that can be connected to a Generator object model. This is provided both as a working model and as an example of how to build such a DLL. The source code is provided on the SourceForge.Net site. Note: The IndMach012 model was built into the OpenDSS program in 2017 as a PCelement. It has the same model as the DLL version.

### *Examples, Docs, and Test Cases*

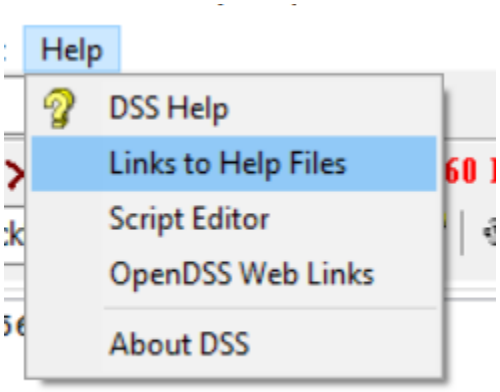
In addition to the program downloads, there are example and documentation downloads. These are available to selectively download folders and files directly from the source code repository. The contents of the Example folders are:

- C_Sharp: Example of using the C# programming language to drive OpenDSS.
- Excel: Examples of use Microsoft Excel to access the COM interface. You will need to allow macros to execute.
- GICExample: Example scripts for geomagnetically-induced current (GIC) test case, including the example in this Manual.
- HarmonicsTMode: Examples of using the HarmonicsTMode for sequential-time

harmonics-mode studies.

- LoadShapes: Several typical annual distribution system loadshapes. Also, solar PV loadshapes.
- Manual: Examples from this document.
- MATLAB: Examples of using MATLAB to drive OpenDSS.
- Python: Some Python code examples for accessing the COM interface.
- Scripts: Several script templates for common, and uncommon, simulation processes. Some conductor (wire) data, too.
- Stevenson: Scripts for the power flow case from the W. Stevenson text book.
- … Other examples are added from time to time.

In the Doc folder are several documents that supplement the information found in this Users Manual. There is a convenient link in the Help menu that provides an annotated index to the help files. This requires the Doc folder to be installed in the default location the installer assumes




The *IEEETestCases* folder contains OpenDSS implementations of most of the IEEE PES Distribution Test Feeders and other IEEE test cases in DSS script format. For more information on these, go to this URL:

http://www.ewh.ieee.org/soc/pes/dsacom/testfeeders/index.html

The *EPRITestCircuit* folders contains DSS script versions of distribution circuit models used in EPRI research projects that have been made available for public use.

## UPDATING OPENDSS

A new version of the Installer is created for major releases. This can be downloaded from Sourceforge.net. In between major releases beta builds are posted on the main website. This normally be in these locations:

https://sourceforge.net/p/electricdss/code/HEAD/tree/trunk/Version8/Distrib/x64/

https://sourceforge.net/p/electricdss/code/HEAD/tree/trunk/Version8/Distrib/x86/

for the 64-bit and 32-bit versions, respectively.

You may update to the latest beta build simply by copying the desired EXE or DLL files over the ones previoiusly installed by the Installer.

Be sure to not mix X64 and X86 versions as these will not work on the opposite type of system.

## OPENDSS-G

A new user interface to OpenDSS has been developed called OpenDSS-G. It has its own site on Sourceforge.Net. It is based on the latest version of OpenDSS version 8. You can download it from here:

https://sourceforge.net/projects/dssimpc/

OpenDSS-G is focused on translating data into information serving as a test bed for new situation awareness methods, planning and operation studies, and control room simulations that can be performed with OpenDSS. OpenDSS-G also bring a framework for facilitating the utilization of advance simulation features included in the most recent versions of OpenDSS, as well as the features and capabilities needed for implementing simulators in different time domains: real-time, fast simulation, and off-line simulation among others.

OpenDSS-G and OpenDSS version will be the main versions supported by EPRI in 2018 and beyond.

# Summary of Simulation Capabilities

The present version of OpenDSS can perform the following analyses/simulations:

## POWER FLOW

While the power flow problem is probably the most common problem solved with the program, the OpenDSS is not best characterized as a power flow program. Its heritage is from general-purpose power system harmonics analysis tools. Thus, it works differently than most existing power flow tools. This heritage also gives it some unique and powerful capabilities. The program was originally designed to perform nearly all aspects of distribution planning for distributed generation (DG), which includes harmonics analysis. It is relatively easy to make a harmonics analysis program solve a power flow, while it is quite difficult to make a power flow program perform harmonics analysis. To learn more about how the algorithm works for the power flow problem, see “Putting It All Together” below.

The OpenDSS is designed to perform a basic distribution-style power flow in which the bulk power system is the dominant source of energy. However, it differs from the traditional radial circuit solvers in that it solves networked (meshed) distribution systems as easily 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 simplified positive-sequence model. The default is a full 3-phase model. Due to the complex multi-phase models that may be created with myriad unbalances, the user may have to create positive-sequence models outside the DSS by defining a single-phase model of the circuit. However, the “MakePosSeq” command will attempt to convert a multi-phase model to a positive-sequence model. (This is not always successful.) By setting the proper flag, all power reports for a positive-sequence equivalent 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 the "Mode" option 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. For solar PV simulation it is common to use 1-s time steps in Dutycycle mode. 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 or provide a myriad of overload and loss information.

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 and works equally well on meshed networks and radial circuits. It works best on systems that

have at least one stiff source.

It is common in distribution power flow papers for the authors to claim that radial distribution feeders with low X/R ratios are difficult to solve. This usually refers to certain traditional Newton-Raphson formulations like those used in positive-sequence transmission system models. To the best of our knowledge, OpenDSS has never suffered from this problem and solves radial circuits quite handily.

The two basic power flow solution types are

1. Iterative power flow
2. Direct solution

For the iterative power flow, nonlinear elements such as 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). The default is “Powerflow”.

There are two iterative power flow algorithms currently employed:

1. "Normal" current injection mode (default)
2. "Newton" mode.

The Normal mode is faster. The Newton mode is somewhat more robust for circuits that are difficult to solve. The default is Normal. The Normal mode solution method is a relatively simple *fixed-point* iterative method and works very well for nearly all distribution systems with a stiff bulk power source. It is the preferred method for Yearly-mode simulations and other lengthy sequential-time simulations due to its speed. It has been updated over the life of OpenDSS to where it is now nearly as robust as the Newton mode.

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. Dynamics mode simulations may also use linear load models or a mixture of linear and nonlinear models.

## FAULT STUDIES

The OpenDSS will perform short-circuit fault 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 faults, 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. The user places one, or more, Fault objects on the system at selected buses, defining the type of fault and the value of the fault resistance. A Fault object is a circuit element (a resistor network) just like any other element 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 to be considered. The program automatically selects one at a time. This is useful for such analyses as examining what voltages are observed at a DG site for various faults on the utility system, computing voltage sag indices, etc.

## HARMONIC FLOW ANALYSIS

The OpenDSS 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. (There are several default spectra.) The spectra are connected to Load, Generator, voltage source (Vsource), current source (Isource) objects and a few other power conversion elements as desired. More recently, PVSystem and Storage models that contribute harmonics have been added. There are reasonable default spectra for each of 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 (solve mode=direct) is generally sufficient to initialize the harmonic solution. Harmonic sources are then initialized to appropriate magnitudes and phases angles to match the solution.

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

```
Solve mode=harmonics
```

The OpenDSS then solves for each frequency presently defined for any of the harmonic-producing circuit elements (these are all presently Power Conversion-class elements – PC Elements). Users may also specify which harmonics are to be computed. 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 in normal ABC, or 123, rotation.
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 2$^{nd}$ harmonic, and a zero-sequence relationship at the 3$^{rd}$ harmonic, etc. In between integer harmonics, the phasors will be somewhere in between (the difficulty will be deciding what that means!).

Note that the shunt element of PC Elements is included in the system Y matrix for harmonics analysis. Load objects have options for how to treat the linear equivalent impedance. The default is 50/50 split between series R-L and parallel R-L branches. This is generally a reasonable assumption if you don't have any better information. This assumption will affect the sharpness of resonances in the frequency sweep result. You have the option of changing the proportion of series and parallel equivalents. Alternatively, you can ignore the shunt branch of the Norton equivalent harmonic current source entirely by the option **Set NeglectLoadY=Yes**. This will apply the current source directly to the bus with no shunt admittance to bleed off current. Thus, if there is a sharp parallel resonance at the bus, the simulation will yield unrealistically high voltages.

## DYNAMICS

The OpenDSS can perform basic electromechanical transients, or dynamics, simulations. The capability of OpenDSS has been expanding steadily due to needs in inverter modeling and other applications where machine dynamics are important. The original intent was to provide enough modeling capability to evaluate DG interconnections for unintentional islanding studies. The built-in Generator model has a simple single-mass swing equation model that is adequate for many DG studies for common distribution system fault conditions. In addition, 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. It was initially provided as a separate DLL for the Generator model named "IndMach01a.DLL". It is provided with the program along with its source code in Delphi. It can perform both power flow and dynamics simulations using a simple symmetrical component model. The model has recently (2017) been built into the OpenDSS as the IndMach012 element. Both models have the same functionality. The DLL serves as a template for users wishing to create their own model under the Generator, which is a common choice for user-written models.

## 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 option documentation for descriptions of the Monte Carlo solution modes.

## GEOMAGNETICALLY INDUCED CURRENT (GIC) ANALYSIS

OpenDSS can perform **geomagnetically induced current** (GIC) analysis of power systems. Currently, the analysis capability is limited to three-phase systems, and cannot be integrated into other types of simulations, e.g. load flow. The GIC analysis takes advantage of the N-phase modeling capability of OpenDSS to perform the analysis on a three-phase basis as opposed to a single-phase basis used in transmission system analysis programs.

GIC are quasi-dc; thus, GIC simulations involve performing a low frequency (typically 0.1 Hz) analysis of the resulting dc network. The driving force behind the flow of GIC in the network is the induced voltage in the transmission lines[1]. The induced voltage is generated by the coupling of the transmission lines with the induced geoelectric field at the surface of the earth. Specialized quasi-dc models, which are described in other sections of the manual, have been added to the program circuit element models for transmission lines, transformers, substation ground grids, and GIC blocking devices.

[1] D.H. Boteler, R.J. Pirjola, "Modeling Geomagnetically Induced Currents Produced by Realistic and Uniform Electric Fields", *IEEE Transactions on Power Delivery*, Vol. 13, No. 4, pp. 1303-1308, October 1998.

Once the dc model of the network has been constructed the following two OpenDSS commands are used to perform the analysis:

```
Set frequency=0.1
Solve
```

The solution frequency is set to 0.1 Hz and then a standard OpenDSS snapshot solution is performed. The sources in the problem at this frequency are the voltages induced along the lines. These sources are contained in the GICLine model. Transformers are modeled by special transformer models using the GICTransformer element.

The GIC modeling is under active investigation and changes are likely before the end of 2018.

# Basic Usage

## OpenDSS Control Panel

There is a "Control Panel" with tabbed windows for constructing and testing scripts and executing them. The main access to scripts and results is generally through script files and output files or the COM interface. However, you may put large scripts into a window on the Control Panel if you wish.

Communication to the DSS is fundamentally accomplished through text strings passed to the OpenDSS command processor. However, if you are driving the program through the in-process COM or the DirectDLL library from another program, there are many functions provided in the 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 that are executed repeatedly, although the OpenDSS 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. A few standard text file reports are provided by the base OpenDSS software component (see Show and Export commands). The intent for users demanding more sophisticated reports is for users to design them through Excel worksheets, MATLAB plot, Python Plots or whatever application is being used to control the OpenDSS in special ways.

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

(Caveat: When invoked from a MS Excel application, Excel may trap some of the keystrokes, so it doesn't always work as smoothly as it does in the standalone EXE application. This may happen in other programs as well but seems to work OK in user-written software that is not trapping keystrokes.)

The control panel (Figure 5) is a typical Windows tabbed control. In Figure 5, the tabs are along the bottom of the main window. The user may execute a script, or a portion of a script, in any of the open window tabs 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 6).

Of course, the script in the windows can redirect the OpenDSS 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.

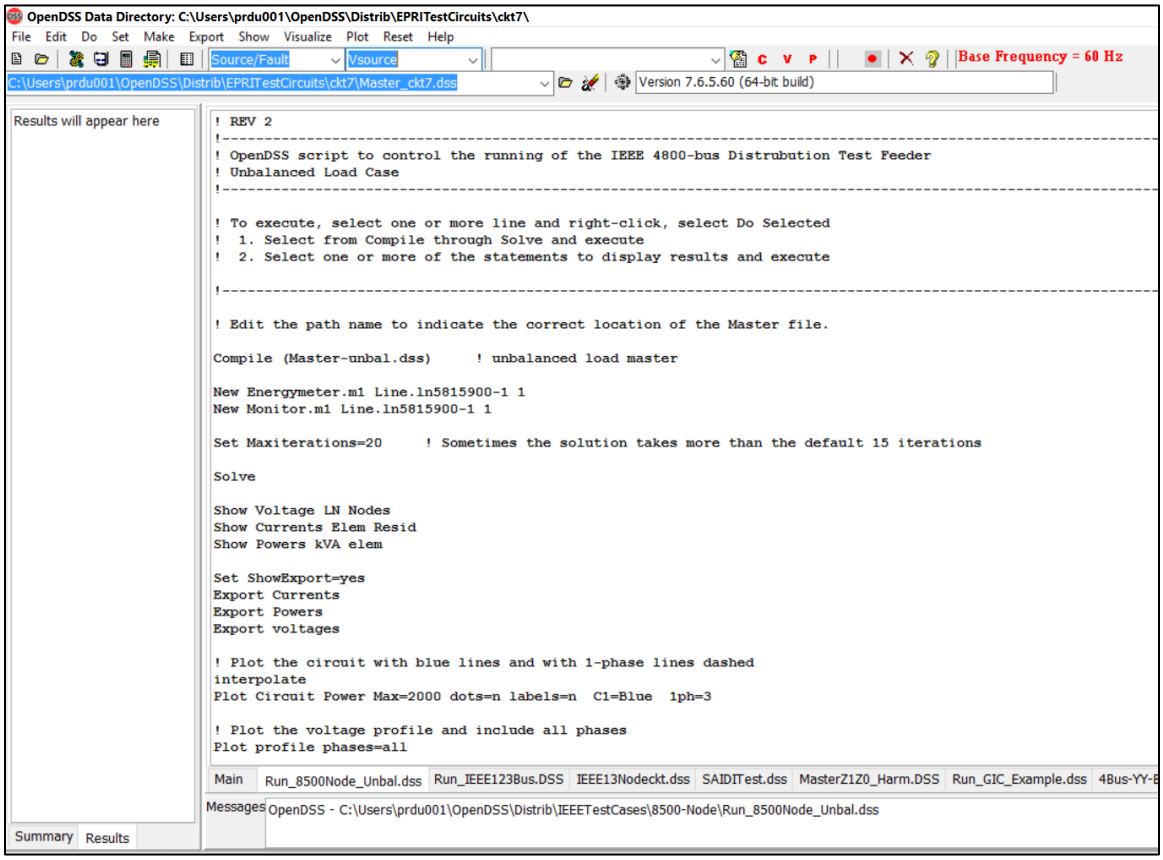



**Figure 5. Example of the Control panel window for the DSS**

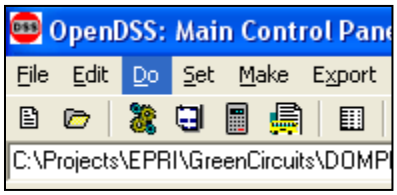



**Figure 6. "Do command" and speed buttons**

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 (such as Editplus or Notepad++) 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*

Where *filename* is the full path name of the text editor you wish to use. This should be a plain text editor. Currently, the program accepts only plain ANSI text from files. The RichEdit windows are actually Unicode but are converted to ANSI when they are saved.

Editors such as Word or Wordpad may add text that is not properly interpreted by OpenDSS. You

may reformat the text in a window while the program is running, but these formatting commands are not saved to the DSS file. Many modern editors can save in a variety of text formats. Set options for ANSI text when saving DSS files for OpenDSS.

It is generally necessary to issue the Set Editor command only one time. The program will remember this value in the Windows Registry after the program shuts down. Likewise, the default base frequency (50 or 60 Hz) is set one time. The assumed default base frequency is displayed at the top of the main screen. If you work on both types of systems (as we do), always make sure it is set to the correct value! Some users always insert th command to set the default base frequency immediately after the Clear command in the Master file.

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 7). "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.

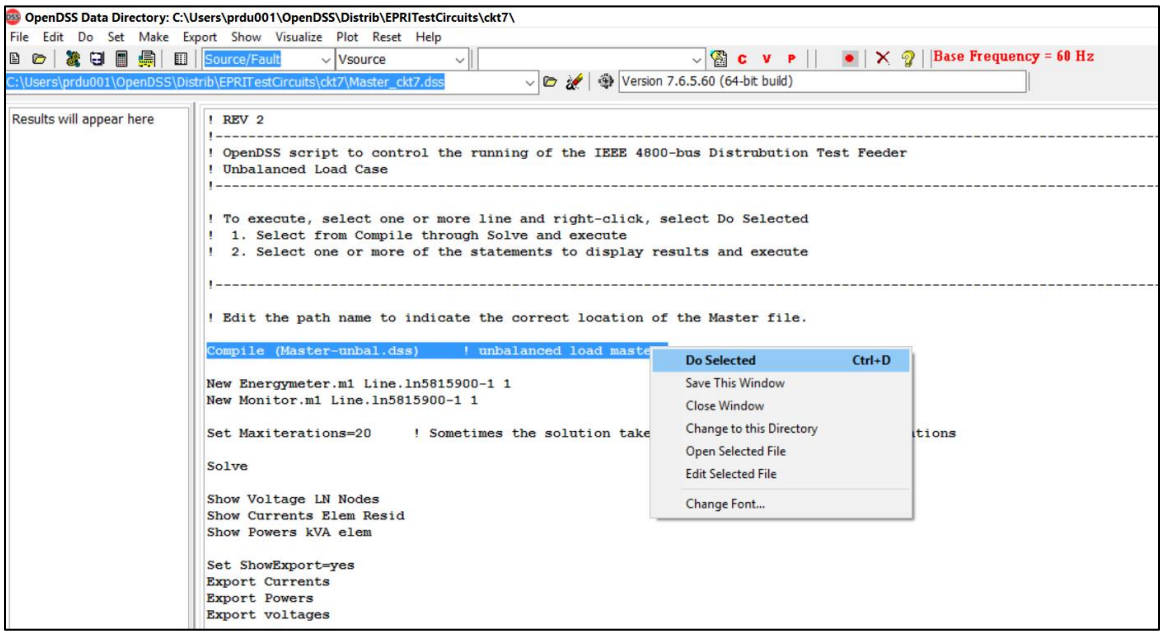



**Figure 7. Compiling a File containing a circuit description script from a script window in the control panel.**

You may change the font for a window by clicking the "Font…" button. The size of the font most recently used is remembered by the program. Font information is not kept with the script files: these are simple plain text files.

## 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 (Circuit Element) name
- Note: Voltage exports are keyed on bus name
- Residual values are also provided (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.

The export options include (check the help – there are frequent updates to the Export command):

| Export Command [Filename] | Default File Name | *Description* |
|---|---|---|
| Export Voltages | EXP_VOLTAGES.CSV | *Voltage, magnitude and angle* |
| Export SeqVoltages | EXP_SEQVOLTAGES.CSV | *Sequence voltage, magnitude only* |
| Export Currents | EXP_CURRENTS.CSV | *All currents, magnitude and angle* |
| Export Overloads | EXP_OVERLOADS.CSV % | *overload for overloaded elements* |
| Export Capacity | EXP_CAPACITY.CSV | *Capacity report on lines and transformers* |
| Export Estimation | EXP_ESTIMATION.CSV | *Results of Estimate command* |
| Export Unserved [UEonly] | EXP_UNSERVED.CSV | *Unserved energy values; requires energymeter* |
| Export SeqCurrents | EXP_SEQCURRENTS.CSV | *Sequence currents, magnitude* |
| Export Powers [MVA] | EXP_POWERS.CSV | *Powers into each terminal, kW and kvar* |
| Export P_ByPhase | EXP_P_BYPHASE.CSV | *Powers by phase* |
| Export SeqPowers [MVA] | EXP_SEQPOWERS.CSV | *Positive-, negative-, and zero-sequence powers* |
| Export Faultstudy | EXP_FAULTS.CSV | *Results of Solve Mode=Faultstudy* |
| Export Generators [Filename \| /m ] | EXP_GENMETERS.CSV | *Values of Generator Energy Meters* |
| Export Loads | EXP_LOADS.CSV | *Values for each Load* |
| Export Meters [Filename \| /m ] | EXP_METERS.CSV | *Energy meter exports. Adding the switch "/multiple" or "/m" will cause a separate file to be written for each meter.* |
| Export Monitors monitorname | file name is assigned | *Specified monitor value* |
| Export YPrims | EXP_YPRIMS.CSV | *All primitive Y matrices* |
| Export Y | EXP_Y.CSV | *System Y matrix (full, complex).* COULD BE HUGE! |
| Export YCurrents | *EXP_YCurrents.CSV* | *Exports the present solution complex Current array in same order as YNodeList. This is generally the injection current array* |

| | | |
|---|---|---|
| **Export YVoltages** | ***EXP_YVoltages.CSV*** | ***Exports the present solution complex Voltage array in same order as YNodeList.*** |
| **Export YNodeList** | ***EXP_YNodeList.CSV*** | ***Exports a list of nodes in the same order as the System Y matrix.*** |
| **Export Yprims** | ***EXP_YPRIMS.CSV*** | ***All primitive Y matrices.*** |
| **Export SeqZ** | **EXP_SEQZ.CSV** | ***Sequence Impedances at each bus*** |
| **Export Summary** | **EXP_SUMMARY.CSV** | ***Time-stamped summary of present solution. (Cumulative) One record per solution*** |
| **Export Profile** | **EXP_Profile.CSV** | ***Data to enable you to recreate a voltage profile plot in another program.*** |
| **Export AllocationFactors** | **(Assigned)** | ***Load allocation factors determined by Allocate or Estimate command or set by user.*** |
| **Export BusCoords** | **EXP_BUSCOORDS.CSV** | ***Bus X-Y coordinates in CSV form*** |
| **Export Counts** | **EXP_Counts.CSV** | ***List of counts of each class type in the active circuit*** |
| **Export NodeNames** | **EXP_NodeNames.CSV** | ***Exports Single-column file of all node names in the active circuit. Useful for making scripts.*** |
| **Export NodeOrder** | **EXP_NodeOrder.CSV** | ***Exports the present node order for all conductors of all circuit elements*** |
| **Export Result** | **EXP_Result.CSV** | ***Exports the text result of the most recent command*** |
| **Export Seqz** | **EXP_SEQZ.CSV** | ***Exports the equivalent Z1, Z0 to each bus*** |
| **Export PVSystem_Meters** | **EXP_PVMETERS.CSV** | ***Present values of PVSystem meters. Adding the switch "/multiple" or "/m" will cause a separate file to be written for each PVSystem.*** |
| **Export Storage_Meters** | **EXP_STORAGEMETERS.CSV** | ***Present values of Storage meters. Adding the switch "/multiple" or "/m" will cause a separate file to be written for each Storage device.*** |
| **Export Losses** | **EXP_LOSSES.CSV** | ***Exports losses for each circuit element*** |

For additional description see the Export Command description in the command reference. Also, see the help from running the program. There are frequent additions.

The **circuit name** or *casename* 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 OpenDSS operation that writes a file) the name of the output file appears in the Result window. You can quickly display/edit the file by one of the following means:

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

Alternatively, you may set the option "Set Showexport=Yes" and the exported file will be automatically displayed in the default text editor. Normally this option is off so that files are not popping up during simulations.[1]

## PLOT MENU COMMANDS

The Plot command has several options and can be confusing. Check the Wiki on SourceForge.net and the Help in the executable for up-to-date information. The Plot command has its own section in the Help menu.

The help text for the "Type" property of the Plot command is:

```
One of {Circuit | Monitor | Daisy | Zones | AutoAdd | General (bus data) |
Loadshape | Tshape | Priceshape |Profile}

A "Daisy" plot is a special circuit plot that places a marker at each Generator
location or at buses in the BusList property, if defined. A Zones plot shows the
meter zones (see help on Object). Autoadd shows the autoadded generators. General
plot shows quantities associated with buses using gradient colors between C1 and
C2. Values are read from a file (see Object). Loadshape plots the specified
loadshape. Examples:


Plot type=circuit quantity=power
Plot Circuit Losses 1phlinestyle=3
Plot Circuit quantity=3 object=mybranchdata.csv
Plot daisy power max=5000 dots=N Buslist=[file=MyBusList.txt]
Plot General quantity=1 object=mybusdata.csv
Plot Loadshape object=myloadshape
Plot Tshape object=mytemperatureshape
Plot Priceshape object=mypriceshape
Plot Profile
Plot Profile Phases=Primary
Plot Profile Phases=LL3ph
```

See the description of the Plot command below in this document or refer to the online information in the Help menu and on the Wiki site. Some tips on creating plots are provided here.

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

according to:

- Power
- Current
- Voltage
- Losses
- Capacity (remaining)

Circuit plots are usually unicolor plots (specified by color C1). However, Voltage plots are multicolor depicting 3 levels for voltage. Meter zone plots use a different color for each feeder.

The following example will display the circuit with the line thickness proportional to POWER relative to a max scale of 2000 kW:

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

Colors may be specified by their RGB color number or *standard color names*. The above example expresses the number in hexadecimal, which is a common form. The last 6 digits of the hexadecimal form give the intensity of the colors in this format: $00BBGGRR. In this way, any color can be produced. The standard color names are:

- Black
- Maroon
- Green
- Olive
- Navy
- Purple
- Teal
- Gray
- Silver
- Red
- Lime
- Yellow
- Blue
- Fuchsia
- Aqua
- LtGray
- DkGray
- White

The "general plot" creates displays on which the color of bus marker dots (see *Markercode=* option description later in this document) are varied according to an arbitrary quantity entered through a CSV file . The following gives a nice yellow-red heat plot variation of CSV file data field 1 (the bus name is assumed to be in the first column, or field 0, of the csv file; field quantity 1 refers to column 2 of the CSV file) with the min displayed as color C1 ($0080FFFF = yellow) and the max=0.003 being C2 ($000000FF = red). Values in between are various shades of orange.

```
plot General quantity=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.

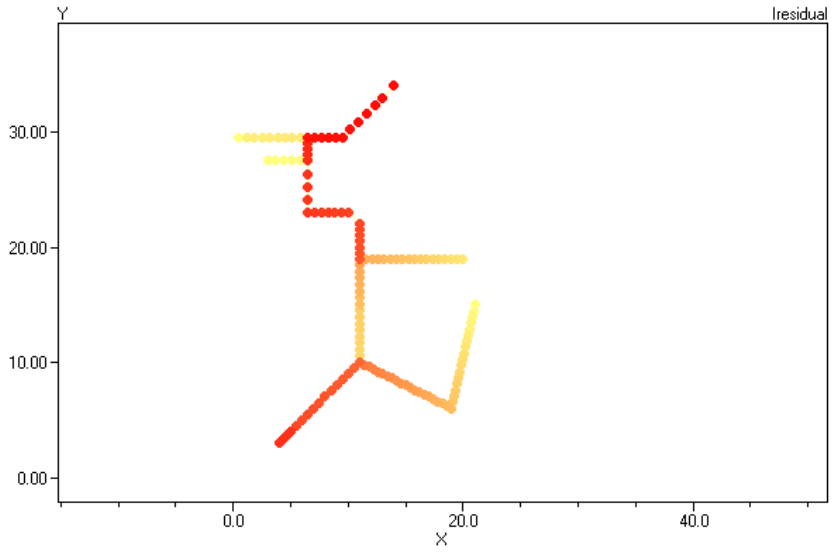



**Figure 8: Color-shaded "heat" plot of harmonic resonance**

There is also an option on the plot circuit command that will permit you to draw the circuit with LINE object widths 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.MyLineName" or simply "MyLineName". 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, to plot the first quantity after the Line object name:

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

The OpenDSS 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).

In the standalone EXE version of the program, the Plot > Circuit Plots > General Line Data menu will guide you through the creation of this command with menus and list prompts (Hint: turn the "VCR button" recorder on). The first (header) row 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.

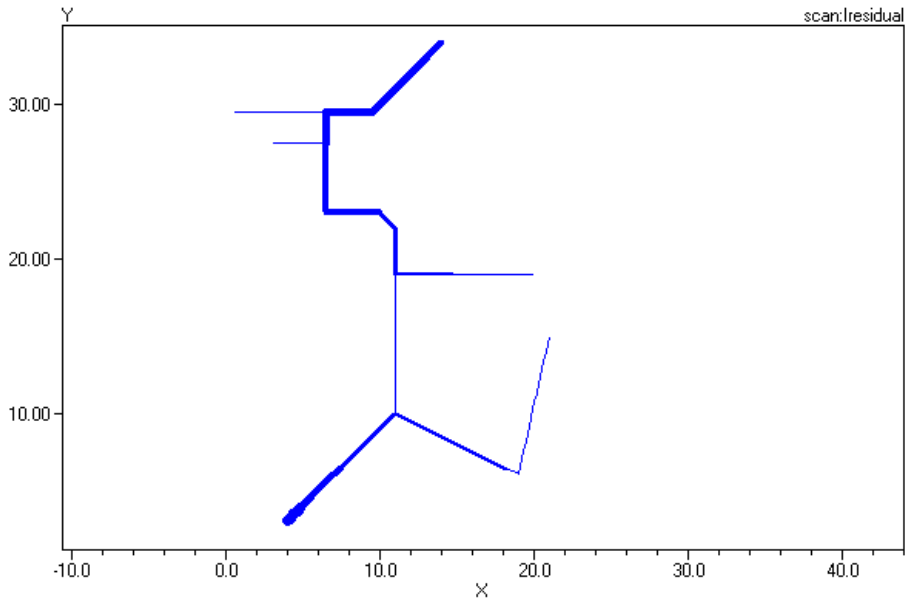



**Figure 9: Thickness-weighted plot of harmonic resonance**

Much more information is available on the Wiki site.

**Profile Plots**

Voltage profile plots are quite useful. Figure 10 shows a profile plot from the unbalanced IEEE 8500-Node Test Feeder case. This was generated by issuing the following command after the solution:

```
Plot Profile  Phases=ALL
```

The options for the Phases property for Line-Neutral voltages are:

```
Plot profile   ! default - 3-phase portion of the circuit
Plot profile phases=all
Plot profile phases=primary
```

Options for Line-Line voltages are:

```
Plot profile phases=LL3ph
Plot profile phases=LLall
Plot profile phases=LLprimary
```

Prerequisites for this command are to have an **Energymeter** at the head of the feeder and have the voltage bases defined. The program plots the per unit voltage magnitude at each node for each Line object in the Energymeter's zone. LV (secondary) Line objects are plotted as a dotted line. This puts the "beard" on the plot.

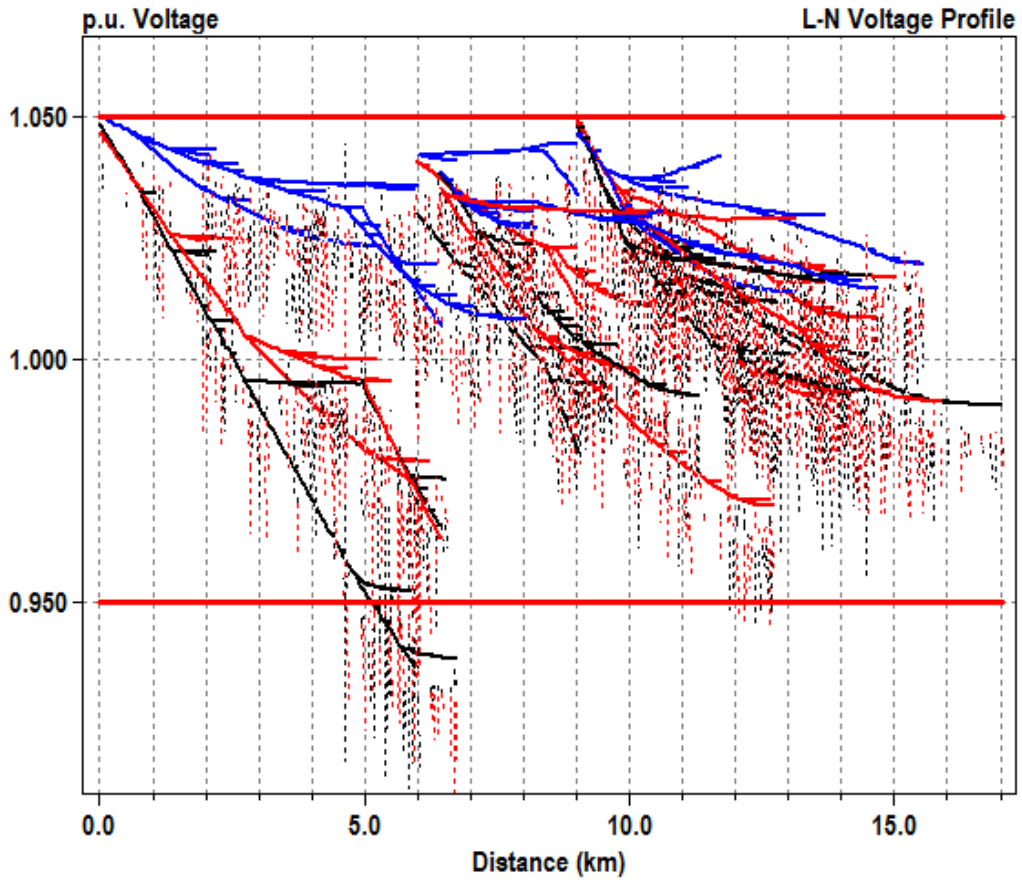



**Figure 10. Voltage Profile Plot from IEEE 8500-Node Test Feeder**

Lines that drop toward zero in the profile plot are for nodes that likely do not have a connection to the main power source (i.e., are floating). These may have no impact on the solution, but should be investigated and corrected if necessary.

To get the best profile plot, each LINE object should have the Length and Units properties specified. If the Units property is omitted and the Length property is 1.0, it will be assumed that this is a 1 km line, which usually is not correct. A distorted profile will result.

The horizontal lines on the plot coincide with the settings for the *normvminpu* and *normvmaxpu* options for the active circuit. These default to 0.95 and 1.05, respectively. To change them, execute a script like this:

```
Set normvminpu=0.90
Set normvmaxpu=1.10
```

Note: You may also export the data for this plot using the Export Profile command. The resulting CSV file gives you coordinates you may use in other graphics tools. The format is similar to the Plot command:

**Export Profile Phases=*option* [*optional file name*]**

The Phases property has the same options as the Plot command. A snippet of the CSV file exported for the command Export Profile Phases=All for the same case as above is shown here (the header row is wrapped):

```
Name, Distance1, puV1, Distance2, puV2, Color, Thickness, Linetype, Markcenter,
Centercode, NodeCode, NodeWidth,Title=L-N Voltage Profile, Distance in km
LN5815900-1, 0, 1.04836, 0.0195933, 1.04802,1, 2, 0, 0, 0, 16, 1
LN5815900-1, 0, 1.04669, 0.0195933, 1.0464,2, 2, 0, 0, 0, 16, 1
LN5815900-1, 0, 1.05016, 0.0195933, 1.05005,3, 2, 0, 0, 0, 16, 1
LN5742828-1, 0.0195933, 1.04802, 0.0448456, 1.04758,1, 2, 0, 0, 0, 16, 1
LN5742828-1, 0.0195933, 1.0464, 0.0448456, 1.04602,2, 2, 0, 0, 0, 16, 1
LN5742828-1, 0.0195933, 1.05005, 0.0448456, 1.0499,3, 2, 0, 0, 0, 16, 1
TPX226195333C0, 0.0448456, 1.04037, 0.0600856, 1.03503,1, 2, 2, 0, 0, 16, 1
TPX226195333C0, 0.0448456, 1.04473, 0.0600856, 1.04596,2, 2, 2, 0, 0, 16, 1
LN5742828-2, 0.0448456, 1.04758, 0.114118, 1.04638,1, 2, 0, 0, 0, 16, 1
LN5742828-2, 0.0448456, 1.04602, 0.114118, 1.04499,2, 2, 0, 0, 0, 16, 1
LN5742828-2, 0.0448456, 1.0499, 0.114118, 1.04951,3, 2, 0, 0, 0, 16, 1
LN5473414-1, 0.114118, 1.04638, 0.221067, 1.04454,1, 2, 0, 0, 0, 16, 1
LN5473414-1, 0.114118, 1.04499, 0.221067, 1.0434,2, 2, 0, 0, 0, 16, 1
LN5473414-1, 0.114118, 1.04951, 0.221067, 1.04891,3, 2, 0, 0, 0, 16, 1
LN5714974-1, 0.221067, 1.04454, 0.343671, 1.04243,1, 2, 0, 0, 0, 16, 1
LN5714974-1, 0.221067, 1.0434, 0.343671, 1.04159,2, 2, 0, 0, 0, 16, 1
LN5714974-1, 0.221067, 1.04891, 0.343671, 1.04823,3, 2, 0, 0, 0, 16, 1
LN5896798-1, 0.343671, 1.04243, 0.387831, 1.04167,1, 2, 0, 0, 0, 16, 1
LN5896798-1, 0.343671, 1.04159, 0.387831, 1.04094,2, 2, 0, 0, 0, 16, 1
LN5896798-1, 0.343671, 1.04823, 0.387831, 1.04799,3, 2, 0, 0, 0, 16, 1
LN5775442-1, 0.387831, 1.04167, 0.493392, 1.03986,1, 2, 0, 0, 0, 16, 1
LN5775442-1, 0.387831, 1.04094, 0.493392, 1.03937,2, 2, 0, 0, 0, 16, 1
LN5775442-1, 0.387831, 1.04799, 0.493392, 1.04745,3, 2, 0, 0, 0, 16, 1
TPX223658A0, 0.493392, 1.03242, 0.508632, 1.03085,1, 2, 2, 0, 0, 16, 1
TPX223658A0, 0.493392, 1.03424, 0.508632, 1.03402,2, 2, 2, 0, 0, 16, 1
```

# Overall Circuit Model Concept

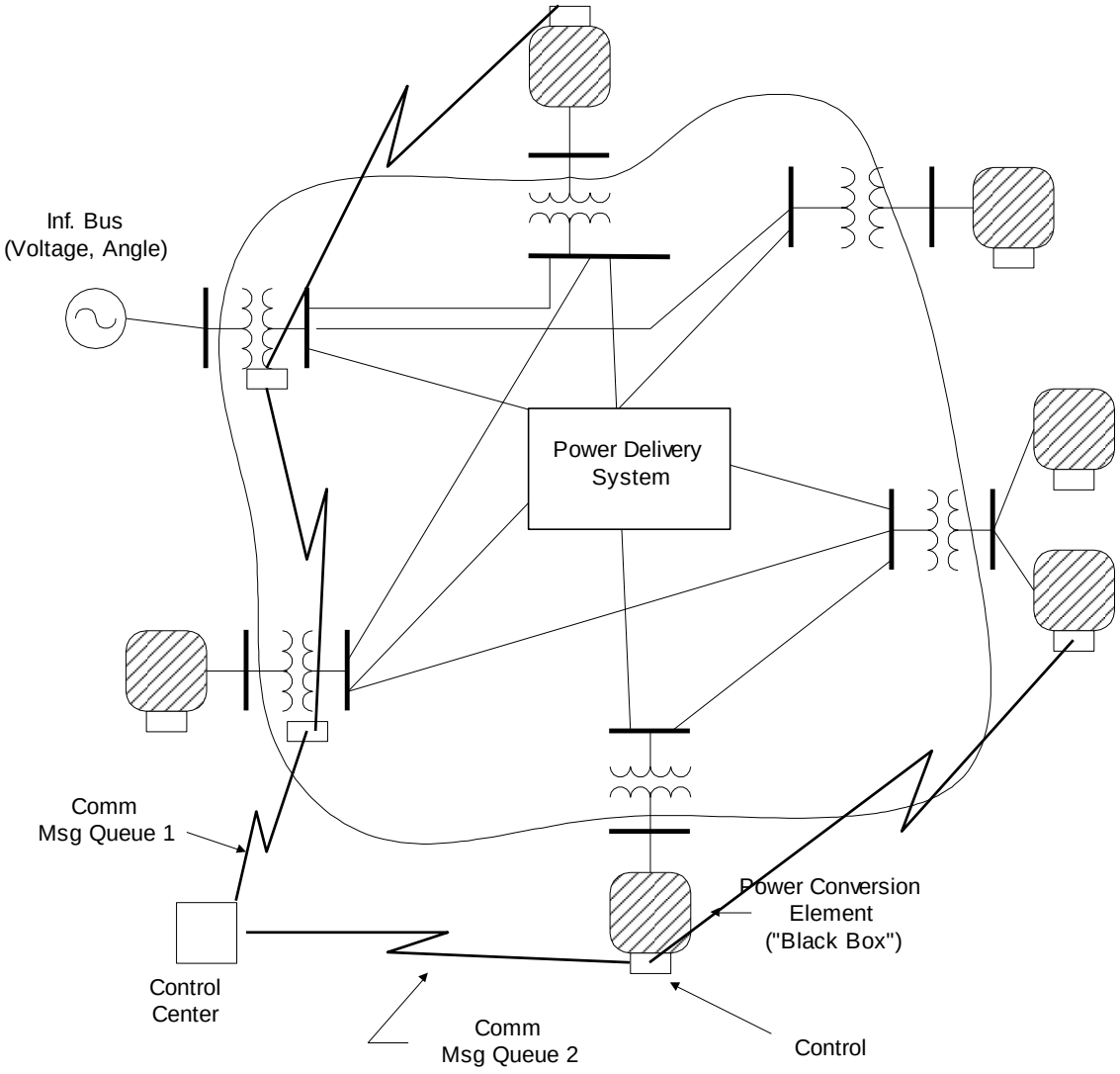



**Figure 11: Electrical Circuit with Communications Network**

The OpenDSS consists of a model of the electrical power distribution 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. Preliminary simulations of packets in a communications network have been performed using separate tools and the scripting capability of OpenDSS. This work continues.]*

# 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. In many power system analysis programs, "bus" and "node" are nearly synonymous, but they are distinctively different in OpenDSS. Bus is the container of Node objects. That is to say, *a Bus has Nodes.*

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). That basically dictates the size of the problem that must be solved, although there is also computational overhead computing the injection, or compensation, currents.

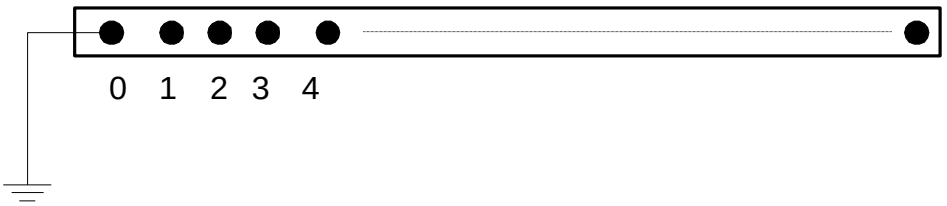



**Figure 12: Bus Definition**

Unlike many power flow programs, there are no special bus types in OpenDSS. All are the same and are simply locations to connect circuit elements together. There is no special load bus or generator bus or slack bus. Just plain buses, similar to most electromagnetic transients (EMT) programs. The function of a bus depends on what is connected to it. This is a liberating concept to the modeler. I could put 30 loads on a bus or put loads and generators on the same bus. While not all configurations one could imagine will easily converge during the power flow iterations, there is no rule that prevents the user from trying.

## TERMINAL DEFINITION

Each electrical element in the power system has one or more terminals. Each terminal has one or more conductors. The conductors are numbered [1, 2, 3,…]. Each conductor conceptually contains a disconnect switch that can be controlled by a control element (the fuse shown in the figure below has been deprecated from the model). Fuses, relays, and reclosers are modeled as control elements that monitor terminal currents and then open or close switches. Thus, it is not necessary to insert an explicit switch element, which increases the problem size.

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 virtually any other conductor but are frequently neutrals or other non-power conductors.

The OpenDSS 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 strings of arbitrary length. You may use quite long names and do not have to declare the length ahead of time.

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).

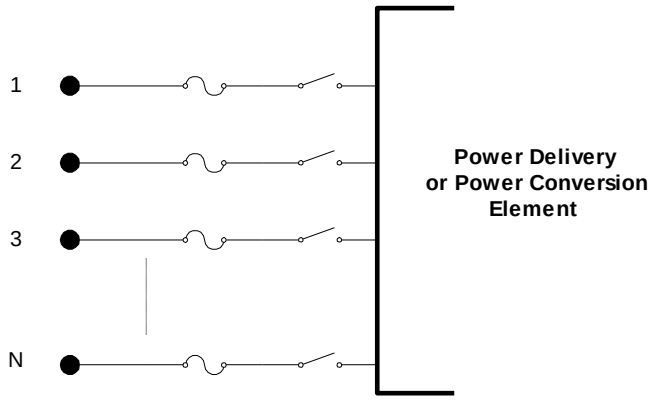



**Figure 13: Terminal Definition (Note: the fuse has been deprecated)**

## BUS NAMING

Buses are named by an alphanumeric string of characters. The names may be numbers, but are always treated as strings. Internally, buses will be numbered (actually, each node is numbered), but are referenced through the COM or command interface only by name. The internal index number is subject to change if something in the circuit changes and is not a reliable way to refer to a bus or node. See Bus Instantiation and Life below.

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, parentheses, or brackets). This can be a source of errors because of the different entry points for the names (user interface, files, COM interface). However, names from models in other programs such as PSS/E may contain blanks (may also be duplicates).

Names may be any reasonable length. In OpenDSS, strings are dynamically allocated on the memory heap. In the COM server, names are passed to the OpenDSS through the COM interface as standard null-terminated strings (actually, widestrings). This is a common way for representing strings in the Windows environment. Thus, it is a simple matter to drive the OpenDSS from most programming languages.

## BUS INSTANTIATION AND LIFE

One of the features of the OpenDSS 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 in the circuit. Then each bus is instantiated. Likewise, a bus may disappear during a simulation.

The reason for this behavior of the OpenDSS is to avoid having to define all the buses in the problem ahead of time. This allows users a great deal of flexibility in simulating distribution circuits in which the topology is changing during the simulation. If you want to add a new bus, simply define a device that is connected to the bus or edit the bus connections of an existing device. Since the OpenDSS does not need to know the voltage bases to perform its solution, there is no need to define the base voltages ahead of time. However, the voltage bases are useful for reports and certain simulations. Just remember to set the voltage bases AFTER the buses exist and before the report is generated.

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

The **CalcVoltageBases** command will automatically set the voltage bases where practical by performing a no-load power flow. The legal voltage bases for a problem are defined using the **Set Voltagebases=** [array of voltagebases in kV L-L].

Otherwise, you can use the **SetkVBase** command to set the voltage base for a selected bus. Often, users will create separate script files with all the necessary SetkVBase commands. These execute quickly but are used only when it is difficult to automatically discern between two voltage bases in a problem.

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

**Implications/Side Effects of Bus Instantiation Rules:**

- You can't define the voltage base for a bus until it exists. The CalcVoltageBases 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 automatically 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 such as voltage base or coordinates 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. However, the conductors can represent other power carrying conductors. There is no hard rule.

**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 conductor at the delta-connected terminal. 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.

**Buses have Nodes:** A bus may have any number of *nodes* (places to connect device terminal conductors). Nodes are integer numbers. The nodes may be arbitrarily numbered. However, the first N are by default reserved for the N phases of devices connected to them. 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 generally maintained automatically, although there can be exceptions.

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).

11. 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).

12. 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 linear electrical elements generally completely defined in the rms steady state by their *impedances*. Thus, they can be represented fully by the primitive **y** matrix ($Y_{prim}$).

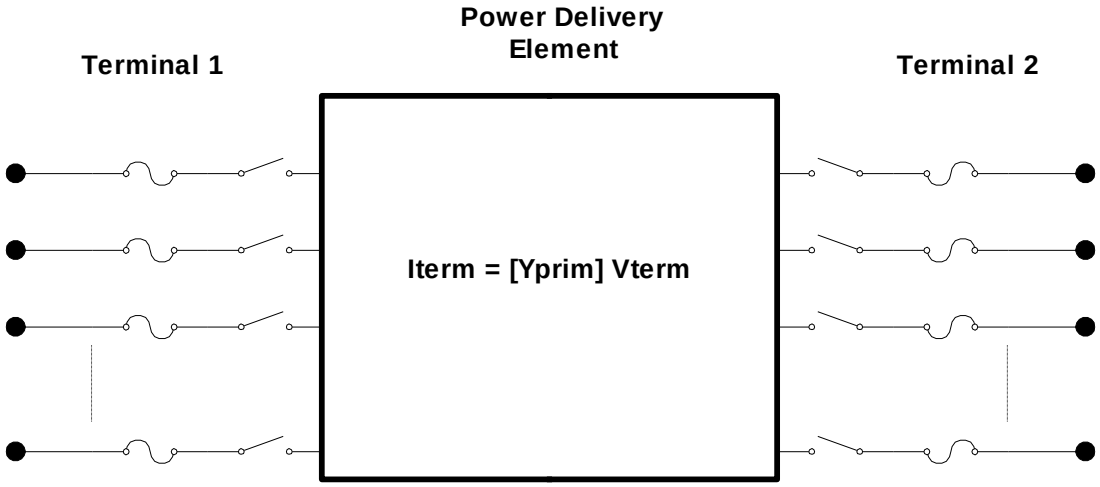



**Figure 14: Power Delivery Element Definition**

# 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 following 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.

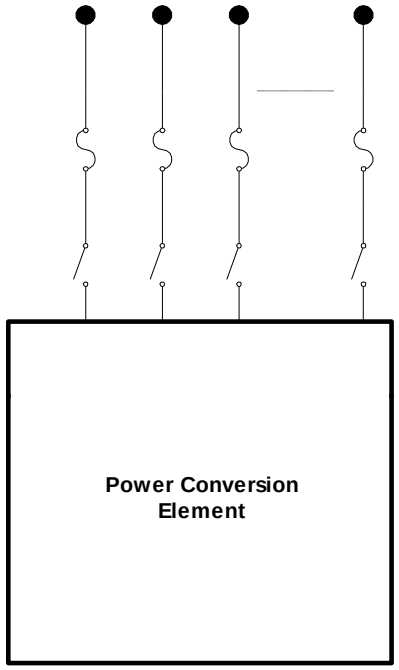



**Figure 15: Power Conversion Element Definition**

Within the OpenDSS, the typical implementation of a PC element is as shown in Figure 16. 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 OpenDSS 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.

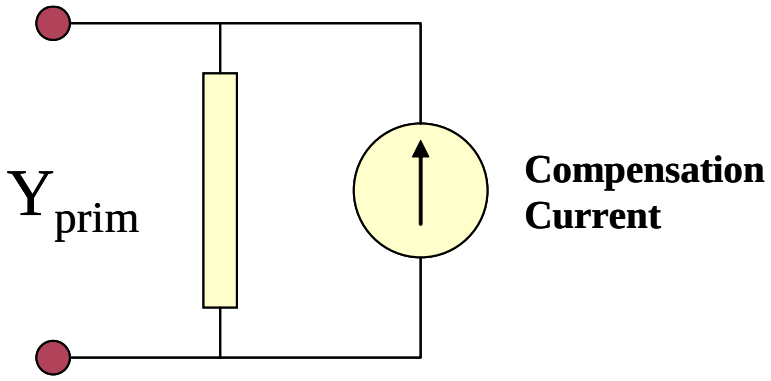



**Figure 16. Compensation Current Model of PC Elements (one-line)**

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 element models presently implemented are:

1. Constant P and constant Q load model: generically called *constant power load model.* It is the model most commonly used in power flow studies. It can suffer convergence problems when the voltage deviates too far from the normal range;
2. Constant Z (or constant impedance) load model: P and Q vary by the square of the voltage. This load model usually guarantees the convergence in any loading condition. The model is essentially linear;
3. Constant P and quadratic Q load model: the reactive power, Q, varies quadratically with the voltage (as a constant reactance) while the active power, P, is independent from the voltage, somewhat like a motor;
4. Exponential load model: the voltage dependency of P and Q is defined by exponential parameters (see CVRwatts and CVRvars). This model is typically used for Conservation Voltage Reduction (CVR) studies. It is also used for general distribution feeder load mix models when the exact behavior of loads is not known;
5. Constant I (or constant current magnitude) load model: P and Q vary linearly with the voltage magnitude while the load current magnitude remains constant. This is a common in distribution system analysis programs;
6. Constant P and fixed Q[2] (Q is a fixed value independent of time and voltage);
7. Constant P and quadratic Q: Q varies with square of the voltage
8. ZIP load model: P and Q are described as a mixture of constant power, constant current

[2] A power (P and/or Q) is defined as “constant” when can be modified by loadshape multipliers while it is defined as"fixed" when is always the same -- at nominal, or base, value.

and constant impedance load models whose contribution is defined by coefficients (see ZIPV).

Loads can be exempt from loadshape mulitipliers.All load models revert to constant impedance constant Z load model outside the normal voltage range (that can defined by the user, see Vminpu and Vmaxpu) in an attempt to guarantee convergence even when the voltage drops very low. This is important for performing annual simulations.

# Putting it All Together

Figure 17 illustrates how the DSS puts all the PD elements and PC elements together to perform a solution.

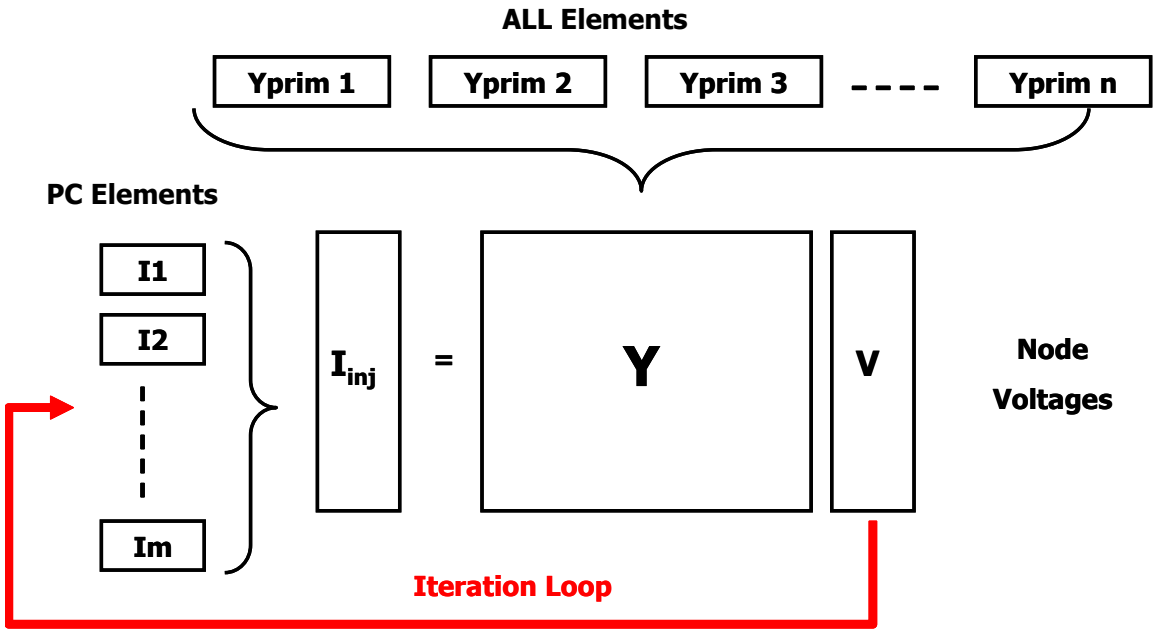



**Figure 17. OpenDSS Solution Loop**

OpenDSS uses a fairly standard Nodal Admittance formulation of the circuit model that can be found documented in many basic power system analysis texts. Most electrical engineers have encounted nodal admittance formulations very early in their education. OpenDSS provides algorithms that build the nodal admittance matrices for common power distribution system elements from the data commonly available to power distribution engineers.

A *primitive admittance* matrix, $Y_{prim}$, is computed for each circuit element in the model. These small nodal admittance matrices are used to construct the main *system* admittance matrix, $\boldsymbol{Y}_{system}$, that knits the circuit model together. Basically, the upper structure of the OpenDSS (the part that is written in Delphi) manages the creation and modification of the $Y_{prim}$ matrices 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 $Y_{prim}$ matrices are fed to the sparse matrix solver, which constructs the system **Y** matrix.

An initial guess at the voltages, ***V***, is obtained by performing a direct solution of ***I=YV***. Loads and generators are modeled by their linear equivalents with no injection currents. This gets all the phase angles and voltage magnitudes in the proper relationship. This is somewhat analogous to a "flat start" in other power flow algorithms except that it takes into account all the connections of the multi-phase, multi-voltage level system. The resulting voltages are often quite close to the final converged solution including the nonlinear elements. This is important because the OpenDSS is designed to solve arbitrary n-phase networks in which there can be all sorts of transformer ratios and connections and it must have a good initial guess at the voltages.

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 $\boldsymbol{I}_{inj}$ vector in the above figure. 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 solution is mainly focused on solving the nonlinear system admittance equation of the form:

$$\boldsymbol{I}_{inj}(V) = \boldsymbol{Y}_{system}\boldsymbol{V}$$

where,

$\boldsymbol{I}_{inj}(V)$ = *compensation, or injection, currents from Power Conversion (PC) elements in the circuit, which may be nonlinear elements*

The currents injected into the circuit from the PC elements, $\boldsymbol{I}_{inj}(\boldsymbol{V})$, are a function of voltage as indicated and basically represent the nonlinear portion of the currents from Load, Generator, PVsystem, and Storage elements in the circuit. Each PC element is queried in each iteration to provide its updated injection currents based on the present guess at the voltages. This process has the advantage of allowing considerable freedom in expressing the nonlinear behavior of the PC element. Thus, there are many load models and generator models.

There are a number of ways this set of nonlinear equations could be solved. The most popular way in OpenDSS is a simple *fixed point* method that can be written concisely:

$$\boldsymbol{V}_{n+1} = [\boldsymbol{Y}_{system}]^{-1}\boldsymbol{I}_{inj}(\boldsymbol{V}_n) \quad n = 0, 1, 2, \ldots \text{ until converged}$$

In words, after building $\boldsymbol{Y}_{system,}$ start with a guess at the system voltage vector, $\boldsymbol{V}_0$, and compute the compensation currents from each PC element to populate the $\boldsymbol{I}_{inj}$ vector. Using a sparse matrix solver, compute the new estimate of $\boldsymbol{V}_{n+1}$. Repeat this process until a convergence criterion is met.

The system $\boldsymbol{Y}$ matrix is typically not rebuilt during this process so the iterations go quickly. Refactorization of $\boldsymbol{Y}$ is not necessary as long as the elements are close to the actual value. This may result in a few extra iterations to reach a converged solution, but such iterations are computationally cheap in comparison to refactoring the large $\boldsymbol{Y}$ matrix.

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. In fact, it has been found that the first guess at the next time step when the time step size is small is often good enough. The *MinIterations* option has been added to allow the user to override the default value of 2 iterations and the minimum number of iterations may be set to 1 for simulations such as Quasi-Static Time Series (QSTS) simulations at 1-s intervals. The program determines if the

solution is good enough by comparing to the previous solution. Additional iterations will be executed automatically, if needed, to converge better. This can save almost half the computational effort in very long simulations at a small time step.

Control iterations are performed outside this loop. That is, a converged solution is achieved before checking to see if control actions are required. Then the control actions are executed without advancing time until there are no more control actions queued up. The time delay specified for each control dictates which controls operate.

# Load Models that Nearly Always Converge

One problem with nearly all power flow solution algorithms is that it is difficult to maintain a converged solution over a wide range of voltages. OpenDSS is no exception to this, but the load models have been modified so that they nearly always converge for very low voltages. The modification is illustrated in Figure 18.

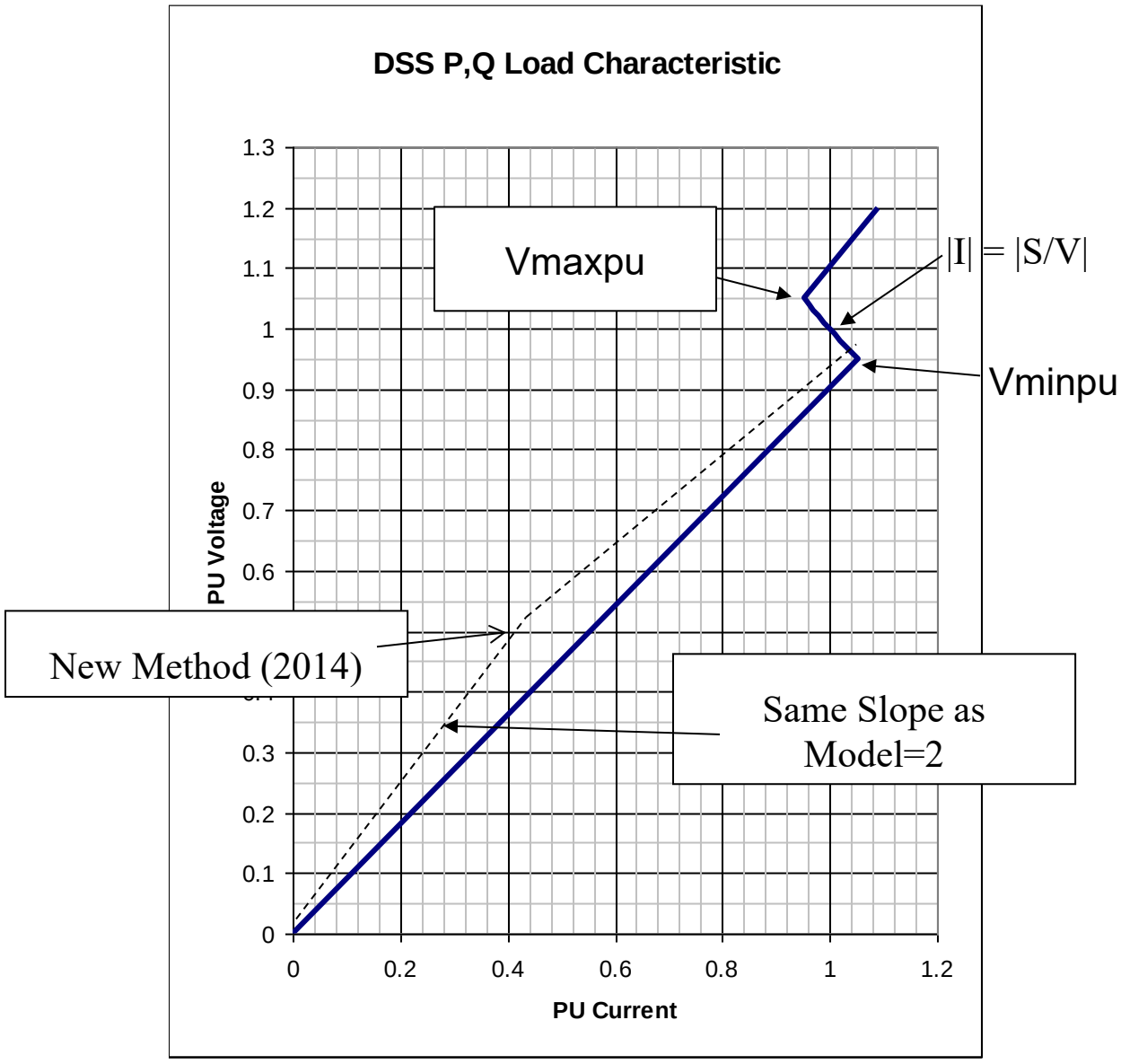



**Figure 18. Load Model Modification to Maintain Power Flow Convergence.**

This is illustrated for the common constant power load model (OpenDSS Model=1, the default). The problem is the hyperbolic characteristic between Vmaxpu and Vminpu voltages. As the voltage continues to drop, the current increases significantly if it is assumed that S remains

constant. When the voltage drops to about 70%, or below, the iterative solution often fails. Some analysts assume that when the power flow fails to converge, that will also happen on the actual power system. This is more true of the transmission grid than the distribution system. The voltage collapsing on the distribution feeder generally does not cause voltage collapse on the bulk power supply.

The voltage will sag low, but the system will generally continue to function in some regard and will recover to normal voltage when the fault causing the sag is removed. The actual voltage can be anything between 0 and normal voltage and EPRI has thousands of power quality voltage sag measurements to prove it.

Also, for multiyear planning studies the voltage will sag low as the load grows year after year. Users do not want a 20-year 8760-hour yearly simulation to bomb out 12 years into the simulation. Such simulations can take hours and you want to avoid having to repeat a simulation simply because the voltage went low. It is sufficient to know that the voltage is low in some hours and planners will have to fix something.

The OpenDSS solution is to revert to a linear model when the voltage drops below the value specified by the VminPU property of the Load model. The load characteristic shifts to a straight line from the VminPU point to zero. This proved adequate for several years, but is not a perfect linear impedance model that would always converge.

It was known that loads modeled as Model=2 (constant impedance) always reached a converged solution. As shown in Figure 18, a further modification was made in 2014 to transition the slope of the linearized model to exactly what a Model=2 load would have at some lower voltage point. A new Load model property was introduced, named *VlowPU*, to define the breakpoint for this transition. This value defaults to 50% and should not be set much different than this.

With the modified linear model, you can actually place a Fault object on the circuit while performing a power flow and it will converge most of the time. Will the answer be accurate? That all depends on your model. It is probably not very accurate if there were meters on the system to show the actual value, but simply indicating a low voltage is sufficient knowledge for planners. The key is that the program will generally not break and your simulation can proceed.

# 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 (… ), or..
- brackets [..], or
- braces {..}.

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.

Note: Be careful of using parantheses on Windows for file names containing full path names because Windows uses the string (x86) for 32-bit programs in the default Program Files folders. Use some other delimiter. Double quotes work fine and are what Windows returns for a full path name when there are blanks in the name. Hint: Right-click on a file name in Windows explorer and select Copy as Path. You will get the full pathname in double quotes that you can use in OpenDSS anywhere a file name is expected.

## 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, etc..

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 (numerical) order. The properties of elements are by default processed in positional order unless the "=" appears in the field.

Some commands are interpreted at more than one lexical level inside the OpenDSS. 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 property definitions it receives via the parameter list. 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 module affected.

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

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

For circuit elements, the next one, or two, parameters are normally the bus connection properties, 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.

## PROPERTIES

The parameters of circuit element editing commands are referred to as "properties". Properties generally set values of some data field in the targeted object, but may also have some side effects. Properties behave like properties in object-oriented programming languages. They may perform

an action as well as setting a value.

For example, setting the **PF** property of a Load object also causes the **kvar** property to be updated.

Many objects have multiple properties that essentially set the same internal data value. For example, you can set the kVA rating by either the **kVA** or **MVA** property is some elements.

This is a different approach than programs using databases typically take where the values are static and the data fields generally fixed.

## DELIMITERS AND OTHER SPECIAL CHARACTERS

The DSS recognizes these delimiters and other special characters:

**Array or string delimiter pairs:**

- [ ]
- { }
- ( )
- " "
- ' '

**Matrix row delimiter:** | (vertical bar)

**Value delimiters:**

- Comma(,)
- any white space (tab or space)

**Class, Object, Bus, or Node delimiter**: period(.)

**Keyword / value separator**: =

**Continuation of previous line:** ~ (actually a synonym for the More command)

**Comment line:** // or !

**In-line comment:** !

**Query a property**: ?

## ARRAY PROPERTIES [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]
```

### *Standard Ways to Define Array Properties*

Arrays can be defined by the following methods:

Entering the numeric values directly

```
mult=[1, 2, 3, 4, ...]
```

Entering the numbers from a single column text file:

```
mult=[File=MyTextDataFile.CSV]
```

Entering the data from a packed binary file of doubles:

```
mult=[dblFile=MyFileOfDoubles.dbl]
```

Entering the data from a packed binary file of singles:

```
mult=[sngFile=MyFileOfDoubles.sng]
```

### *Enhanced Syntax*

The 'File=...' capability for text files has been enhanced to allow you to extract a column of data from a multi-column CSV file (like you might get from an Excel spreadsheet or the OpenDSS Export command). The complete syntax is:

```
mult=[File=myMultiColumnFile.CSV, Column=n, Header=Yes/No]
```

Enter *n* for the column number you want. The default is always set to 1 for each array.

Specifying *Header=Yes* causes the first record in the CSV file to be skipped. This allows for a single line of non-numeric data in the first line, as is common in OpenDSS Export files. Default is "*Header=No*".

Example

```
New Loadshape.Ramp2 npts=4000 sInterval=1
~ mult= (file=MultiChannelTest.csv, column=3, header=yes)
```

### *Special Reserved File Name*

The name **%result%** is now used to designate the last result file. For example, if you want to export a file (such as a monitor file) and then immediately read in column 5 to do something with, you could do it with the following syntax:

```
mult=[File=%result%, Column=5, Header=Yes]
```

This must be executed before another command is issued that writes a result file.

## MATRIX PROPERTIES

Matrix parameters are entered by extending the Array syntax: Simply place a vertical 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. There have been various hashing algorithms implemented and it is difficult to keep the documentation up with the present release. Some algorithms would hash only the first 8 characters. 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 actually proceed more efficiently if the names of commands and properties 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 correctly 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 line. 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 by the New command or selected by some other command. 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.

Changes made "on the fly" are NOT saved back to the original script files. If you wish to capture the present state of the circuit for future analysis, execute the Save Circuit command, which saves

the present circuit to a separate folder. It will not overwrite an existing folder. Some manual fixup may be required to get the saved circuit to compile properly.

## 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 OpenDSS has a "stack" of 10 registers just like some calculators. 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 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 $2\pi$. The result is then multiplied by 60 to yield $\omega$ ($2\pi f$). 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 . . .
```

Recommendation: Always used the full element name (classname.elementname) to avoid confusion at a later date. It is better to make the text human readable than exploit shortcuts the program allows.

# COMMAND REFERENCE

The following text documents selected command definitions as of this writing. There are 107 commands 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.

## *// (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**

## */* … */ Block Comments*

New at version 7.6, you can now comment out whole sections (whole lines of script) using the block comment capability. The block comment must begin with **/*** in the **FIRST column** of the line. The block comment terminates after the appearance of ***/** anywhere in a line or with the end of a script file or selection in a script window. Example:

```
Compile "C:\Users\prdu001\DSSData\CDR\Master3.DSS"

New Monitor.Line1-PQ  Line.LINE1 1  mode=1 ppolar=no
New Monitor.Line1-VI  Line.LINE1 1  mode=0 VIpolar=Yes
/*   comment out the next two monitors
New Monitor.Source-PQ  Vsource.source 1  mode=1 ppolar=no
New Monitor.source-VI  Vsource.source 1  mode=0 VIpolar=Yes
****/  End of block comment

New Monitor.Tran2-VI Transformer.PHAB  2 mod=0 VIPolar=no
New Monitor.Tran3-VI Transformer.PHAB  3 mod=0  VIPolar=no


Solve
```

## *Cleanup*

Force execution of the end-of-time-step cleanup functions that samples/saves meters and updates selected state variables such as storage level. See FinishTimeStep command.

## *Connect/Disconnect*

Request to create/terminate a TCP/IP socket to communicate data with external modules. This function requires the host address and TCP port to connect.

## *Customizing Solution Processes*

The next seven commands, all beginning with an underscore ('_') character, allow you to script your own solution process by providing access to the different steps of the solution process.

**_DoControlActions**

For step control of solution process: Pops control actions off the control queue according to the present control mode rules. Dispatches contol actions to proper control element

"DoPendingAction" handlers.

**_InitSnap**
For step control of solution process: Intialize iteration counters, etc. that normally occurs at the start of a snapshot solution process.

**_SampleControls**
For step control of solution process: Sample the control elements, which push control action requests onto the control queue.

**_ShowControlQueue**
For step control of solution process: Show the present control queue contents.

**_SolveDirect**
For step control of solution process: Invoke direct solution function in DSS. Non-iterative solution of Y matrix and active sources only.

**_SolveNoControl**
For step control of solution process: Solves the circuit in present state but does not check for control actions.

**_SolvePFlow**
For step control of solution process: Invoke iterative power flow solution function of DSS directly.

## *? [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.

## *About*

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

## *AddBusMarker*

Add a marker to a bus in a circuit plot. Markers must be added before issuing the Plot command. Effect is persistent until circuit is cleared or ClearBusMarkers command is issued. Example:

```
ClearBusMarkers    !...Clears any previous bus markers
AddBusMarker Bus=Mybusname1 code=5 color=Red size=3
AddBusMarker Bus=Mybusname2 code=5 color=Red size=3
...
```

You can use any of the standard color names or RGB numbers. See Help on C1 property in Plot

command.

To clear the present definitions of bus markers, issue the ClearBusMarkers command. If you specify a busname that doesn't exist, it is simply ignored. See Set Markercode= description below to see a list of marker codes presently implemented.

### *AggregateProfiles*

Aggregates the load shapes in the model using the number of zones given in the argument. Use this command when the number of load shapes is considerably big, this algorithm will simplify the amount of load shapes in order to make the memory consumption lower for the model. The output of this algorithm is a script describing the new load shapes and their application into loads across the model. The argument on this command can be: Actual/pu to define the units in which the load profiles are. Check here for details.

### *AlignFile*

Aligns the commands and properties of DSS script files into even columns for easier reading. Creates a new file with the string "Aligned_" prepended and opens the file in the default editor. You may use the created file in place of the original if desired.

```
Alignfile [file=]filename.
```

### *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.

### *AllPCEatBus*

Returns an array with the names of all PCE (Power Conversion Element- Loads, capacitors and reactors shunt connected, DER…) connected to the active bus.

### *AllPDEatBus*

Returns an array with the names of all PDE (Power Delivery Element- Line, transformer, capacitors and reactors series connected…) connected to the active bus.

### *BatchEdit*

Batch edit objects in the same class. Useful for wholesale changes to objects of the same class. Example:

```
BatchEdit Load..* duty=duty_shape
```

In place of the object name, supply a PERL regular expression. ".*" matches all names. The subsequent parameter string is applied to each object selected. This example is equivalent to creating a script file containing the following commands, for example:

```
Load.Load1.duty=dutyshape
```

```
Load.Load2.duty=dutyshape
Load.Load3.duty=dutyshape
Etc.
```

### *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.

### *BusCoords*

Define x,y coordinates for buses. Execute after Solve or MakeBusList command is executed so that bus lists are defined.Reads coordinates from a CSV file with records of the form:

```
busname, x, y.
```

You may use spaces and tabs as well as commas for value separators.

Example:

```
BusCoords [file=]xxxx.csv
```

See also **LatLongCoords**.

### *CalcIncMatrix*

Calculates the incidence matrix of the Active Circuit. This matrix is stored in compressed format in memory. The user can export the matrix using the COM/DLL interface or by using the *Export* sommand.

### *CalcIncMatrix_O*

Calculates the incidence matrix of the Active Circuit. In this case the matrix will be calculated considering its hierarchical order, forcing the buse list considering their distribution from the substation to the last load in a radial configuration. This matrix is stored in compressed format in memory. The user can export the matrix using the COM/DLL interface or by using the *Export* sommand.

### *CalcLaplacian*

Calculate the laplacian matrix using the incidence matrix previously calculated, this means that before calling this command the incidence matrix needs to be calculated using calcincmatrix/calcincmatrix_o. This matrix is stored in compressed format in memory. The user can export the matrix using the COM/DLL interface or by using the *Export* sommand.

### *CalcVoltageBases*

Estimates the voltage base for each bus based on the array of voltage bases defined with a "**Set Voltagebases=...**" command. Performs a zero-current power flow considering only the series power-delivery elements of the system. No loads, generators, or other shunt elements are included in the solution. The voltage base for each 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. Note that the OpenDSS does not need the voltage base for most calculations, but uses it for reporting. Exceptions include processes like the AutoAdd solution mode where the program needs to specify the voltage rating of capacitors and generators it will automatically add 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 help confirm that everything in the circuit is connected as it should be. It is especially useful for devices with unusual connections. Supplement this check with a Faultstudy mode solution to verify that the system impedances are also properly specified.

### *Capacity*

Find the maximum load the active circuit can serve in the PRESENT YEAR. Gradually increases the load until something in the circuit is overloaded. Uses the EnergyMeter objects with the registers set with the SET UEREGS= (..) command for the AutoAdd functions. Syntax (defaults shown):

```
capacity [start=]0.9 [increment=]0.005
```

Returns the metered kW (load + losses - generation) and the per unit load multiplier for the loading level at which something in the system reports an overload or undervoltage. If no violations, then it returns the metered kW for peak load for the year (1.0 multiplier). Aborts and returns 0 if no energymeters.

### *CD Directoryname*

Changes the current directory to the specified directory. The current directory is also changed if a full path name is supplied for the Compile command.

### *CktLosses*

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

### *Classes*

Returns comma-separated list in Result variable with the tist of intrinsic DSS Classes.

### *Cleanup*

Force execution of the end-of-time-step cleanup functions that samples/saves meters and updates selected state variables such as storage level.

### *Clear*

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

### *ClearAll*

Clears all the circuits and all the actors, after this instruction there will be only 1 actor (actor 1) and will be the active actor.

### *ClearBusMarkers*

Clear all bus markers created with the AddBusMarker command.

### *Clone*
Clones the active circuit. This command creates as many copies of the active cirucit as indicated in the argument if the number of requested clones does not overpass the number of local CPUs. The form of this command is *clone X*, where X is the number of clones to create.

### *Close [Object] [Term] [Cond]*
Opposite of Open command.

### *CloseDI*
Close all Demand Interval (DI) files. This must be issued at the end of the final yearly solution in a Yearly or Daily mode run where DI files are left open while changes are made to the circuit. 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. Otherwise computer system I/O errors may result.

### *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
```

### *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) or the text form of the EXE version. The commands are similar except that Redirect returns to the directory from which it was invoked while Compile reset the current directory to that of the file being compiled. In DSS convention, use Compile when defining a new circuit and Redirect within script files to signify that the input 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 of the COM interface. Do not wrap 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.

### *Connect*
Request to create a TCP/IP socket to communicate data with external modules. This function requires the host address and TCP port to connect.

### *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.

### *CvrtLoadShapes*
Convert all Loadshapes presently loaded into either files of single or files of double. Usually files

of singles are adequate precision for loadshapes. Syntax:

```
cvrtloadshapes type=sng  (this is the default)
cvrtloadshapes type=dbl
```

A DSS script for loading the loadshapes from the created files is produced and displayed in the default editor.

## *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
```

## *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.)

## *Dissconnect*

Request to terminate a TCP/IP socket. This function requires the host address and TCP port to disconnect.

## *Distribute*

```
kw=nn how={Proportional | Uniform |Random | Skip} skip=nn PF=nn
file=filename MW=nn
```

Distributes generators over the system in the manner specified by "how". This command is useful for studying distributed rooftop solar PV for example. A file is saved that may be edited, if necessary, and used in subsequent simulations.

**kW** = total generation to be distributed (default=1000)

**how**= process name as indicated (default=proportional to load)

**skip** = no. of buses to skip for "How=Skip" (default=1)

**PF** = power factor for new generators (default=1.0)

**file** = name of file to save (default=distgenerators.txt)

**MW** = alternate way to specify kW (default = 1)

Example

```
Distribute kW=2000 how=proportional PF=1 file=MyPVGenfile.DSS
```

### *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.

### *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. Leave "debug" off to just see property values.

### *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.

### *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.)

### *Estimate*
Execute state estimator on present circuit given present sensor values. See also AllocateLoads.

### *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 MATLAB (use csvread), 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. (This list is incomplete. Check Help for the Export commands on line. When in doubt, execute the export and observe the result.)

**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. Note: This file can be HUGE!

**Export SeqZ [Filename] (EXP_SEQZ.CSV) .** Exports the equivalent sequence short circuit impedances at each bus. Should be preceded by a successful "Solve Mode=Faultstudy" command. This will initialize the short circuit impedance matrices at each bus.

### *ExportOverloads*
Exports the overloads report with the content avaiable at the moment of the call. It only affects the overloads report for the active actor. For executing this command, it is necessary to have an Energy meter within the circuit and the overloads report enabled (set overloadreport=True). This command overrides the CloseDI command and allows to extract the overloads report at anytime during the simulation (if the simulation is not running).

### *ExportVViolations*
Exports the voltage violations report with the content avaiable at the moment of the call. It only affects the voltage violations report for the active actor. For executing this command, it is necessary to have an Energy meter within the circuit and the volt exceptions report enabled (set Voltexpectionreport=True). This command overrides the CloseDI command and allows to extract the overloads report at anytime during the simulation (if the simulation is not running).

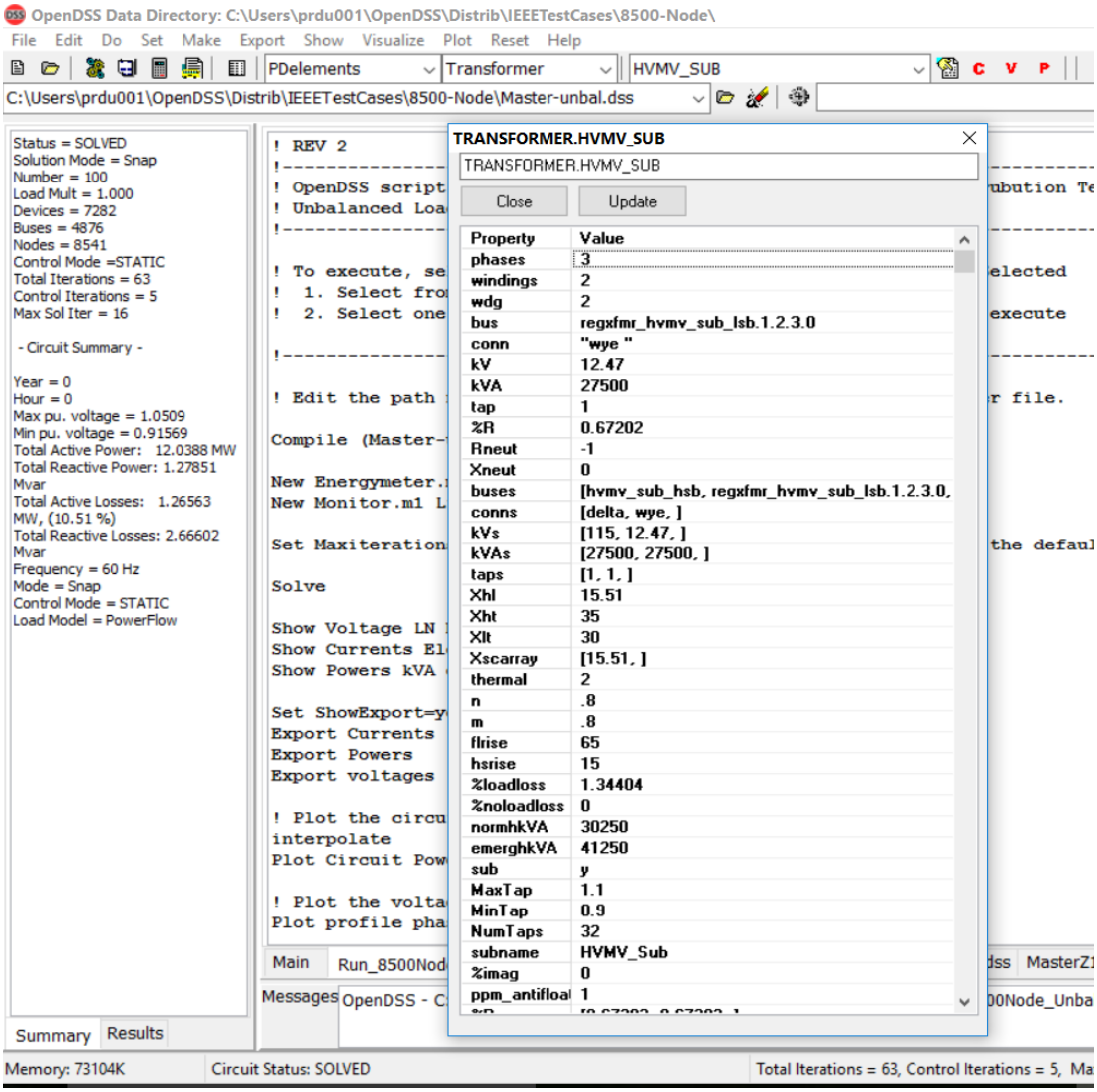
### *Fileedit [filename]*
Edit specified file in default text file editor (see Set Editor= option).

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

### *FinishTimeStep*
Do Cleanup, sample monitors, and increment time. See Cleanup command.

### *FNCSPublish*
Read FNCS publication topics from a JSON file.

### *Formedit [Class.Object]*
Brings up form editor to display the present property of the specified active DSS object. You may change the values of properties on this form for the present solution. However, the changes are NOT automatically saved to the original script files. You must execute a Save Circuit to capture the changes. This command may be executed from the speed button to the right of the two drop-down boxes at the top of the control panel. (to the right of the HVMV_SUB box in the figure below). The result is shown in the figure.




## *Get [Opt1] [opt2] etc.*

Basically, the opposite of the SET command. Returns DSS property values for options 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 Set command description.

## *Guids*

Read GUIDS (Globally-unique identifiers) for class names. Tab or comma-delimited file with full

object name and GUID.

### *GISCoords*
Define x, y coordinates for buses using real GIS Latitude and Longitude values (decimal numbers). Similar to BusCoords command. Execute after Solve command or MakeBusList command is executed so that bus lists are defined. Reads coordinates from a CSV file with records of the form: busname, Latitude, Longitude.

```
Example: GISCoords xxxx.csv
```

For using only if OpenDSS-GIS is locally installed.

### *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.

### *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.

### *Interpolate {All | MeterName}*
Default is "All". Interpolates coordinates for missing bus coordinates in meter zones. Only affects buses on line sections contained in a **Energymeter** zone. Fills in evenly spaced coordinates between existing coordinates.

### *LatLongCoords*
Define x,y coordinates for buses using Latitude and Longitude values (decimal numbers). Similar to **BusCoords** command. Execute after Solve command or MakeBusList command is executed so that bus lists are defined. Reads coordinates from a CSV file with records of the form:

```
busname, Latitude, Longitude.
```

Example:

```
LatLongCoords [file=]xxxx.csv
```

Note: Longitude is mapped to x coordinate and Latitude is mapped to y coordinate.

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

### *M*
Continuation of editing on the active object. An abbreviation for More.

### *MakeBusList*

Updates the buslist, if needed, using the currently enabled circuit elements. (This happens automatically for Solve command.) See **ReprocessBuses**

### *MakePosSeq*

Attempts to convert the present 3-phase, or multi-phase, circuit model to a single-phase positive-sequence equivalent. It is recommended that you execute a **Save Circuit** after issuing this and edit the saved version to correct possible misinterpretations. Some devices such as single-phase regulators will have to be manually converted.

### *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
```

### *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 often 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 again at any later time by invoking the Edit command or by continuing

with the next command line (see More or ~). The 'later time' does not have to occur immediately after definition. One can make up a script later that edits one or more object properties.

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 for further editing. 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 using this format. 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, which is always recommended for readability, 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.

Class.ElementName.Property1 = Value1 property2=value2 …

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 all 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
```

While all devices have a Like property, for Line objects and Transformer objects, users generally prefer to use the Linecode and Xfmrcode properties instead when objects have the same properties.

## *[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.

### *NewActor*

This command creates a new actor (OpenDSS Instance) and sets the new actor as the active actor. There can be only 1 circuit per actor. The NewActor Instruction will increment the variable NumOfActors; however, if the number of actors is the same as the number of available CPUs the new actor will not be created generating an error message. This instruction will deliver the ID of the active actor. This command does not require a precedent command.

### *Next*

{Year | Hour | t} Increments year, hour, or time as specified. If "t" is specified, then increments time by current step size.

### *NodeList [Circuit element name]*

Returns a list of node numbers for all conductors of all terminals of the active circuit element in the Result window or interface. If the optional circuit element name is supplied, the program makes it the active element. Usage:

```
NodeList
NodeList Line.Myline
```

### *NodeDiff*

The global result variable is set to voltage difference, volts and degrees, (Node1 - Node2) between any two nodes. Syntax:

```
NodeDiff Node1=MyBus.1 Node2=MyOtherBus.1
```

In the EXE version, this value will show up in the Result window. From the COM server, you can access the value from the Result property in the Text interface.

### *Obfuscate*

Change Bus and circuit element names to generic values to remove identifying names. Generally, you will follow this command immediately by

```
Save Circuit Dir=MyDirName.
```

This is useful when making a circuit model public and you wish the scrub a model of identifying names.

### *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 have to worry less about it causing a floating point exception.

## *Panel*

Displays main control panel window.

## *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.

## *Plot (options …)*

Check the Help in the EXE version for additions to the Plot command. There are many options to the Plot command and it has been moved to its own branch in the Help tree.

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 include (in order):

**Type** = {Circuit | Monitor | Daisy | Zones | AutoAdd | General (bus data) | Loadshape | Tshape | Priceshape |Profile}. 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 | Loadshape name]. Specifies what object to plot: meter zones, monitor or CSV file, or previously defined Loadshape. Note: for Loadshape plot, both the Mult and Qmult properties, if defined, are plotted.

**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 [0.85] (Color C3)

**R2** = pu value for tri-color plot mid range [0.50] (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. Ex: Channels=[1, 3, 5]

**Bases** = (array of base values for each channel for monitor plot). Default is 1.0 for each. This will per-unitize the plot to the specified bases. Set Bases=[ … ] **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.

**Buslist =** {Array of Bus Names | File=filename } This is for the Daisy plot.

```
Plot daisy power max=5000 dots=N Buslist=[file=MyBusList.txt]
```

A "daisy" marker is plotted for each bus in the list. Bus names may be repeated, which results in multiple markers distributed in a circle around the bus location. This gives the appearance of a daisy if there are several symbols at a bus. Not needed for plotting active generators.

**Phases** = {default* | ALL | PRIMARY | LL3ph | LLALL | LLPRIMARY | (phase number)} For Profile plot. Specify which phases you want plotted.

default = plot only nodes 1-3 at 3-phase buses (default)
ALL = plot all nodes
PRIMARY = plot all nodes -- primary only (voltage > 1kV)
LL3ph = 3-ph buses only -- L-L voltages)
LLALL = plot all nodes -- L-L voltages)
LLPRIMARY = plot all nodes -- L-L voltages primary only)
(phase number) = plot all nodes on selected phase

Note: Only nodes downline from an energy meter are plotted.

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. Some examples:

```
Plot circuit quantity=7 Max=.010 dots=Y Object=branchdata.csv
Plot General Quantity=2 Object=valuefile.csv
Plot type=circuit quantity=power
Plot Circuit Losses 1phlinestyle=3
Plot Circuit quantity=3 object=mybranchdata.csv
Plot daisy power 5000 dots=N
Plot daisy power max=5000 dots=N Buslist=[file=MyBusList.txt]
Plot General quantity=1 object=mybusdata.csv
Plot Loadshape object=myloadshape
Plot Tshape object=mytemperatureshape
Plot Priceshape object=mypriceshape
Plot Profile
Plot Profile Phases=Primary
```

**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 OpenDSS EXE will throw it in anyway.

**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.

## *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.

### *Pstcalc*

Pst estimation.

```
PstCalc Npts=nnn Voltages=[array] dt=nnn freq=nn lamp=120 or 230.
```

Set Npts to a big enough value to hold the incoming voltage array. dt = time increment in seconds. default is 1. freq = base frequency in Hz 50 or 60. Default is default base frequency. Lamp= 120 for North America; 230 for Europe. Default is 120. Example:

```
PSTCalc Npts=1900 V=[file=MyCSVFile.CSV, Col=3, Header=y] dt=1 freq=60
lamp=120
```

### *Puvoltages*

Just like the Voltages command, except the voltages are in per unit if the kVbase at the bus is defined.

### *Quit*

Shuts down DSS unless this is the DLL version. Then it does nothing; DLL parent is responsible for shutting down the DLL.

### *Reconductor Line1=name1 Line2=name2 {Linecode= | Geometry=} EditString="string" Nphases=nnn*

The Reconductor command will change the Linecode or Geometry definition for all Line objects between Line1 and Line2. You can specify Line1 and Line2 in either order; OpenDSS figures out how to trace between them, if there is a path. The OpenDSS will trace between Line1 and Line2, replacing either the *Linecode* property or the *Geometry* property with the value specified. There is error checking to make sure that the lines exist and there is a path between them. The command works like most other DSS commands. The rightmost specification of a property takes precedence.

*EditString* is an optional additional editing string to be executed for each Line object segment to change the value(s) of other properties of a line.

*Nphases* is an optional filter definition to limit the actions of this command to objects that match the specified number of phases.

Normally, specify only one of *Linecode=* or *Geometry=.* It is not possible to apply both.

Both lines must be within an Energymeter zone and must be in the same zone. One line must be upline from the other in the path back to the Energymeter location. Otherwise, an error occurs.

Line names are specified without the "Line." prefix since only Line objects are expected. If you specify a fully qualified line name (line.xxxx), the "Line." prefix is simply ignored.

Note: The action of the Reconductor command applies only to the active session. It does not change your input scripts. You have to save the circuit to a new circuit script to save the changes. However, you can simply add the Reconductor command to your existing script so that it changes the conductors each time you load the script.

The command would be placed **after** the meter zones are established, which occurs after the initial Solve command, CalcVoltageBases command, or the MakeBusList command.

```
…
Set VoltageBases=[138, 34.5, 12.47, 0.208]
CalcV
! Reconductor 3-phase sections only
Reconductor Line1=Lin230231  Line2=Lin230322 Linecode=397ACSR
Editstring="normAmps=650" Nphases=3  ! all on one line !!
```

### *Redirect [filename]*

Redirects the OpenDSS input stream to the designated file that is expected to contain DSS commands. It processes them as if they were entered directly into the command line. Do not wrap lines in the DSS files (you may use the ~, or more, command.) Similar to "Compile", but leaves current directory where it was when Redirect command was invoked. Will temporarily change to subdirectories if there are nested Redirect commands that change to other folders. See Compile.

Example:

```
redirect filename
```

Note: If you use a relative file name (no or partial path) the file expected to be in the current default data directory. If you get an unexpected message that a file was not found, check the path listed in the window caption (EXE control panel). Often, if you right-click on the script window and click on the "Change to this directory" option, the problem is resolve. Use full path names for library-type files of general data that are kept in other folders. (See also the CD command.)

### *Reduce {All | MeterName}*

Default is "All". Reduce the circuit according to circuit reduction options. See "Set ReduceOptions" and "Set Keeplist" options The Energymeter objects actually perform the reduction. "All" causes all meters to reduce their zones.

### *Refine_BusLevels*

This function takes the bus levels array and traces all the possible paths considering the longest paths from the substation to the longest branches within the circuit. Then, the new paths are filled with 0 to complement the oroginal levels proposed by the calcincmatrix_o command.

### *RelCalc [restore=Y/N]*

Perform predictive reliability calcs: Failure rates and number of interruptions. These calculations are performed by Energymeter objects on their zone. A radial system is assumed and the Energymeter object is the only element that can set the radial circuit tree.

Optional parameter: If restore=y automatic restoration of unfaulted downline section is assumed.

### *Remove {ElementName=} [KeepLoad=Y*/N] [EditString="…"]*

Remove (disable) all branches downline from the named PDelement by the "ElementName" property. Circuit must have an Energymeter on this branch. If KeepLoad=Y (default) a new Load element is defined at the "From" bus of the PDelement and set to present (kW, kvar) power flow solution of the element eliminated. The EditString is applied to each new Load element defined. This is useful for defining loadshapes and other load properties.

If KeepLoad=N, all downline elements are disabled.

Examples:

```
Remove Line.Lin3021
Remove Line.L22 Editstring="Daily=Dailycurve Duty=SolarShape
Remove Line.L333 KeepLoad=No
```

## *Rephase*

Generates a script file to change the phase designation of all Line objects downstream from a starting Line. Useful for such things as moving a single-phase lateral from one phase to another while keeping the phase designation consistent for reporting functions that need it to be (Note: this is not required for simply solving power flow. The OpenDSS doesn't care what you call the nodes.).

Parameters

```
StartLine=...
PhaseDesignation="..."
EditString="..."
ScriptFileName=...
StopAtTransformers=Y/N/T/F
```

Enclose the PhaseDesignation in quotes since it contains periods (dots). You may add an optional EditString to edit any other line properties.

Example:

```
Rephase StartLine=Line.L100  PhaseDesignation=".2"  EditString="phases=1"
ScriptFile=Myphasechangefile.DSS  Stop=No
```

This produces a script file that you may edit to achieve the desired result (some complex connections may get misinterpreted). Include this script in subsequent simulations to move the lines in question after building the base circuit. This command was used in the EPRI Green Circuits project to investigate the impacts of balancing the phases by moving entire laterals.

## *ReprocessBuses*

Forces reprocessing of bus definitions whether there has been a change or not. Use for rebuilding meter zone lists when a line length changes, for example or some other event that would not normally trigger an update to the bus list. May be done at any time, but is generally not necessary unless you notice something such as a line length change is not represented on a profile plot. These plots are created from Energymeter zones and it may be necessary to refresh the zones to reflect the proper lengths.

## *Reset {Meters | Monitors }*

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

## *Rotate angle=degrees*

Rotate the circuit plot coordinates by the specified number of degrees. This comes in handy sometimes to align with maps, etc.

## *Sample*

Force all monitors and Energymeters to take a sample for the most recent solution. Keep in mind that Energymeters will perform integration each time they take a sample.

## *Save*

Syntax:

```
[class=]{Meters | Circuit | Voltages | (classname)} [file=]filename
[dir=]directory
```

Default class = Meters, which saves the present values in both monitors and energy meters in the active circuit.

"**Save Circuit**" saves the present enabled circuit elements to the specified subdirectory in standard DSS form with a Master.txt file and separate files for each class of data. If Dir= not specified a unique name based on the circuit name is created automatically. If Dir= is specified, any existing files are overwritten.

"**Save Voltages**" saves the present solution in a simple CSV format in a text file. Used for VDIFF command.

Any class can be saved to a file. If no filename is specified, the classname is used.

The filename of the file written appears in the Result window after execution of this command.

## *Select [elementname] [terminal]*

Selects an element and makes it the active element. You can also specify the active terminal (default = 1). Syntax:

```
Select [element=]elementname  [terminal=]terminalnumber
```

Examples:

```
Select Line.Line1
~ R1=.1(continue editing)
Select Line.Line1 2
Voltages  (returns voltages at terminal 2 in Result)
```

## *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).

## *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).

### *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.

See the Options Reference for descriptions of the various options you can set with the Set command.

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

### *SetBusXY*

Bus=... X=... Y=... Set the X, Y coordinates for a single bus. Prerequisite: Bus must exist as a result of a Solve, CalcVoltageBases, or MakeBusList command.

### *SetkVBase [bus=…] [kvll=..]*

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

### *SetLoadAndGenkV*

Set load and generator object kv to agree with the bus they are connected to using the bus voltage base and connection type.

### *Show <Quantity>*

See the separate help on the Show command in the OpenDSS executable. Show commands are added frequently, often making this document out of date.

The Show commands generally 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 – so if you mistype the name of the quantity you want, you will get the sequence 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) May be huge!!!

**BusFlow** busname [MVA|kVA*] [Seq* | Elements]. Power flow report centered on specified bus. Can show values 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.

## *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 serverimplementation. 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  stepsize=15m   number=96
```

## *SolveAll*

Solves all the circuits (actors) loaded into memory. Depending on the activation of the parallel processing suite, the program will display different behaviors.

## *Summary*

Returns a power flow summary of the most recent solution in the global result string.

## *Tear_Circuit*

Estimates the buses for tearing the system in many parts as CPUs - 1 are in the local computer, is used for tearing the interconnected circuit into a balanced (same number of nodes) collection of subsystems for the execution of the A-Diakoptics algorithm.

## *TOP*

[class=]{Loadshape | Tshape | Monitor } [object=]{ALL (Loadshapes only) | objectname}. Send specified object to TOP. Loadshapes and TShapes must be hourly fixed interval.

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

### *TotalPowers*
Returns the total powers (complex) for ALL terminals of the active circuit element in the Result string. (See Select command.) Returned as comma-separated kW and kvar.

### *UpdateStorage*
Update Storage elements based on present solution and time interval. Causes the kWh stored values to be updated (integrated) based on the present kW charge or discharge values and the present time interval. This is done automatically by most time-sequential solution modes. This command is provided for scripting simulations.

### *UserClasses*
List of user-defined DSS Classes. Returns comma-separated list in Result variable.

### *Uuids*
Read UUIDs (v4) for class names and other CIM objects. Tab or comma-delimited file with full object name (or key) and UUID. Side effect is to start a new UUID list for the Export CIM100 command; the UUID list is freed after the Export UUIDs command.

### *Var*
Define and view script variables that can be used to substitute for actual values in a script. Variable names begin with **"@".** There are several predefined script variables that you can see by issuing the "var" command. Usage:

```
var @varname1=values  @varname2=value2    ...
var @varname1  (shows the value of @varname1)
var            (displays all variabiles and values)
```

Example of using a variable:

```
FileEdit @LastFile
```

A more detailed Example: Defining a phase shifting transformer. Use the carat "^" to separate the var name from subsequent text such as phase designation A, B, C.

```
Before redirecting to this file, set the following variables, for example

Var @Input=MyBus1
Var @Output=MyBus2
Var @TR=MyMainTrName
Var @Bridge=MyBridgeXf
Var @Jumper=MyJumperName
Var @MonitorPT=MyPTMonitorName
Var @Pt=MyPTName
Var @PTJumper=MyPTJumperName
Var @PTsec=MyPTSecBus
Var @PToutput=MyPTOutputBus

****************/

// Transformer code for phase shifter model
```

```
/*
   This is the main transformer
   It is a 3-winding transformer with special connections to get the phase
shift.
   So it has to be modeled as 3 single-phase units
*/

New  Xfmrcode.333KVA  phases=1  Windings=3 ppm=0
~ Xhl=5   Xht=10   Xlt=6  !
~ %noloadloss=0.3   %imag=0.3
~ kVs=[7.344  7.2  0.288]     ! ratings of windings
~ kVAs=[333 333 333]
~ %Rs = [0.5  0.5  0.5]
~ conns=[delta delta delta]

// Auto for bridging Transformer

New  Xfmrcode.Auto  phases=1  Windings=2 ppm=0
~ Xhl=1                 !---- %noloadloss=.2
~ kVs=[0.250 0.250]     ! ratings of windings
~ kVAs=[50000 50000]    ! Big transformer with low impedance
~ %Rs = [0.05 0.05]
~ conns=[delta delta]

// Wye-Connected Phase shifter model -- new design
//
// main phase shifter
New Transformer.@TR^A Xfmrcode=333KVA  Buses=[@Input.1.4  @Output.1.0
@Input.7.0 ]
New Transformer.@TR^B Xfmrcode=333KVA  Buses=[@Input.2.5  @Output.2.0
@Input.8.0 ]
New Transformer.@TR^C Xfmrcode=333KVA  Buses=[@Input.3.6  @Output.3.0
@Input.9.0 ]

! Bridging transformer for producing the phase shift
New Transformer.@Bridge^A  Xfmrcode=Auto Buses=[@Input.8.23  @Input.23.9]
New Transformer.@Bridge^B  Xfmrcode=Auto Buses=[@Input.9.31  @Input.31.7]
New Transformer.@Bridge^C  Xfmrcode=Auto Buses=[@Input.7.12  @Input.12.8]

! Connection Jumpers for connecting main phase shifter to bridging transformer
New Reactor.@Jumper^1 Phases=1 Bus1=@Input.4 Bus2=@Input.23 R=0 X=0.01
New Reactor.@Jumper^2 Phases=1 Bus1=@Input.5 Bus2=@Input.31 R=0 X=0.01
New Reactor.@Jumper^3 Phases=1 Bus1=@Input.6 Bus2=@Input.12 R=0 X=0.01

<snip>
```

## *Variable*

Syntax:

```
Variable [name=] MyVariableName  [Index=] IndexofMyVariable
```

Returns the value of the specified state variable of the active circuit element, if a PCelement. Applies only to PCelements (Load, Generator, etc) that contain state variables. Returns the value as a string in the Result window or the Text.Result interface if using the COM server.

You may specify the variable by name or by its index. You can determine the index using the **VarNames** command. If any part of the request is invalid, the Result is null. These may change from time to time, so it is always good to check.

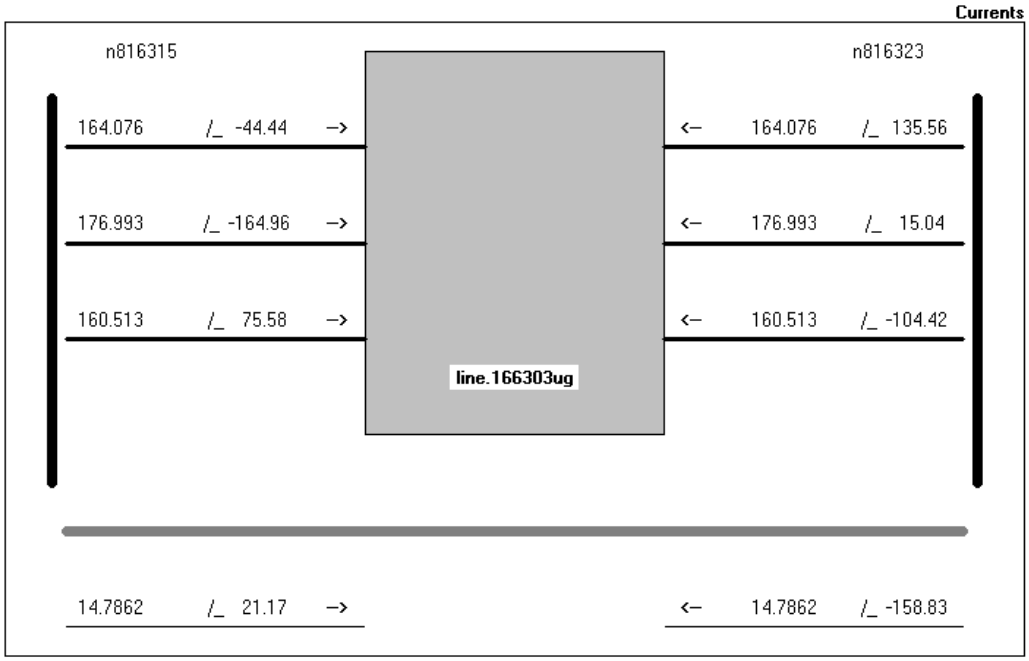
### *Varnames*
Returns all variable names for active element if PC element. Otherwise, returns null.

### *VarValues*
Returns all 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:




```
Visualize what=currents element= Line.Line1
Visualize powers Line.Line1
```

### *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.

### *Wait*
Pauses the scripting thread until all the active actors are Ready to receive new commands (have finished all their tasks and are ready to receive new simulation orders).

### *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.

Check the on-line Help while the DSS is running for information on additional commands that might have been added since the printing of this manual.

## OPTIONS REFERENCE

There are 111 options as of this writing. DSS options are set using either the Set command or the Solve command. The Solve command first executes the Set and then executes a solution. This allows for more concise syntax for some cases. For example, the sequence

```
Set mode=snapshot
Solve
Set mode=harmonics
Solve
```

May be accomplished with just 2 lines:

```
Solve mode=snapshot
Solve mode=harmonics
```

Selected Options currently implemented are described below. Options are added frequently. Check the on-line Help.

### *%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%.

### *ActiveActor =*

Gets/Sets the number of the active actor, if the value is * (set active actor=*), the commands send after this instruction will be aplied to all the actors previously created by the user.

### *ActorProgress =*

Gets progress (%) for all the actors created by the user when performing a task.

### *Addtype =*

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

### *ADiakoptics =*

{YES/TRUE | NO/FALSE} Activates the A-Diakoptics solution algorithm for using spatial parallelization on the feeder. This parameter only affects Actor 1, no matter from which actor is called. When activated (True), OpenDSS will start the initialization routine for the A-Diakoptics parallelization 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. The factors are applied to the XFkVA property of Load objects.

### *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. See also DefaultBaseFrequency.

### *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

***CapMarkerCode =***

Numeric marker code for capacitors. Default is 37. See MarkerCode option.

***CapMarkerSize =***

Size of Capacitor device marker. Default is 3.

***Cfactors=***

Similar to Set Allocationfactors= except this applies to the kWh billing property of Load objects. Sets all Load Cfactor properties to the same value. A typical value is 4. See online help on the Load object.

***circuit =***

Set the active circuit by name. (Current version allows only one circuit at a time, so this option is basically ignored at present.)

***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. Any line with sequence parameter inputs will use the long-line equivalent pi section.

***Class =***

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

***ConcatenateReports =***

Activates/Deactivates the option for concatenate the reports generated by the existing actors, if Yes, everytime the user executes a show/export monitor command the report will include the data generated by all the actors, otherwise the report will containThe data generated by the active actor.

***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. This is the default for the standard Snapshot mode as well as Daily and Yearly simulations where the stepsize is typically greater than 15 min.

**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. Use this for dutycycle mode and dynamic mode.

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.

### *Coverage =*

Percentage of coverage expected when estimating the longest paths on the circuit for tearing, the default coverage is the 90% (0.9), this value cannot exceed 1.0. When used with the "Set" command is used for the algorithm for estimating the paths within the circuit but when the "get" command is used after executing the tear_circuit command it will deliver the actual coverage after running the algorithm.

### *CPU =*

Gets/Sets the CPU to be used by the active actor.

### *DaisySize =*

Default is 1.0. Relative size (a multiplier applied to default size) of daisy circles on daisy plot.

### *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.

### *DefaultBaseFrequency=*

Set Default Base Frequency, Hz. The default value when first installed is 60 Hz. Side effect: Sets the solution Frequency and default Circuit Base Frequency. This value is saved in the Windows Registry when the DSS closes down. Therefore, it need only be set one time. This is useful for users studying 50 Hz systems.

### *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.

### *DSSVInstalled*

Returns Yes/No if the OpenDSS Viewer installation is detected in the local machine (Read Only).

### *EarthModel =*

One of {Carson | FullCarson | Deri*}. Default is Deri, which is a fit to the Full Carson that works well into high frequencies. "Carson" is the simplified Carson method that is typically used for 50/60 Hz power flow programs. Applies only to Line objects that use LineGeometry objects to compute impedances.

### *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. Use the complete path name for any other Editor. Does not require a circuit defined. This value is saved in the Windows Registry and need only be specified one time. EPRI uses the EditPlus editor.

### *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. See also the Select command.

### *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.

### *FuseMarkerCode =*

Numeric marker code (0..47 see Users Manual) for Fuse elements. Default is 25.

### *FuseMarkerSize =*

Size of Fuse marker. Default is 1.

### *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.

### *GISColor =*

Defines the active color for drawing forms on top of the active map in OpenDSS-GIS. The color is defined as a hex string representing a 24-bit integer (6 char). By default, this color is red (FF0000).

### *GISCoords =*

An array of doubles defining the longitud and latitude for an area to be used as reference for the OpenDSS-GIS related commands, the order goes as long1, lat1, long2, lat2 describing the area's top, left, bottom, right.

### *GISInstalled*

Returns Yes/No if the OpenDSS GIS installation is detected in the local machine (Read Only).

### *GISThickness =*

An integer defining the thickness for drawing lines on top of the active map in OpenDSS-GIS (default = 3).

### *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)
```

### *KeepLoad =*

Keeploads = Y/N option for ReduceOption Laterals option.

### *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.

### *LinkBranches*

Returns the names of the link branches used for tearing the circuit after initializing using set ADiakoptics = True. Using this instruction will set the Active Actor = 1. If ADiakoptics is not initialized, this instruction will return an error message.

### *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".)

### *LoadShapeClass =*

{Daily | Yearly | Duty | None*} Default loadshape class to use for mode=time and mode=dynamic simulations. Loads and generators, etc., will follow this shape as time is advanced. Default value is None. That is, Load will not vary with time.

### *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 =*

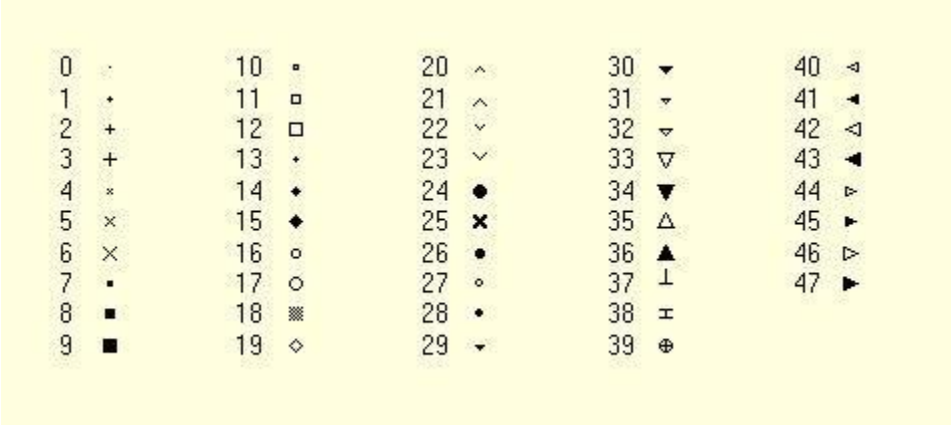
Weighting factor for Losses in AutoAdd functions. Defaults to 1.0. Autoadd mode minimizes

```
(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 (these are currently the SDL Components MarkAt options with this version). Marker codes are:




### *MarkCapacitors =*

{YES/TRUE | NO/FALSE} Default is NO. Mark Capacitor object locations with a symbol. See CapMarkerCode. The first bus coordinate must exist.

### *MarkFuses =*

{YES/TRUE | NO/FALSE} Default is NO. Mark Fuse locations with a symbol. See FuseMarkerCode and FuseMarkerSize. The bus coordinate must exist.

### *MarkPVSystems =*

{YES/TRUE | NO/FALSE} Default is NO. Mark PVSystem locations with a symbol. See PVMarkerCode. The bus coordinate must exist.

### *MarkReclosers =*

{YES/TRUE | NO/FALSE} Default is NO. Mark Recloser locations with a symbol. See RecloserMarkerCode and RecloserMarkerSize. The bus coordinate must exist.

### *MarkRegulators =*

{YES/TRUE | NO/FALSE} Default is NO. Mark RegControl object locations with a symbol. See RegMarkerCode. The bus coordinate for the controlled transformer winding must exist.

***MarkRelays =***

{YES/TRUE | NO/FALSE} Default is NO. Mark Relay locations with a symbol. See RelayMarkerCode and RelayMarkerSize. The bus coordinate must exist.

***MarkStorage =***

{YES/TRUE | NO/FALSE} Default is NO. Mark Storage device locations with a symbol. See StoreMarkerCode. The bus coordinate must exist.

***Markswitches =***

{YES/TRUE | NO/FALSE} Default is NO. Mark lines that are switches or are isolated with a symbol. See SwitchMarkerCode.

***Marktransformers =***

{YES/TRUE | NO/FALSE} Default is NO. Mark transformer locations with a symbol. See TransMarkerCode. The coordinate of one of the buses for winding 1 or 2 must be defined for the symbol to show

***Maxcontroliter =***

Max control iterations per solution. Default is 10.

***Maxiter =***

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

***MinIterations =***

Minimum number of iterations required for a solution. Default is 2, as it has been from the beginning. However, we now know that many simulations will require only 1 to go from one time step to another, particularly when using 1-s time step. Setting this property = 1 will save some solution time. OpenDSS will do more iterations if necessary.

***Mode=***

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

***Mode=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).

***Mode=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).

***Mode=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.

### *Mode=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).

### *Mode=Dynamics:*

Sets the solution mode for a dynamics solution. Must be preceded by a successful power flow solution so that the machines can be initialized. Changes to a default time step of 0.001s and ControlMode = TIME. Generator models are changed to a voltage source behind the value specified for transient reactance for each generator and initialized to give approximately the same power flow as the existing solution. Be sure to set the number of time steps to solve each time the Solve command is given.

### *Mode=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.

### *Mode=Harmonics:*

Sets the solution mode for a Harmonics solution. Must be preceded by a successful power flow solution so that the machines and harmonics sources can be initialized. Loads are converted to harmonic current sources and initialized based on the power flow solution according to the Spectrum object associated with each Load. Generators are converted to a voltage source behind subtransient reactance with the voltage spectrum specified for each generator. A Direct solution is performed for each harmonic frequency (more precisely, non-power frequency). The system Y matrix is built for each frequency and solved with the defined injections from all harmonic sources. A solution is performed for each frequency found to be defined in all the spectra being used in the circuit. Note that to perform a *frequency scan* of a network, you would define a Spectrum object with a small frequency increment and assign it to either an Isource or Vsource object, as appropriate.

### *Mode=HarmonicsT:*

Sets the solution mode for a Harmonics solution with time and control actions involved. Must be preceded by a successful power flow solution so that the machines and harmonics sources can be initialized. Loads are converted to harmonic current sources and initialized based on the power flow solution according to the Spectrum object associated with each Load. Generators are converted to a voltage source behind subtransient reactance with the voltage spectrum specified for each generator. A Direct solution is performed for each harmonic frequency (more precisely, non-power frequency). The system Y matrix is built for each frequency and solved with the defined injections from all harmonic sources. A solution is performed for each frequency found to be defined in all the spectra being used in the circuit. Note that to perform a *frequency scan* of a network, you would define a Spectrum object with a small frequency increment and assign it to either an Isource or Vsource object, as appropriate. This mode is adequate for performing harmonics simulation in which the values of the PCElements, time and the circuit topology change.

### *Mode=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 in past revisions was always 1 hour. However, it may now be any value.. 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.

### *Mode=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).

### *Mode=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.

### *Mode=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.

### *Mode=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.

### *Mode=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.

### *Mode=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.

### *Mode=Peakdays:*

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

### *NeglectLoadY =*

{YES/TRUE | NO/FALSE} Default is NO. For Harmonic solution, neglect the Load shunt admittance branch that can siphon off some of the Load injection current. If YES, the current injected from the LOAD at harmonic frequencies will be nearly ideal.

### *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.

### *Num_SubCircuits =*

This is the number of subcircuits in which the circuit will be torn when executing the tear_circuit command, by default is the number of local CPUs - 1.

### *NumActors*

Delivers the number of Actors created by the user, 1 is the default.

### *NumAllocIterations =*

Default is 2. Maximum number of iterations for load allocations for each time the AllocateLoads or Estimate command is given. Usually, 2 are sufficient, but some cases are more difficult. Execute an Export Estimation report to evaluate how well the load

allocation has worked.

***NUMANodes=***

Delivers the number of Non-uniform memory access nodes (NUMA Nodes) available on the machine (read Only). This information is vital when working with processor clusters (HPC). It will help you know the number of processors in the cluster.

***Number=***

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

***NumCores***

Delivers the number of physical processors (Cores) available on the computer considering all the NUMA nodes. If your computers processor has less than 64 cores, this number should be equal to the half of the available CPUs, otherise the number should be the same (Read Only).

***NumCPUs***

Delivers the number of threads (CPUs) available on the machine (read Only).

***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.

***OpenDSSViewer=***

{YES/TRUE | NO/FALSE} Default = No. Activates/Deactivates the extended version of the plot command for figures with the OpenDSS-Viewer (if installed).

***Overloadreport =***

{YES/TRUE | NO/FALSE} Default = FALSE. For yearly solution mode, sets overload.

***Parallel =***

{YES/TRUE | NO/FALSE} Default = No. Activates/Deactivates the parallel machine in OpenDSS, if deactivated OpenDSS will sequentially.

***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.

***ProcessTime (read only)***

The time in microseconds to execute the solve process in the most recent time step or

solution (read only).

***PVMarkerCode =***

Numeric marker code for PVSystem devices. Default is 15. See MarkerCode option.

***PVMarkerSize =***

Size of PVSystem device marker. Default is 1.

***QueryLog =***

{YES/TRUE | NO/FALSE} Default = FALSE. When set to TRUE/YES, clears the query log file and thereafter appends the time-stamped Result string contents to the log file after a query command, "?".

***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.

***RecloserMarkerCode =***

Numeric marker code (0..47 see Users Manual) for Recloser elements. Default is 17. (color=Lime).

***RecloserMarkerSize =***

Size of Recloser marker. Default is 5.

***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 | Laterals | 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.

"Ends" eliminates dangling ends only.

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

"Laterals [Keepload=Yes*/No]" uses the *Remove* command to eliminate all 1-phase laterals and optionally lump the load back to the parent 2- or 3-phase feeder bus (the default behavior).

Marking buses with "Keeplist" will prevent their elimination.

### *RegistryUpdate = {YES*/TRUE | NO/FALSE}*

Default is Yes/True. Update Windows Registry values upon exiting. You might want to turn this off if you temporarily change fonts or DefaultBaseFrequency, for example.

### *RegMarkerCode =*

Numeric marker code for Regulators. Default is 47. See MarkerCode option.

### *RegMarkerSize =*

Size of RegControl device marker. Default is 1.

### *RelayMarkerCode =*

Numeric marker code (0..47 see Users Manual) for Relay elements. Default is 17. (Color=Lime).

### *RelayMarkerSize =*

Size of Recloser marker. Default is 5.

### *SampleEnergyMeters ={YES/TRUE | NO/FALSE}*

Overrides default value for sampling EnergyMeter objects at the end of the solution loop. Normally, **Time** and **Duty** modes do not automatically sample EnergyMeters whereas Daily, Yearly, M1, M2, M3, LD1 and LD2 modes do.

Use this Option to turn sampling on or off.

### *SeasonRating =*

{YES/TRUE | NO/FALSE} Default = FALSE. Enables/disables the seasonal selection of the rating for determining if an element is overloaded. When enabled, the energy meter will look for the rating (NormAmps) using the SeasonSignal to evaluate if the PDElement is overloaded).

### *SeasonSignal =*

It is the name of the XY curve defining the ratings seasonal change for the PDElements in the model when performing QSTS simulations. The seasonal ratings need to be defined at the PDElement or at the general object definition such as linecodes, lineGeometry, etc.

### *Sec =*

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

### *ShowExport=*

{YES/TRUE | NO/FALSE} Default = FALSE. If YES/TRUE will automatically show the results of an Export command after it is written. Normally, the result of an Export command (a

CSV file) is not automatically displayed.

### *Stepsize (or h)=*

sets the time step size (default unit is sec) for the solution of the active circuit. Normally, specified for dynamic solution but is 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. Yearly simulations can also go in any time increment. Reasonable default values are set when you change solution **mode**s. You may specify the stepsize in seconds, minutes, or hours by appending 's', 'm', or 'h' to the size value. If omitted, 's' is assumed. Example: "*set stepsize=15m"* is the same as "*set stepsize=900*." Do not leave a space between the value and the character.

### *StepTime (Read Only)*

Process time + meter sampling time in microseconds for most recent time step - (read only).

### *StoremarkerCode =*

Numeric marker code (0..47 see Users Manual) for Storage elements. Default is 9.

### *StoremarkerSize =*

Size of Storage marker. Default is 1.

### *Switchmarkercode =*

Numeric marker code for lines with switches or are isolated from the circuit. Default is 4. See markswitches option and markercode option.

### *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.

### *TotalTime =*

The accumulated time in microseconds to solve the circuit since the last reset. Set this value to zero to reset the accumulator.

### *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.

### *TransMarkerCode =*

Numeric marker code for transformers. Default is 35. See MarkTransformers option and

MarkerCode option.

### *TransMarkerSize =*

Size of transformer marker. Default is 1.

### *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).

***Zmag=***

Sets the Zmag option (in Ohms) for ReduceOption Shortlines option. Lines have less line mode impedance are reduced.

***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.

# OpenDSS parallel processing suite

The classic OpenDSS program is a simulation platform built for execution in a single, sequential process. Each procedure/function is called from each object sequentially to perform a QSTS simulation. The performance that can be achieved is based on the structure of the low-level routines, the simplicity of the routines and the efficiency of the compiler.

EPRI has explored several methods to accomplish parallel processing in OpenDSS, including the parallelization of the whole program using a different interface (the Direct DLL API) and the modification of the solver using other programming languages, among other methods. However, these approaches demand significant extra work for the user and will be always tied to an interface. Consequently, desirable features that users are accustomed to, such as the COM interface, would be at risk of being deprecated for this type of processing.

EPRI has evolved OpenDSS into a more modular, flexible and scalable parallel processing platform we are calling OpenDSS-PM based with the following guidelines:

1. The parallel processing machine will be interface-independent
2. Each component of the parallel machine should be able to work independently
3. The simulation environment should deliver information consistently
4. The data exchange between the components of the parallel machine should respect the interface rules and procedures
5. The user handle of the parallel machine should be easy and support the already acquired knowledge of OpenDSS users

## *THE PARALLEL MACHINE*

To create the parallel machine, OpenDSS uses the actor model. There are several actor frameworks for Delphi proposed by various authors; however, we already had a framework developed to evaluate Delphi's tools for this purpose so we chose to use it. Each actor is created by OpenDSS-PM, runs on a separate processor (if possible) using separate threads and has its own assigned core and priority (*real-time* priority for the process and *time critical* for the thread).

The interface for sending and receiving messages from other actors will be the one selected by the user, this is, either the COM interface, the Direct DLL API, or a text script using the EXE version of the program. With this interface, the user will be able to create a new actor (instance), send/receive messages from these actors, and define the execution properties of the actors such as the execution core, simulation mode, and circuit to be solved, among others. This concept is shown in Figure 19.

Using the existing interfaces, the user can:

1. Request the number of available cores and the number of physical processors available.
2. Create/Destroy actors
3. Execute the simulation of each circuit concurrently and in parallel (hardware dependent)
4. Assign the core where the actor will be executed
5. Modify the simulation settings for the active actor

6. Set the name of the circuit that will be simulated

Basically, the user can do the same things he can do with the classic version plus the operations related with parallel processing.

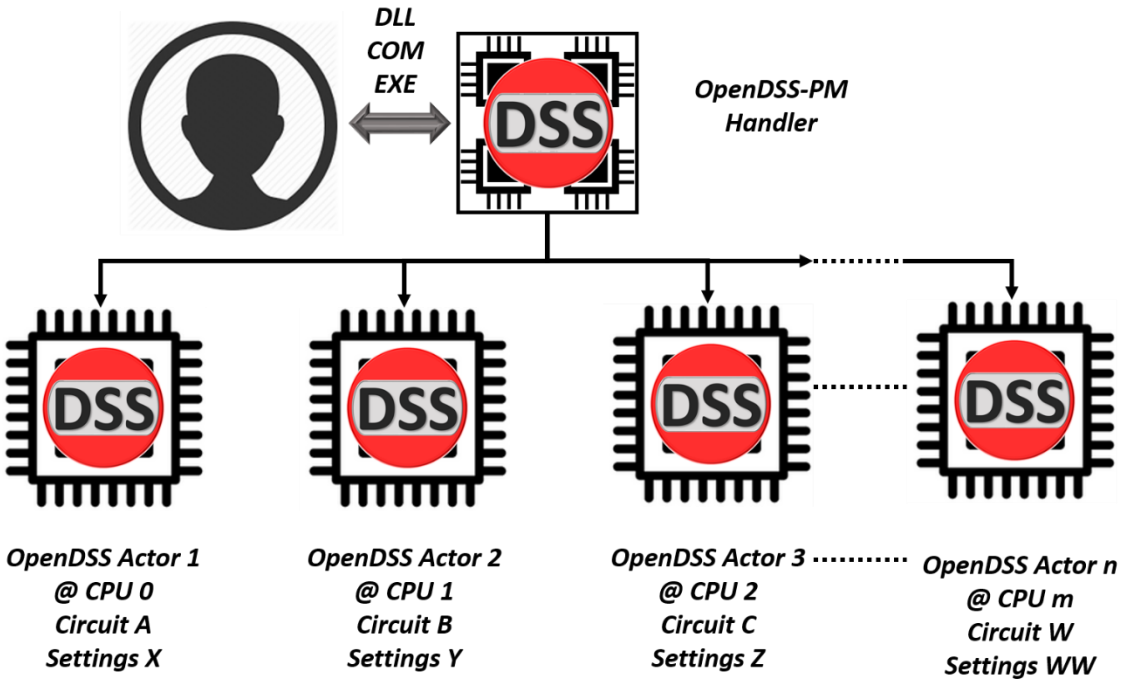



**Figure 19. Operational architecture proposed for OpenDSS**

As can be seen in Figure 19, the interface will work as the communication medium between the different actors on the parallel machine. This feature enables several simulation modes inspired in parallel computations such as temporal, Diakoptics, among others. These type of simulations will be driven by an external program that will handle the actors within the parallel machine, in keeping with the actor concept as a *message driven state machine*.

To operate the parallel machine, the suggested procedure is as follows:

1. The program will create a new default actor every time the start function is called in an OpenDSS-PM COM or DLL interface or when the EXE version is started. OpenDSS-PM will create Actor 1, designate a memory space, open an instance for KLUSolve, and define the execution thread. In return, OpenDSS-PM will return to the user an ID (integer) to identify the created actor.
2. After the program has started the user issues the NewActor command to create a new actor.
3. After a new actor is created, the user will designate the core in which the actor is to be executed using the Set Core=nn command. This command will apply to the active actor using the selected interface. Core 0 will be the default core for the initial actor created at start up, but this can be changed.
4. To change the active actor, the user will issue the Set ActiveActor=nn command.

5. After the active actor is set, the user can execute OpenDSS commands as done for the classic version using the selected interface. The commands will apply to the process executed by the active actor.
6. There are two options for solving the systems with actors:
    a. Solve the active actor
    b. Solve all of the actors
7. If the user selects to solve only the active actor while there are other actors created, the user can continue to interact with the other actors while the solving actor is working. On the other hand, if the user selects to solve all the actors, the ability to exchange information with an actor will depend on the availability of its core. If there are not enough cores to handle the request the user program will have to wait until one of the actors finishes the solution routine.
8. Each actor can be asked for data and can store its own monitor and energy meter samples locally.

To make this possible it is necessary to clone essential classes of OpenDSS. This cloning process must be done every time the user requests it. By default, there will be at least 1 actor active for performing simulations and the default core will be Core 0.

The configuration of each instance (actor) can be made sequentially, however, the parallel processing of each actor circuit is done using multithreading, defining the process and thread affinity and its priority.

## *INSTRUCTION SET FOR PARALLEL PROCESSING*

### *NumCPUs*

Returns the number of threads of virtual cores (CPUs) available in the computer. This is a *read-only* value and must be executed using the "*Get*" command (opposite of the Set command). The number of CPUs will be equal to the number of threads of the active processor multiplied by the number of NUMA nodes. For more information about the processor architecture visit: http://www.howtogeek.com/194756/cpu-basics-multiple-cpus-cores-and-hyper-threading-explained/

### *NumCores*

Returns the number of physical processors (Cores) available in the computer. If the computer has less than 64 CPUs the number of cores will be the half of the number of CPUs (2 threads per Core), otherwise, these numbers will be the same. This is a read-only value and must be executed using the "*Get*" command. The value is returned in the Result string in the EXE version and the *@result* variable for the text scripting interface. The string is also available in the Result property of the Text interface in the COM interface and the result of the DSSPut_Command function of the DLL interface. The number of Cores will be equal to the number of number of physical cores of the active processor multiplied by the number of NUMA nodes.

### *NewActor*

This command creates a new actor (OpenDSS Instance) and sets the new actor as the active actor. There can be only 1 circuit per actor. The *NewActor* Instruction will increment the variable *NumOfActors*; however, if the number of actors is the same as the number of available CPUs the new actor will not be created, generating an error message. This instruction will return the ID of the active actor. This command does not require a precedent command.

### *NumActors*
Returns the number of actors created, this number cannot be higher than the number of available CPUs. This is a read-only value and must be executed using the "*Get*" command. By default, there is at least 1 actor created.

### *ActiveActor*
This option can be used to get or set the active actor. The active actor cannot be higher than *NumActors* nor lower than 1. Use the commands *Get* or *Set* to operate on this value. If the value assigned is the star character (*), all the commands after setting ActiveActor=* will be executed for all the actors. This feature only works with the scripting tool or when using the Text interface in COM/DLL.

### *CPU*
This option can be used to get or set the CPU assigned to the *active actor*. CPU numbers are indexed from 0..n. That is, the CPU number is 0-based and cannot be greater than (*NumCPUs* – 1). Actors are indexed from 1..n. When a new actor is created it is automatically assigned to the corresponding core in sequence starting from 0 (i.e. Actor 1 will be assigned to CPU 0, Actor 2 to CPU 1, and Actor 3 to CPU 2 and so on). Use the command *Get* or *Set* commands to operating on this value.

### *ActorProgress*
This option will show on the summary tab the progress for all the actors when performing a task. For example, if each actor is performing a *yearly* simulation and the user wants to know the progress of each actor (%), this is the instruction that must be used. This is a read-only value and must be executed using the "*Get*" command.

### *ClearAll*
Clears all the existing circuits and their classes. After executing this command OpenDSS will create only one actor, the active actor will be actor number 1 and the CPU assigned will be the CPU assigned originally for this actor.

### *Wait*
This function freezes the execution of the Frontal Panel Actor until all the other actors are available to receive new messages or start new processes. By using this function, it is probable that the user may not be able to see updates on the summary/results tab, but is a very good mechanism for synchronizing all the actors within the parallel environment and make classical scripts compatible with OpenDSS.

### *Parallel*
This function enables/disables the parallel processing functionalities of OpenDSS. By disabling this features, OpenDSS will behave as the classic version of OpenDSS and even if the user can create new actors, each actor will operate sequentially one after the other (no concurrency). By default, this option is disabled and it is required to activate it to gain access to the parallel processing features of OpenDSS.

### *SolveAll*
This command starts the solution process for all the existing actors.

### *ConcatenateReports*
This option can be used to Enable/Disable the report concatenation of the monitor's content. When disabled (default) the user needs to specify the actor to gain access to the monitor's data using the commands show or export. When enabled, this option allows to summarize all monitors content with the same name working at different actors into a single file.

### *CalcIncMatrix*
This command can be used to calculate the Branch-To-Node incidence matrix (B2N) of the active circuit. The calculation is performed considering the PDElements as branches (rows) and the buses as nodes (Columns). The order of the PDElements as rows goes as follows: Lines, transformers, capacitors in series and reactors in series.

### *CalcIncMatrix_O*
This command can be used to calculate the Branch-To-Node incidence matrix (B2N) of the active circuit. However, this command delivers and optimized matrix that is organized inside the algorithm by using the CktTree class. The calculation is performed considering the PDElements as branches (rows) and the buses as nodes (Columns). Additionally, this algorithm calculates the levels of each bus to populate an internal array called BusLevels. The Bus level is an integer that reveals how far in terms of buses is the Bus respect to the feeder's backbone. The Backbone is a randomly selected continuous path from the feeder head to a point in the feeder selected as feeder end.

### *Tear_Circuit*
This command tears apart the circuit in many pieces as CPUs – 1 are available in the local PC. The tearing takes place by using the *CalcIncMatrix_O* command internally. Then, the algorithm estimates the best route for generating a set of sub-Circuits by placing an energy meter at the tearing points selected by the algorithm. As a result, this command will generate a folder called *Torn_Circuit* inside the project's folder. Each sub-Circuit will be contained in this folder starting at the substation (*Torn_Circuit* folder root) and the other Sub-Circuits in folders called *Zone_1, Zone_2* and so on until the total number of sub-Circuits.

### *Export IncMatrix*
This command exports in a csv file the B2N matrix using a compressed coordinated format (Row, Column, and Value). This format is used to facilitate uploading this data into a sparse matrix.

### *Export IncMatrixRows*
This command exports in a csv file the names of the rows (PDElements) of the latest calculation of the B2N matrix using any of the methods for this purpose.

### *Export IncMatrixCols*
This command exports in a csv file the names of the columns (buses) of the latest calculation of the B2N matrix using any of the methods for this purpose.

### *Export BusLevels*
This command exports in a csv file the names and levels of the columns (buses) of the latest calculation of the B2N matrix using any of the methods for this purpose.

### *Abort*
This command aborts all the simulation jobs running and gives back the control to the caller.

### *CalcLaplacian*
This command calculates the Laplacian matrix using the incidence matrix previously calculated, this means that before calling this command the incidence matrix needs to be calculated using calcincmatrix/calcincmatrix_o.

### *Export Laplacian*
This command exports in a csv file the Laplacian matrix previously calculated using a compressed coordinated format (Row, Column, and Value). This format is used to facilitate uploading this data into a sparse matrix.

### *Clone*
This command clones the active circuit as many times as specified in the argument, for example, Clone 4 will create 4 actors with the same model as the one in memory at the time "Clone" is invoked. The argument needs to be a positive number greater than 0. If the number of clones requested exceeds the number of CPUs, OpenDSS will deliver and error message (7001). If the number of clones requested is invalid the error message will be 7004.

### *NUMANodes*
Returns the number of NUMA nodes in the local hardware architecture, this number should match with the number of processor's sockets (if you are working with HPC, it should be 1 for standard computers). This is a read-only value and must be executed using the "*Get*" command.

## *ERROR CODES ASSOCIATED TO THE PARALLEL MACHINE (PM) OPERATION*

| Code | Description |
|---|---|
| 7000 | This error is generated when the user is trying to create a new circuit but the number of available CPUs is already assigned to other circuits. To avoid this message, before creating a new circuit check if there are available CPUs by requesting OpenDSS-PM to deliver the number of CPUs and the number of existing Actors. |
| 7001 | This error is generated when the user tries to create a new actor but there are no more CPUs available. The number of actors cannot exceed the number of available CPUs on the computer. Check the number of actors (*NumActors*) and the number of CPUs (*NumCPUs*) before executing the *NewActor* Command. |
| 7002 | This message is displayed when the user is trying to activate an inexistent actor. To avoid this message, check the number of actors (*NumActors*) before activating one. The ID of the actor should be less than or equal to *NumActors*. |
| 7003 | This message is displayed when the user is trying to assign a non-existent CPU to the active actor. To avoid this message, check the number of CPUs (*Get NumCPUs*) before activating one. The CPU ID should be lower to the Number of existing CPUs (starts in CPU 0). |

7004 The number of clones requested is not valid, this number cannot be 0 or a negative number.

---

## *EXAMPLES*

The following example will create three actors using a DSS script with the EXE version of OpenDSS-PM. Then, the simulation mode, start time and number of steps is set for each actor. Finally, all the compiled systems are solved concurrently. The system being solved is EPRI's Ckt5 test circuit.

```
clearAll
set parallel=No

compile "C:\Program Files\OpenDSS\EPRITestCircuits\ckt5\Master_ckt5.dss"
set CPU=0
Solve

Clone 2

set parallel=Yes

set activeActor=*
set mode=yearly number=2000 totaltime=0

Set ActiveActor=1
Set hour=0

set activeActor=2
set hour = 2000

set activeActor=3
set hour = 4000

SolveAll
Wait

set ConcatenateReports=Yes
show monitor MS2
```

You need to be careful when organizing OpenDSS-PM scripts – remember that now everything is happening at the same time. If the user wants to see the voltages, export monitors or execute any other "report" command, it is necessary to wait until all the processes are executed. Otherwise, OpenDSS-PM will execute the processes immediately. As in the classic OpenDSS version, to have control of the operations that you are performing in the script, you can select the part of the script that you want to execute, and then, by right-clicking with the mouse, select "Do selected" from the pop-up menu as shown in **Figure 20**.

If you want to verify the execution of the solution in terms of CPU allocation after executing the script presented above, execute the windows *Resource Monitor (http://www.digitalcitizen.life/how-use-resource-monitor-windows-7)* or the *Task Manager.* You will be able to check how each system is being solved on a separate CPU. In the case of the example script, the first actor will be executed on CPU 0, the second on CPU 2 and the third on CPU 3. In the *performance* tab of the *Task Manager* you will see the utilization of the processors by the different actors when executing the solve command as shown in Figure 21.

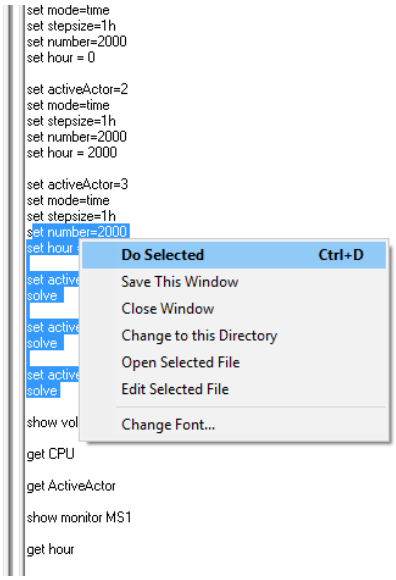



**Figure 20. Selecting the parts of the script that the user wants to be executed**

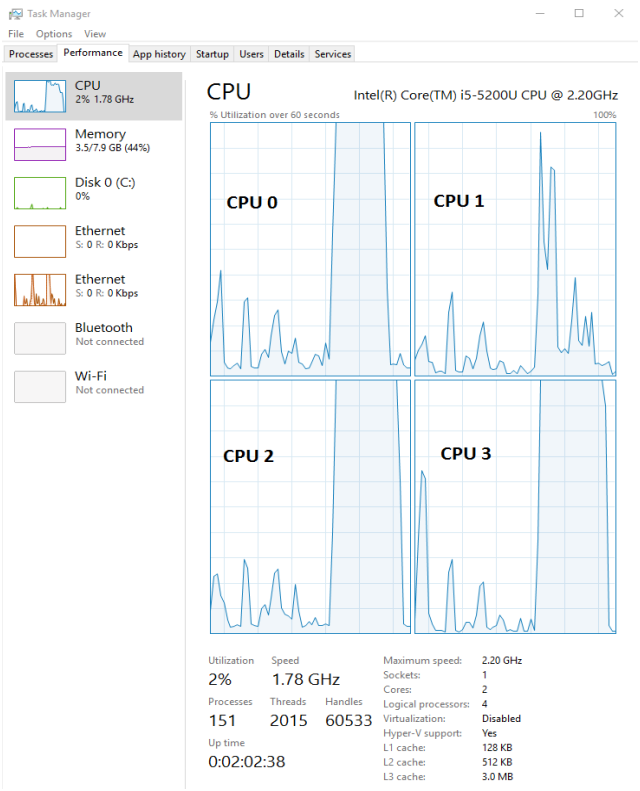



**Figure 21. Processor usage when performing parallel processing with OpenDSS**

As a general recommendation, do not to use all the available CPUs for an OpenDSS simulation. Your system can freeze and will not let you to perform other activities in parallel to the simulation.

# OpenDSS-GIS Integration

OpenDSS-GIS is a complementary program that allows users to incorporate GIS capabilities into their power system analysis tools. OpenDSS-GIS operates as a JSON server through TCP that provides GIS data and visualization to the connected clients.
OpenDSS provides a set of commands for driving OpenDSS-GIS, facilitating the interaction with the last software to integrate GIS capabilities and visualizations to the OpenDSS model. These capabilities can also be accessed through COM/DLL interfaces, making them part of customized developments using OpenDSS as simulation engine.

The list of GIS commands available in OpenDSS for driving OpenDSS-GIS are presented in Table 1. All the commands must be preceded by the command GIS (e.g. *GIS Start*). The command set presented in this chapter is compatible with OpenDSS-GIS v 1.7.

**Table 1.**
**GIS command set**

| | | | |
|---|---|---|---|
| *BatchFormat* | *GetRoute* | *PlotCircuit* | *Start* |
| *ClearMap* | *GetSelect* | *PlotFile* | *StopDraw* |
| *Close* | *GISColor (get/set)* | *PlotPoint* | *StopSelect* |
| *Distance* | *GISCoords* | *PlotPoints* | *Text* |
| *DrawLine* | *GISCoords (get/set)* | *RouteDistance* | *TextFromFile* |
| *DrawLines* | *GISThickness (set)* | *RouteSegDistances* | *WindowDistribLR* |
| *ExportMap* | *GoTo* | *Select* | *WindowDistribRL* |
| *FindRoute* | *Installed (get)* | *ShowBus* | *WindowSize* |
| *Format* | *JSONRoute* | *ShowCoords* | *ZoomMap* |
| *GetAddress* | *LoadBus* | *ShowLine* | |
| *GetPolyline* | *MapView* | *ShowRoute* | |

## *DRIVING OPENDSS-GIS*

### *GISCoords*

The first step before using OpenDSS-GIS is to define the GIS coordinates for the active model in OpenDSS. The GIS coordiantes can be defined using the GISCoords command in the same way the BusCoords are defined, this is, by typing the commands and using the path to the file containing the GIS coordinates as argument:

```
GISCoords myFilePath
```

The GIS coords file format is like the BusCoords file, it is composed by comma separated values that include the name of the bus, the latitude and longitude.

```
…
MyNode1,xx.xxxxxxxx,-xx.xxxxxxxx
MyNode2,xx.xxxxxxxx,-xx.xxxxxxxx
MyNode3,xx.xxxxxxxx,-xx.xxxxxxxx
MyNode4,xx.xxxxxxxx,-xx.xxxxxxxx
…
```

If this file is not defined in the model, the model's schematic cannot be displayed on the map.

### *Close*

Closes OpenDSS-GIS and finishes the connection.

```
GIS Close
```

### *FindRoute*

Finds a route between the given buses using roads and geographical information. The buses are defined using the *GISCoords* buffer. This command is also compatible with *LoadBus*. In the following example, the route between two given coordinates will be calculated. The calculation takes place by following streets and intersections. The router used in OpenDSS-GIS is OSRM (http://project-osrm.org/).

```
set GISCoords=[35.92209179726962, -84.14211464700368,
35.90779395922772, -84.14752180168537]

GIS FindRoute
```

Using the LoadBus command, if Bus1 and Bus2 correspond to the coordinates presented above respectively:

```
GIS LoadBus Bus1
GIS LoadBus Bus2
GIS FindRoute
```

The following conditions need to be fulfilled:

1. OpenDSS-GIS must be installed

2. OpenDSS-GIS must be initialized (use GIS Start command)

3. The model needs to have the correct GISCoords file

### *RouteSegDistances*

Returns the distances in meters for each segment within the route previously estimated. For example, consider the route presented above, this route has 6 segments from the orgin to the destination completing 2.841 km:

```
set GISCoords=[35.92209179726962, -84.14211464700368,
35.90779395922772, -84.14752180168537]
GIS FindRoute
GIS RouteSegDistances
```

With this command OpenDSS-GIS will return the distances of each segment included in this route as follows:

```
[160.90,166.30,408.20,141.60,1504.10,460.20,0.00,]
```

The following conditions need to be fulfilled:

1. OpenDSS-GIS must be installed

2. OpenDSS-GIS must be initialized (use GIS Start command)

3. The model needs to have the correct GISCoords file

### *GetRoute*

Returns the GIS coords of the route between 2 buses step by step in a 2-dimension array. For example, consider the route found above, the command *GetRoute* can be applied as follows:

```
      set        GISCoords=[35.92209179726962,        -84.14211464700368,
35.90779395922772, -84.14752180168537]
      GIS FindRoute
      GIS GetRoute
```

OpenDSS-GIS will return the coordinates of each segment (2 coordinates per segment) included in this route as follows:

```
      [-84.1423200000000000,35.9222830000000000,-
84.1411360000000000,35.9219720000000000,-
84.1398070000000000,35.9230060000000000,-
84.1375080000000000,35.9214170000000000|-
84.1367180000000000,35.9205030000000000,-
84.1378340000000000,35.9196210000000000,-
84.1421150000000000,35.9173700000000000|-
84.1444060000000000,35.9160410000000000|-
84.1484180000000000,35.9127300000000000|-
84.1506070000000000,35.9110880000000000,-
84.1490140000000000,35.9091680000000000|-
84.1480800000000000,35.9083550000000000|-
84.1474460000000000,35.9078590000000000,]
```

In the format shown above the character comma (,) is a vector separator between the same pair latitude-longitude, the character vertical bar (|) is used as row separator.

The following conditions need to be fulfilled:

1. OpenDSS-GIS must be installed

2. OpenDSS-GIS must be initialized (use GIS Start command)

3. GISFindRoute has been executed at some point before this command (at least once)

4. The model needs to have the correct GISCoords file

### *RouteDistance*

Returns the total distance in meters of the last route calculated between 2 buses. For example, consider the route presented above, this route has 6 segments from the orgin to the destination completing 2.841 km:

```
      set       GISCoords=[35.92209179726962,       -84.14211464700368,
35.90779395922772, -84.14752180168537]
      GIS FindRoute
      GIS RouteDistance
```

With this command OpenDSS-GIS will return the total distances of the route calculated:

```
      [2841.40,]
```

The following conditions need to be fulfilled:

1. OpenDSS-GIS must be installed

2. OpenDSS-GIS must be initialized (use GIS Start command)

3. GISFindRoute has been executed at some point before this command (at least once)

4. The model needs to have the correct GISCoords file

### *JSONRoute*

Exports the JSON script describing the last route calculated between 2 buses into a text file. The output from OpenDSS is the path to the JSON file. The JSON file contains all the details of the route, including steps, hints, and locations, among other information that may results interesting for the user. The following is an extract of the JSON generated for the route calculated above:


```
{"code":"Ok","waypoints":[{"hint":"s1MuhctTLoWZAAAAwgAAAOkAAAA2AAAAPNh_
QjNooUJAxcFCIZ6zQZkAAADCAAAA6QAAADYAAAB8QwAAEBf8-
mshJALfF_z6rCAkAgMATw7FAVHD","distance":28.251245,"location":[-
84.14232,35.922283],"name":""},{"hint":"r3uQgOl7kIA3AAAAHgAAAAAAAAAAAA
AjseeQujnKEIAAAAAAAAAADcAAAAeAAAAAAAAAAAAAAB8QwAACgP8-hPpIwK-Avz60-
gjAgAAbwjFAVHD","distance":9.87283,"location":[-
84.147446,35.907859],"name":"Lovell
Road"}],"routes":[{"legs":[{"steps":[{"intersections":[{"out":0,"entry"
:[true],"location":[-
84.14232,35.922283],"bearings":[49]}],"driving_side":"right","geometry"
:"gagzEn`q`OkAiBlAuA\\Q\\Y","duration":44.5,"distance":160.9,"name":"",
"weight":44.5,"mode":"driving","maneuver":{"bearing_after":49,"location
":[-
84.14232,35.922283],"type":"depart","bearing_before":0,"modifier":"righ
t"}},{"intersections":[{"out":0,"in":2,"entry":[true,true,false],"locat
ion":[-
84.141136,35.921972],"bearings":[45,225,330]}],"driving_side":"right","
geometry":"i_gzEbyp`OWa@GIkAiBYe@iAmA","duration":26.6,"distance":166.3
,"name":"Corridor Park Boulevard","weight":26.6,"mode":"driving","maneu…
```


For more information about the JSON string visit http://project-osrm.org/docs/v5.24.0/api/#.

The following conditions need to be fulfilled:

1. OpenDSS-GIS must be installed

2. OpenDSS-GIS must be initialized (use GIS Start command)

3. GISFindRoute has been executed at some point before this command (at least once)

4. The model needs to have the correct GISCoords file

### *ShowRoute*

Shows the last route calculated between 2 buses in the OpenDSS-GIS map. For example, consider the route calculated in previous examples, for visualizing the route in OpenDSS-GIS use the following commands:

```
      set       GISCoords=[35.92209179726962,       -84.14211464700368,
35.90779395922772, -84.14752180168537]
      GIS FindRoute
      set GISColor = 0000FF   ! the color for the route (blue)
      set GISThickness=4      ! the thickness for the line (4 pix)
      GIS ShowRoute
```

The result in OpenDSS-GIS will look as shown in Figure 22.

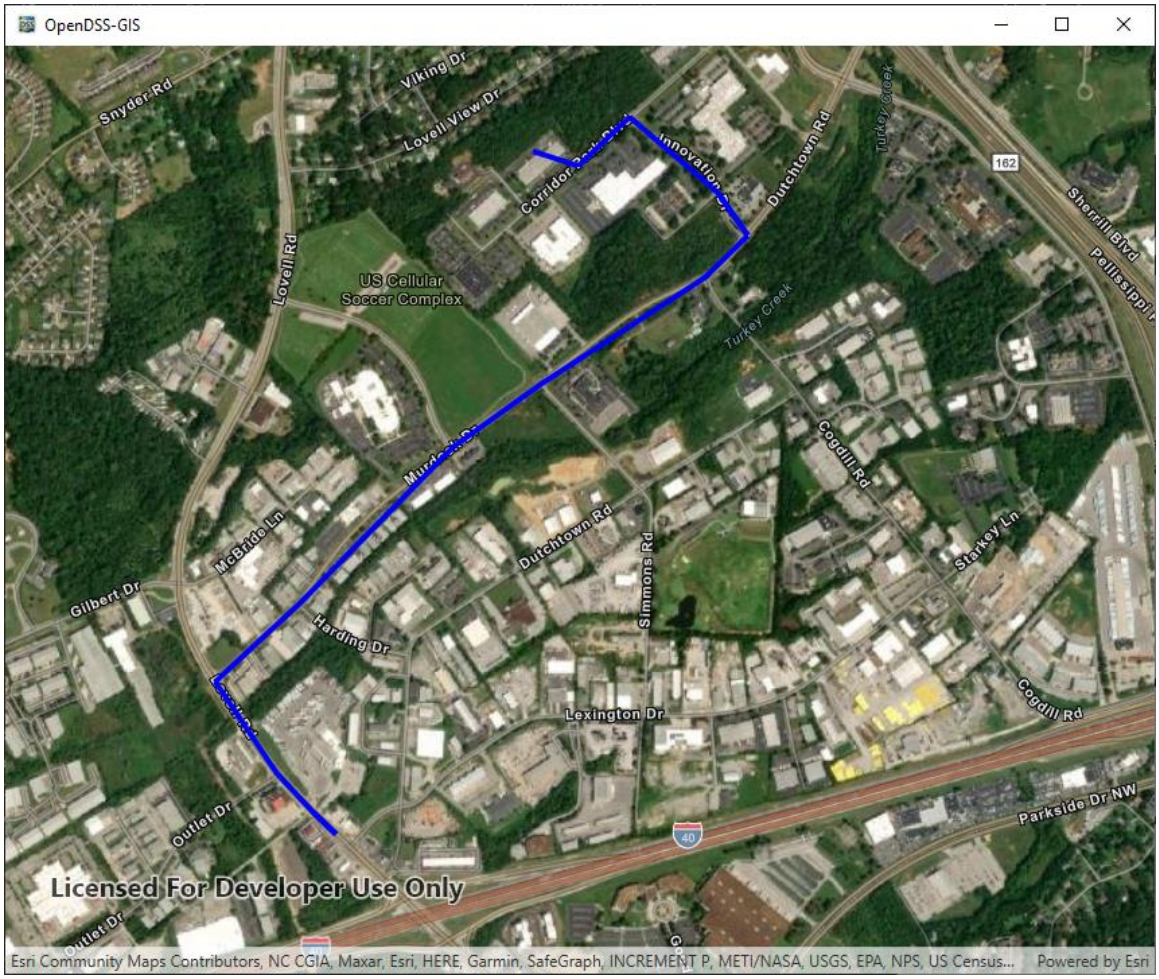



**Figure 22. Showing route in OpenDSS-GIS**

The following conditions need to be fulfilled:

1. OpenDSS-GIS must be installed

2. OpenDSS-GIS must be initialized (use GIS Start command)

3. GISFindRoute has been executed at some point before this command (at least once)

4. The model needs to have the correct GISCoords file

## *Start*

This command checks if OpenDSS-GIS is installed in the local PC, if not, it will return an error message telling the user that the software is not present. Otherwise, OpenDSS will start OpenDSS-GIS as shown in Figure 23. After starting OpenDSS-GIS, OpenDSS will establish connection with the TCP server.

```
GIS Start
```


**Figure 23. Starting OpenDSS-GIS from OpenDSS**

If there is any error during this operation, the user can try to start OpenDSS-GIS manually and then execute the command *GISStart* to stablish connection between OpenDSS-GIS and OpenDSS. If successful, OpenDSS will report the connection using the results window as shown at Figure 24. GISStart can be used as required for re-establishing communication with OpenDSS-GIS after an error message is obtained.

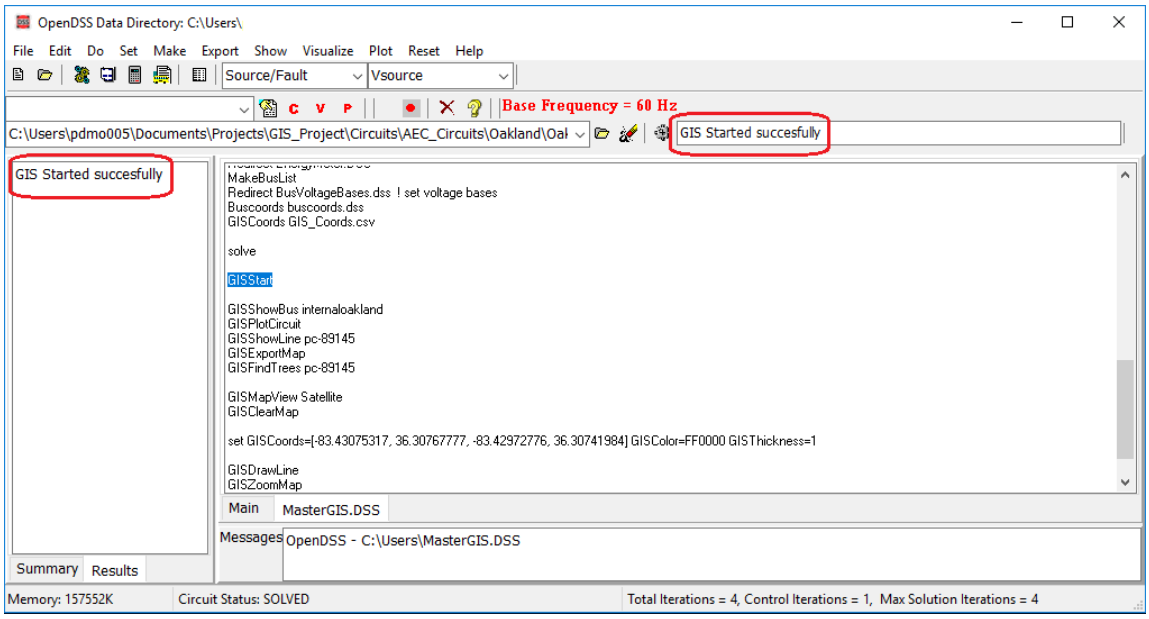



**Figure 24. Connecting OpenDSS and OpenDSS-GIS Succesfully**

## *PlotCircuit*

With this command the user can plot the model's schematic on top of the map. OpenDSS-GIS will zoom in the map to display the entire model. The schematic draw will follow the features set when assigning values to options *GISColor* and *GISThickness*. By default, GISColor ir red (FF0000) and GISThickness is 3. Consider the model proposed in

**Figure 25**. After executing *GISPlotCircuit* the model schematic will be display in OpenDSS-GIS as shown at

**Figure 25**.

```
GIS PlotCircuit
```

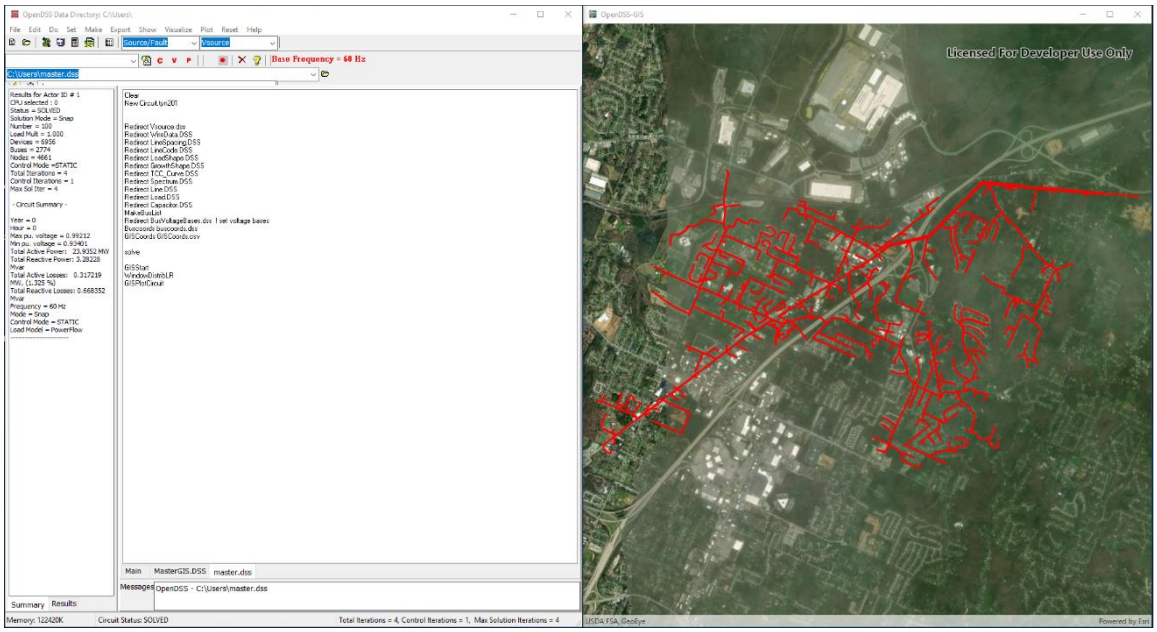



**Figure 25. Plotting circuit model in OpenDSS-GIS**

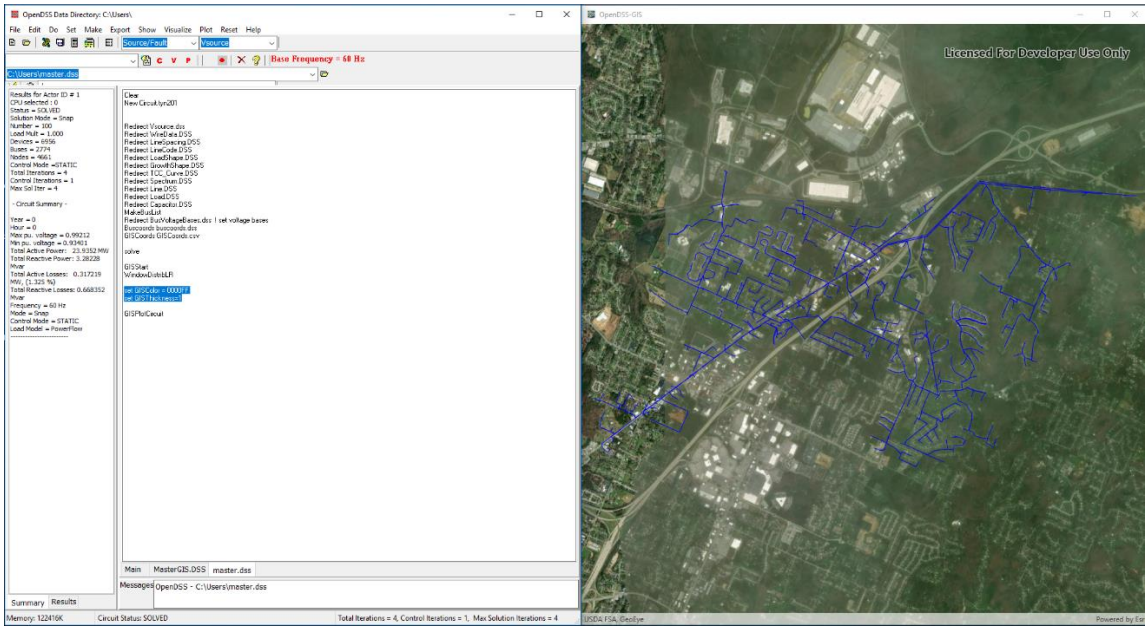



**Figure 26. changing the default settings for plotting the circuit diagram in the map**

Now, suppose you want to change the circuit line color to blue and make the model thinner, set GISColor = 0000FF (blue) and GISThickness = 1. The output will be as shown at Figure 26.

## *PlotFile*

With this command the user can plot a customized graph on top of the map. The path to the file containing the plot is the argument for this command. The file contains the coordinates, color and thickness for each line that composes the draw in a comma separated values file as follows:

```
…
myLong11,myLat11,myLong12,myLat12,myColor1,myThickness1
myLong21,myLat21,myLong22,myLat22,myColor2,myThickness2
myLong31,myLat31,myLong32,myLat32,myColor3,myThickness3
…
```

Where `myLongXX` and `myLatXX` are longitude and latitude for describing the beginning and end of the line, `myColorX` is the color of the line and `myThicknessX` is the thickness expressed in pixels. X refers to the index for differenciating data within the given example.

At Figure 27, there is an example for plotting the power flow across the feeder using red as the for the lines with maximum flow and a transition to blue passing thorugh green to display the lines with lower power flow.

```
GIS PlotFile C:\myFile.csv
```

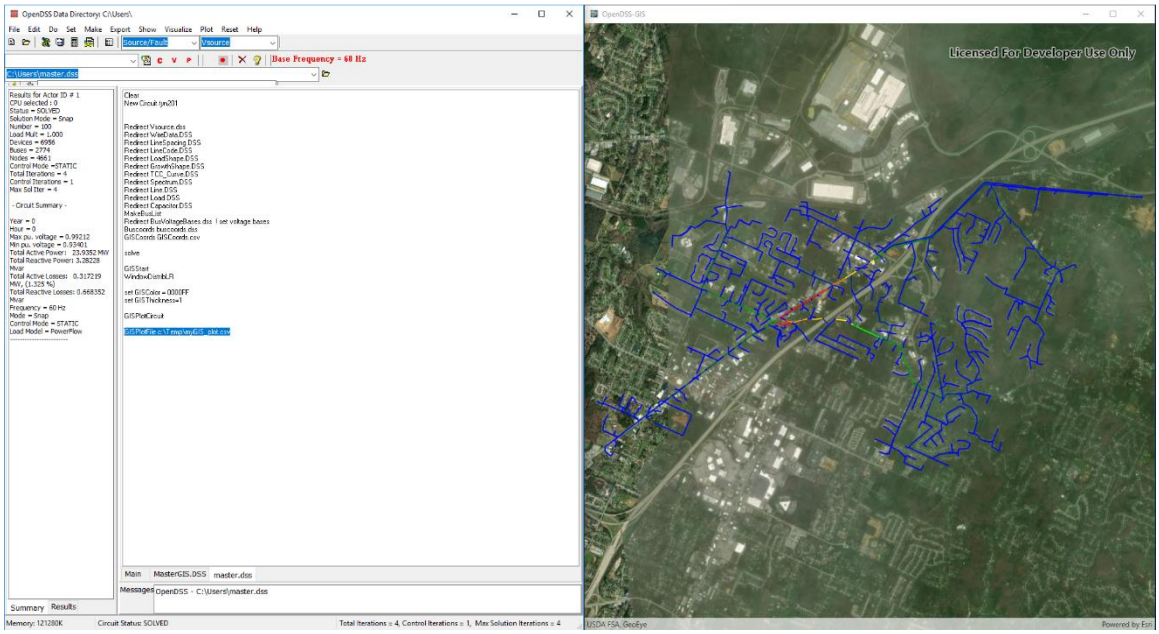



**Figure 27. Plotting a costumized plot on the map using files**

## *PlotPoint*

With this command the user can plot a mark on the map. The name of the shape for the mark must be specified in the argument and can be one of the following:

1. Circle
2. Cross
3. Diamond
4. Square
5. Triangle
6. X

The size of the mark can be specified in pixels using the option GISThickness, the color of the mark can be specified using the option GISColor. By default, the size is 3 pix and color red (*0xFF0000*).

For example, assume that the user wants to plot a blue triangle at -118.59292195047032692, 34.22088014327628969, then, the OpenDSS code will be as follows:

```
set GISCoords = [-118.59292195047032692,34.22088014327628969]
set GISThickness = 15 GISColor=0000FF
GIS PlotPoint triangle
```

### *PlotPoints*

With this command the user can plot a customized set of shapes on top of the map. The path to the file containing the plot is the argument for this command. The file contains the coordinates, shape, color and thickness for each mark, comma separated values as follows:

```
…
myLong1,myLat1,myShape1,myColor1,mySize1
myLong2,myLat2,myShape2,myColor2,mySize2
myLong3,myLat3,myShape3,myColor3,mySize3
…
```

Where `myLongX` and `myLatX` are longitude and latitude where the mark will be placed. myShapeX is an integer representing the shape that the user wants to draw, it can be one of the following:

0. Circle
1. Cross
2. Diamond
3. Square
4. Triangle
5. X

`myColorX` is the color of the mark and `mySizeX` is the size expressed in pixels. X refers to the index for differenciating data within the given example. At Figure 28, there is an example for plotting the power flow across the feeder using red as the for the lines with maximum flow and a transition to blue passing thorugh green to display the lines with lower power flow.

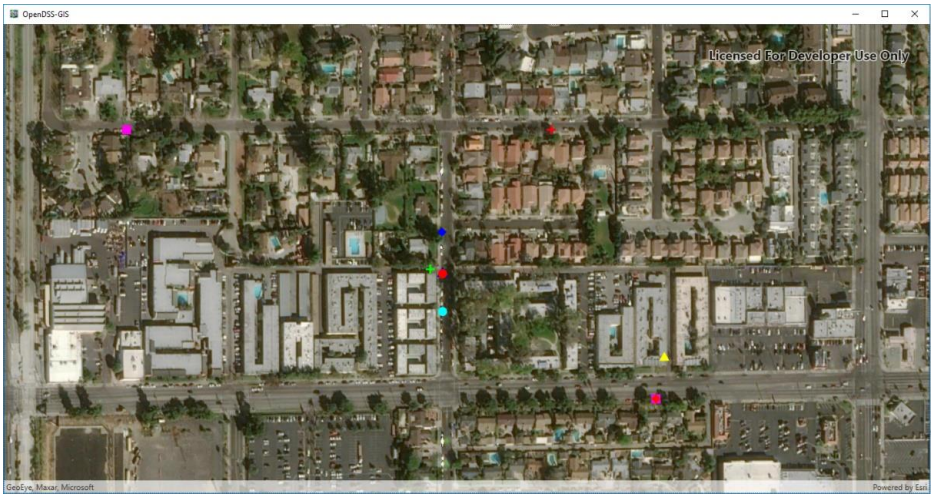



**Figure 28. Plotting a costumized set of marks on the map using files**

```
GIS PlotPoints C:\myPointsFile.csv
```

The content of the file plotted at Figure 28 is as follows:

```
-118.59292195047032692,34.22088014327628969,0,FF0000,13
-118.59304844439994042,34.22092078111747071,1,00FF00,13
-118.59292235908066004,34.22123360816988225,2,0000FF,13
-118.59071510141255601,34.21981100373600526,3,FF00FF,13
-118.59062557916303149,34.22016247763471597,4,FFFF00,13
-118.59292209798299211,34.22055454182444834,0,00FFFF,13
-118.5917951321892474,34.22211120247868621,1,FF000F,13
-118.59071510141255601,34.21981100373600526,2,FF0F00,13
-118.59619816989912522,34.22211095793186786,3,FF00F0,13
```

### *ClearMap*

If at certain point the user wants to clear all the plots drawn on top of the map to keep the map only, use this command. It will remove all the previous plots from OpenDSS-GIS leaving only the active map view.

```
GIS ClearMap
```

### *ShowBus*

It will zoom in the map to show the requested bus on the map. The argument for this command is the name of the bus to display as follows:

```
GIS ShowBus myBusName
```

### *ShowLine*

It will zoom in the map to show the requested line on the map. The argument for this command is the name of the line to display as follows:

```
GIS ShowLine myLineName
```

### *DrawLine*

It plots a line between the coordiantes defined at the option *GISCoords*. This command doesn't have arguments, but it requires the user to previously define the coordinates for the line as follows:

```
set GISCoords=[myLong1,myLat1,myLong2,myLat2]
```

Where `myLongX` and `myLatX` are longitude and latitude for describing the beginning and end of the line as shown in Figure 29. In Figure 29 we are drawing a green line (4 pix thick) at the same location of a Line to highlight it.

```
GIS DrawLine
```

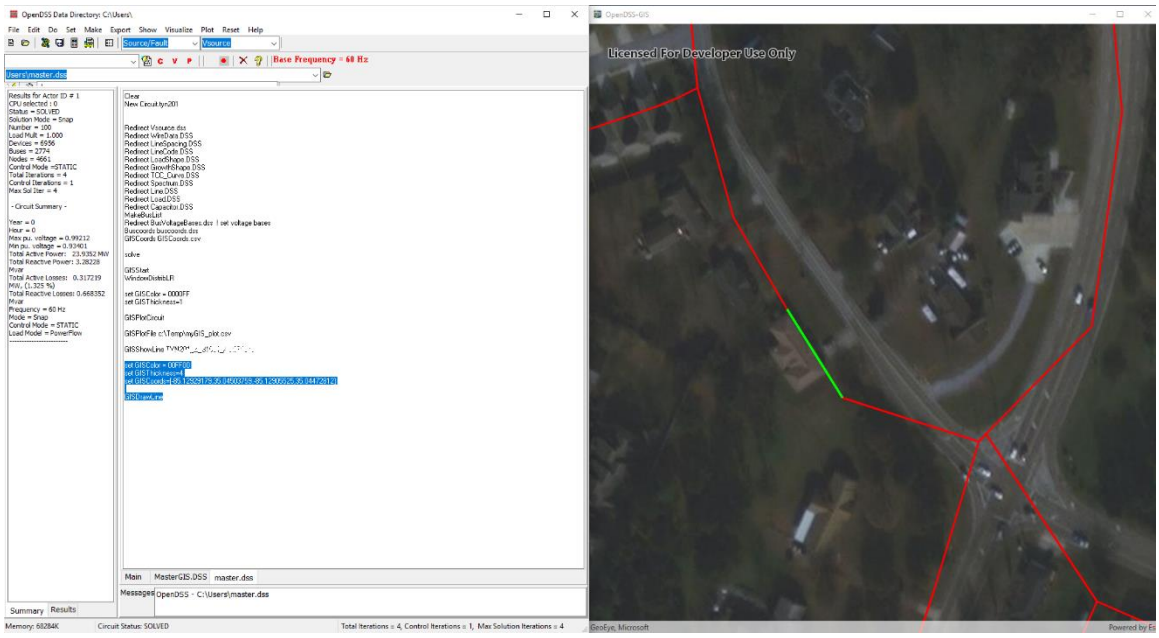



**Figure 29. Drawing a green line within the map**

## *ExportMap*

Exports the current map view to a PNG file within the model's folder. The name of the file will be *myMap_xxxxxxxxxxxxxxxxx.png*, where *xxxxxxxxxxxxxxxxx* is a time stamp for avoiding overwriting the same image.

```
GIS ExportMap
```

## *WindowSize*

Resizes the OpenDSS-GIS window using the coordinates given by the user. The coordiantes need to be given as: Left, Top, Right, Bottom. For example:

```
GIS WindowSize 0 0 800 800
```

Use this command to resize and position the window of OpenDSS-GIS as required within the screen.

## *WindowDistribLR/ WindowDistribRL*

This automated functionality allows users to redistribute OpenDSS and OpenDSS-GIS windows across the screen. With *WindowDistribLR*, OpenDSS will divide the screen in the half (horizontaly) and will locate OpenDSS to the left, leaving OpenDSS-GIS to the right as shown at

**Figure 30**. Otherwise, *WindowDistribRL* puts OpenDSS to the right of the screen and OpenDSS-GIS to the left as shown at Figure 31.

```
GIS WindowDistribLR

GIS WindowDistribRL
```

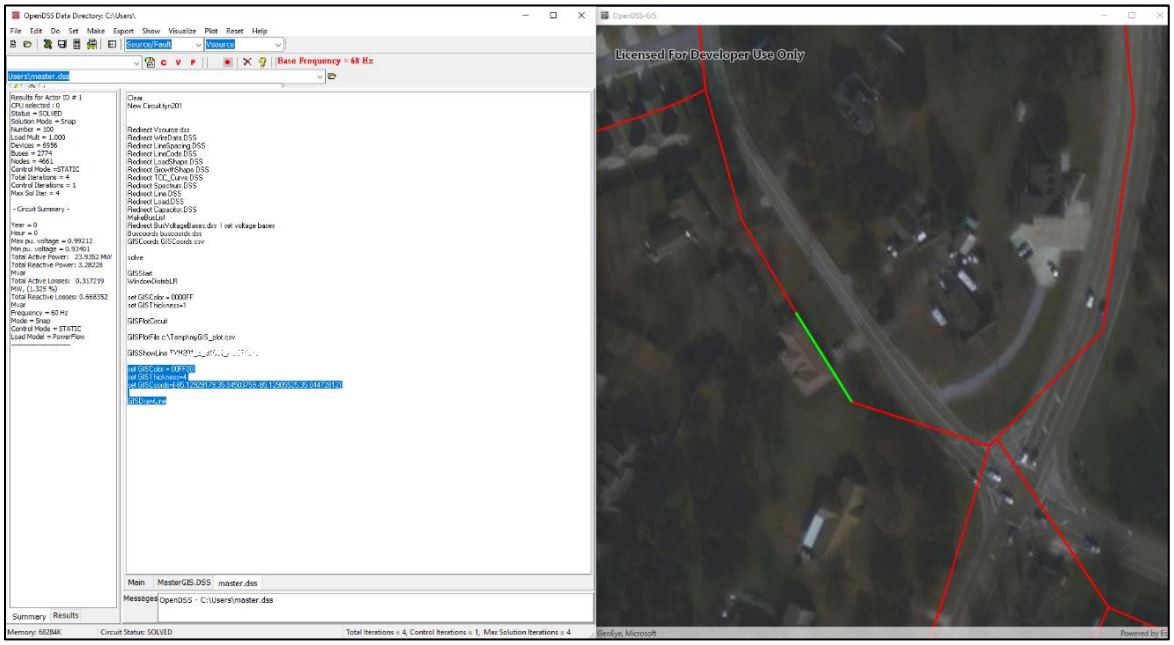



**Figure 30. WindowDistribLR**

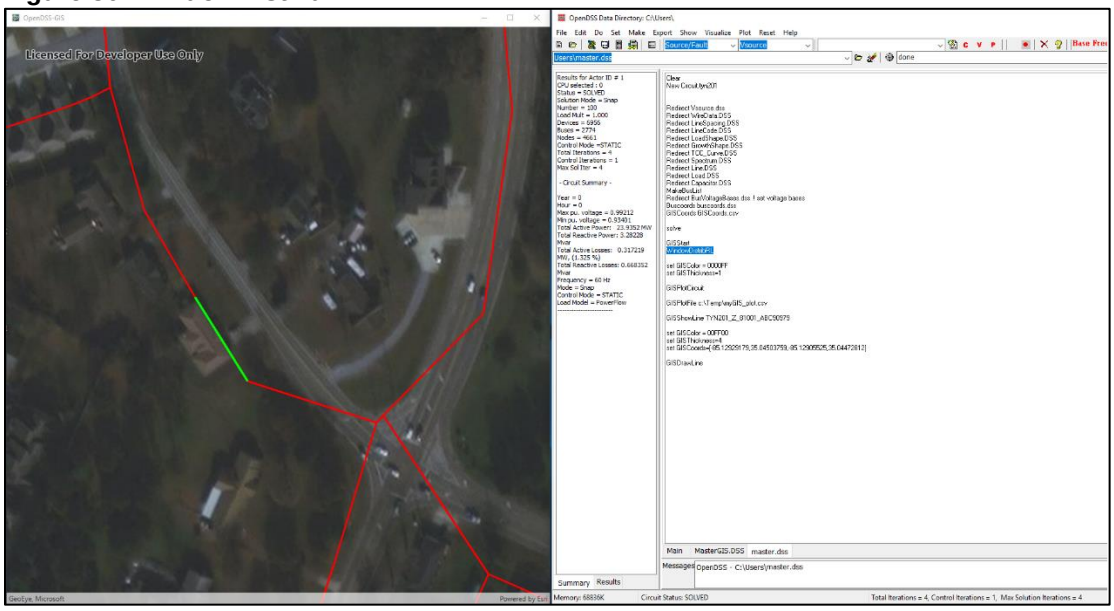



**Figure 31. WindowDistribRL**

## *ZoomMap*

Zoom the map in the area described with the coordiantes defined at the option *GISCoords*. This command doesn't have arguments, but it requires the user to previously define the coordinates for the area of interest as follows:

```
set GISCoords=[myLong1,myLat1,myLong2,myLat2]
```

`myLong1` and `myLat1` represent the top left corner (longitude and latitude) of the rectangle describing the area of interest. `myLong2` and `myLat2` are the bottom right corner of the area

of interest.

```
GIS ZoomMap
```

### *MapView*

Updates the map view in OpenDSS-GIS using one of the following as argument:

- Streets
- StreetsVector
- StreetsNight
- Satellite
- SatelliteLabels
- SatelliteLabelsVector
- DarkGrayCanvas
- LightGrayCanvas
- LightGrayCanvasVector
- Navigation
- OpenStreetMap

For example, to update the map view to the streetsNigth view use GISMapView as follows:

```
GIS MapView StreetsNight
```

Then, OpenDSS-GIS will change the map view as shown in Figure 32.


**Figure 32. StreetsNight map view**

Use the multiple arguments provided above to change the map view for highlighting different GIS aspects of the terrain, adding/removing labels, among others.

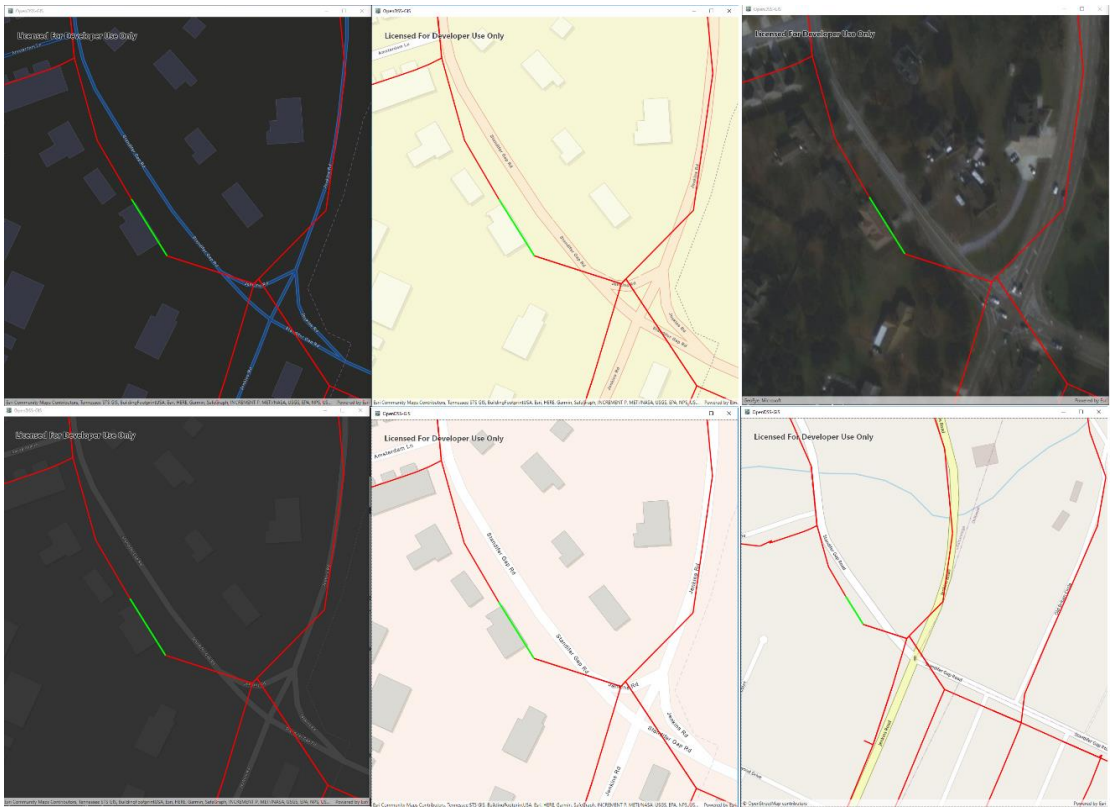



**Figure 33. Map views in OpenDSS-GIS**

### *GoTo*

Shows the location in the map at given the coordinates. The coordiantes must be defined using *GISCoords*. This command doesn't have arguments, but it requires the user to previously define the coordinates for the area of interest as follows:

```
set GISCoords = [35.922115, -84.142100]
GIS GoTo
```

The following conditions need to be fulfilled:

1. OpenDSS-GIS must be installed
2. OpenDSS-GIS must be initialized (use GIS Start command)

### *Distance*

Returns the linear distance in meters between two coordinates. The coordiantes must be defined using *GISCoords*. This command doesn't have arguments, but it requires the user to previously define the coordinates for the area of interest as follows:

```
set GISCoords = [40.50007607834282197,-80.01294794446681635,
40.50019376002214955, -80.01323963543782725]
GIS Distance
```

When combined with the command *LoadBus*, if the first coordiantes in the previous example correspond to myBus1 and the second pair to myBus2, GISDistance will look as follows:

```
GIS LoadBus myBus1 !uploads bus1 coords into the buffer
GIS LoadBus myBus2 !uploads bus2 coords into the buffer
GIS Distance
```

The following conditions need to be fulfilled:

1. OpenDSS-GIS must be installed
2. OpenDSS-GIS must be initialized (use GIS Start command)

### *LoadBus*

Uploads the coordinates of the bus specified in the argument to the coordinates buffer (*GISCoords*) by pushing the previous down. The coordinates buffer has 4 positions, the coordinates of the bus specified will be at the first 2 positions. This functionality can be useful for bringing back GIS coordinates of a bus or a pair of buses, and in combination with other commands, can facilitate the command execution.

For example, consider the example given at *DrawLine*, in this example for drawing a line on the map it is necessary to specify the coordinates of both ends of the line. Assum that the first pair of coordinates correspond to bus *myBus1* and the second pair to bu *myBus2*. Then, drawing the line using *GISLoadBus* will be as follows:

```
GIS LoadBus myBus1 !uploads bus1 coords into the buffer
GIS LoadBus myBus2 !uploads bus2 coords into the buffer
GIS DrawLine       !draws the line
```

In combination with *GoTo*, GISLoadBus has the same effect. Consider the example given at *GoTo*, assume that the coordinates given correspond to bus *myBus*, then, using GISLoadBus the example will be as follows:

```
GIS LoadBus myBus
GIS GoTo
```

Finally, suppose the user wants to draw a blue X at the bus location, the list of commands including *PlotPoint* will be as follows:

```
GIS LoadBus myBus
set GISThickness = 15 GISColor=0000FF
GIS PlotPoint x
```

The following conditions need to be fulfilled:

1. OpenDSS-GIS must be installed
2. OpenDSS-GIS must be initialized (use GIS Start command)

### *ShowCoords*

Returns the content of the coordinates buffer. The following conditions need to be fulfilled:

1. OpenDSS-GIS must be installed
2. OpenDSS-GIS must be initialized (use GIS Start command)

3. The model needs to have the correct GISCoords file

### *BatchFormat*

Formats the coordinates within the file specified. The first argument indicates the original format and the second argument the destination format. The third argument is the path to the source file containing the coordinates, which should be organized in 2 columns comma separated. The format can be one of the following:

- LatLong (latitude, Longitude - WGS84))
- DMS (Degrees, minutes, seconds)
- UTM (Universal Transverse Mercator)
- USNG

The data needs to be entered in the file following a strict convention:

LatLong: Latitude|Longitude. E.g. 36.28711454|-83.49819683
DMS: DegreeMinuteSecondNDegreeMinuteSecondW. E.g. 361713.6N0832953.5W
UTM: ZoneLatitudeLongitude. E.g. 17N2756404018690
USNG: ZoneCoords. E.g. 17SKA75641869
For example, consider a file containing the following coordiantes in Latitude-Longitude format (WGS84):

```
36.28711454|-83.49819683
36.30546503|-83.43104901
36.30477455|-83.43186156
36.30477455|-83.43186156
36.30643529|-83.4312484
```

The command for converting this file to UTM format, located at C:\Temp\myFile.txt, will be as follows:

```
GIS BatchFormat latlong utm C:\Temp\myFile.txt
```

The converted file will be saved as myFileutm.txt at the same folder and will look as follows:

```
17N 275640 4018690
17N 281723 4020572
17N 281648 4020497
17N 281648 4020497
17N 281708 4020680
```

The following conditions need to be fulfilled:

1. OpenDSS-GIS must be installed
2. OpenDSS-GIS must be initialized (use GIS Start command)
3. The model needs to have the correct GISCoords file

### *Format*

Formats the coordinates located at the first 2 places of the coordiantes buffer. The first argument indicates the original format and the second argument the destination format. The format can be

one of the following:

- LatLong (latitude, Longitude - WGS84))
- DMS (Degrees, minutes, seconds)
- UTM (Universal Transverse Mercator)
- USNG

The data needs to be entered in the file following a strict convention:

LatLong: Latitude|Longitude. E.g. 36.28711454|-83.49819683
DMS: DegreeMinuteSecondNDegreeMinuteSecondW. E.g. 361713.6N0832953.5W
UTM: ZoneLatitudeLongitude. E.g. 17N2756404018690
USNG: ZoneCoords. E.g. 17SKA75641869

For example, consider converting the coordinates 35.07053683, -85.15665416 (latitude, longitude WGS84) to UTM format, then the command will be as follows:

```
GIS Format latlong utm 35.07053683|-85.15665416
```

The output will be displayed at the results tab of the executable panel in OpenDSS.
The following conditions need to be fulfilled:

1. OpenDSS-GIS must be installed
2. OpenDSS-GIS must be initialized (use GIS Start command)
3. The model needs to have the correct GISCoords file

## *Select*

This command enables the user to draw a reactagle on the map. Once this command is executed, the user can go to OpenDSS-GIS and start drawing the rectangle by clicking on the map and dragging the cursor across as shown at Figure 34.

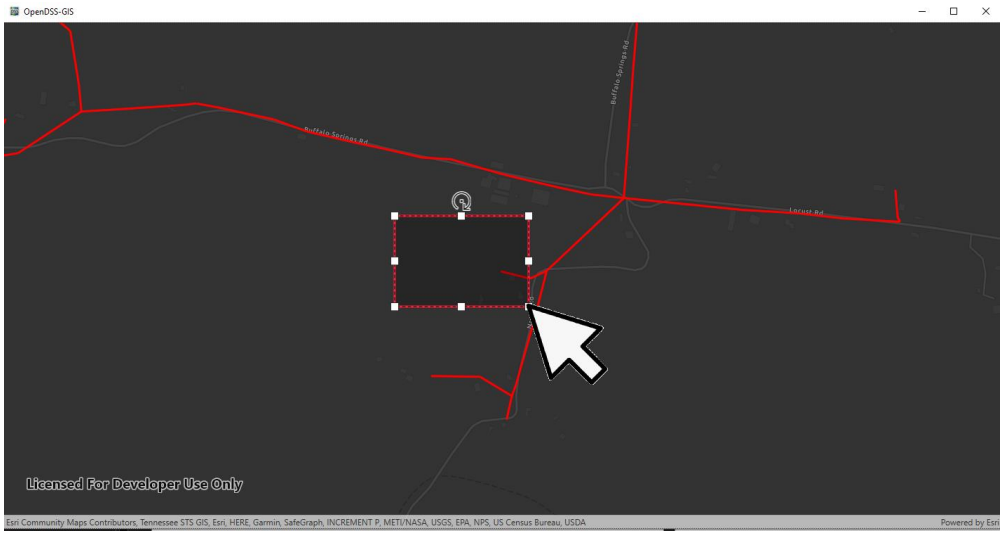



**Figure 34. User drawing a rectangle over the map**

For drawing the rectangle first click on the starting point, drag the cursor until reaching the desire size/area and then, release the mouse button. After drawn, the rectangle shape/dimensions can be edited by selecting one of the editing squares around the shape. At this point, the shapes features have not been saved and require the execution of the command *StopSelect* for storing the reactangle coordinates in memory.

The following conditions need to be fulfilled:

1. OpenDSS-GIS must be installed
2. OpenDSS-GIS must be initialized (use GIS Start command)
3. The model needs to have the correct GISCoords file

### *StopSelect*

This command terminates the select command started with *Select*. It also removes the latest shape drawn by the user (rectangle) from the map and stores its coordinates in memory for further queries.

### *GetSelect*

This command returns the coordiantes of the rectangle drawn by the user after using the command *StopSelect*. The coordiantes will be returned in WGS 84 format in a comma separated string describing the min longitude, min latitude, max longitude, max latitude. For example:

```
-83.521466956175,36.211375646857,-83.5181546602918,36.213180627696,
```

### *DrawLines*

This command enables the user to draw a polyline on the map. Once this command is executed, the user can go to OpenDSS-GIS and start drawing the polyline by clicking on the map at the points where the polyline edges will be, this will update the map dynamically drawing the polyline as shown at Figure 35.

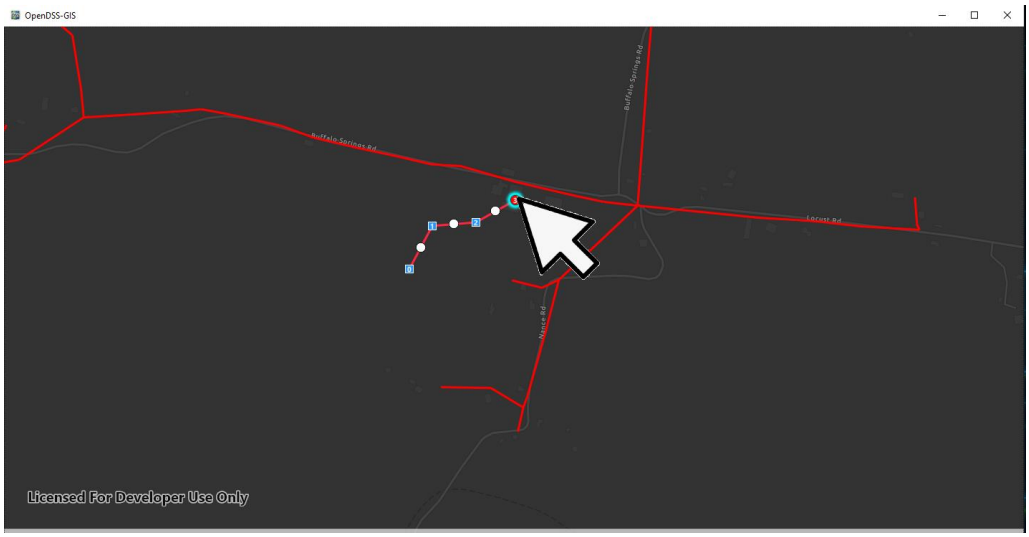



**Figure 35. User sketching polylines on the map**

After drawing the polyline, the size and coordinates the the polyline can be edited by using the white circles across the polyline to alter the polyline topology as shown at Figure 36. Once the

draw is finalized, the user needs to execute the command *StopDraw* to save the polyline coordinates in memory for futher queries.

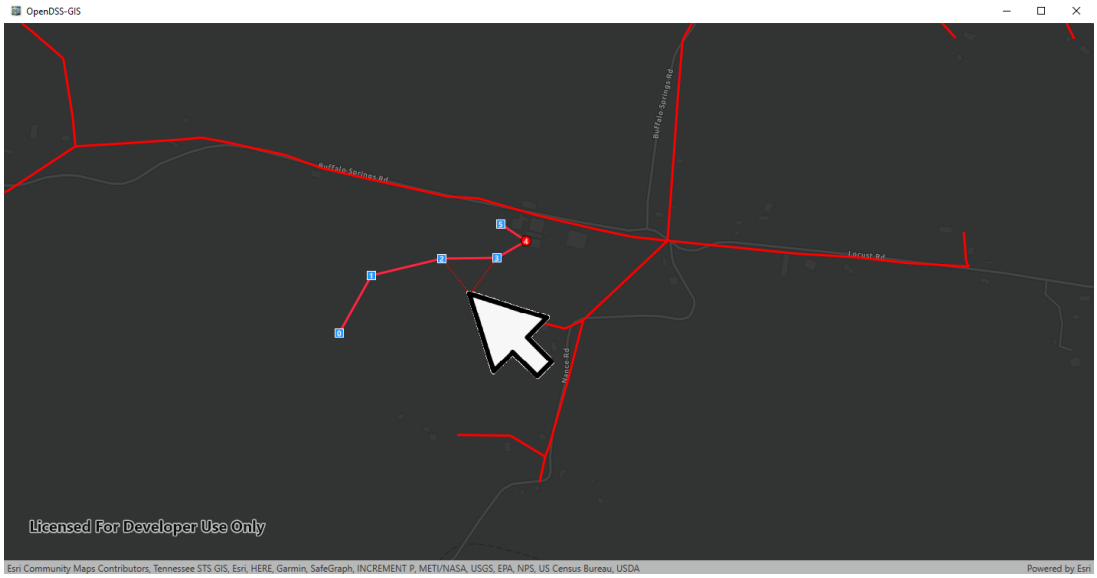



**Figure 36. Altering the polyline topology**

### *StopDraw*

This command terminates the DrawLines command started with *DrawLines*. It also removes the latest shape drawn by the user (polyline) from the map and stores its coordinates in memory for further queries.
The following conditions need to be fulfilled:

1. OpenDSS-GIS must be installed
2. OpenDSS-GIS must be initialized (use GIS Start command)

### *GetPolyline*

This command returns the coordiantes of the polyline drawn by the user after using the command *StopDraw*. The coordiantes will be returned in WGS 84 format in a JSON string containing all the edges describing the form. For example:

```
{paths:[[[-83.523235799526304,36.211851362079948],
[-83.522507458110596,36.212900759076554],
[-83.520912020397404,36.21320859009181],
[-83.520270368033351,36.212564972457137],
[-83.519663414327567,36.213222591374503],
[-83.519004431288977,36.213530413962665],
[-83.519576712729815,36.21383822592572]]],
spatialReference:{wkid:4326}}
```

The following conditions need to be fulfilled:

1. OpenDSS-GIS must be installed
2. OpenDSS-GIS must be initialized (use GIS Start command)

### *GetAddress*

Returns the address at the given coordinates. The coordiantes must be defined using *GISCoords*, *LoadBus* can also be used with this command. This command doesn't have arguments, but it requires the user to previously define the coordinates for the point of interest as follows:

```
set GISCoords = [35.92214527098195, -84.14210580168506]
GIS GetAddress
```

The answer from OpenDSSiGIS is a JSON string describing the address details. Using the commands above, the response from OpenDSS-GIS will be as follows:


```
{   "address": {     "Match_addr": "900-998 Corridor Park Blvd,
Knoxville, Tennessee, 37932",   "LongLabel": "900-998 Corridor Park Blvd,
Knoxville, TN, 37932, USA",   "ShortLabel": "900-998 Corridor Park Blvd",
"Addr_type": "StreetAddress",   "Type": "",   "PlaceName": "",   "AddNum":
"970",   "Address": "970 Corridor Park Blvd",   "Block": "",   "Sector":
"",    "Neighborhood": "",    "District": "",    "City": "Knoxville",
"MetroArea": "",    "Subregion": "Knox County",    "Region": "Tennessee",
"Territory": "",   "Postal": "37932",   "PostalExt": "",   "CountryCode":
"USA"    },    "location": {      "x": -84.14173410620927,      "y":
35.921630248787075,      "spatialReference": {         "wkid": 4326,
"latestWkid": 4326   }  } }
```


The following conditions need to be fulfilled:

1. OpenDSS-GIS must be installed
2. OpenDSS-GIS must be initialized (use GIS Start command)

### *Text*

Prints the text given at the argument at the coordinates specified in *GISCoords*. The coordiantes must be defined using *GISCoords*, *LoadBus* can also be used with this command. The text color will match the color defined at *GISColor*, and the font size must be specified at *GISThickness*. This command requires the user to previously define the coordinates for the point of interest, the text color and size (if desired) as follows:

```
set GISCoords = [35.92209179726962,-84.14211464700368]
set GISColor = FF0000
set GISThickness=20
GIS Text "Electric Power Research Institute (EPRI)"
GIS GoTo
```

The previous example will show the location displayed at Figure 37, the text will be overimposed at the location.


**Figure 37. Writing text on the map**

### *TextFromFile*

Prints the text within a file given in the argument on the map. The text features as coordiantes, color and font size are specified within the file. The command must be used as follows:

```
GIS TextFromFile C:\MyFilepath\myGISText.txt
```

The file extension is not relevant. The file's content must follow the following order:

```
…
myLong1,myLat1,myText1,myColor1,myFontSize1
myLong2,myLat2,myText2,myColor2,myFontSize2
myLong3,myLat3,myText3,myColor3,myFontSize3
…
```

For example, assume that the content of the file mentioned in the example above is as follows:

```
35.92209179726962,-84.14211464700368,my Location 1,FF0000,20
35.90779395922772,-84.14752180168537,my Location 2,FFFF00,14
```

The result in OpenDSS-GIS will be as shown in Figure 38.

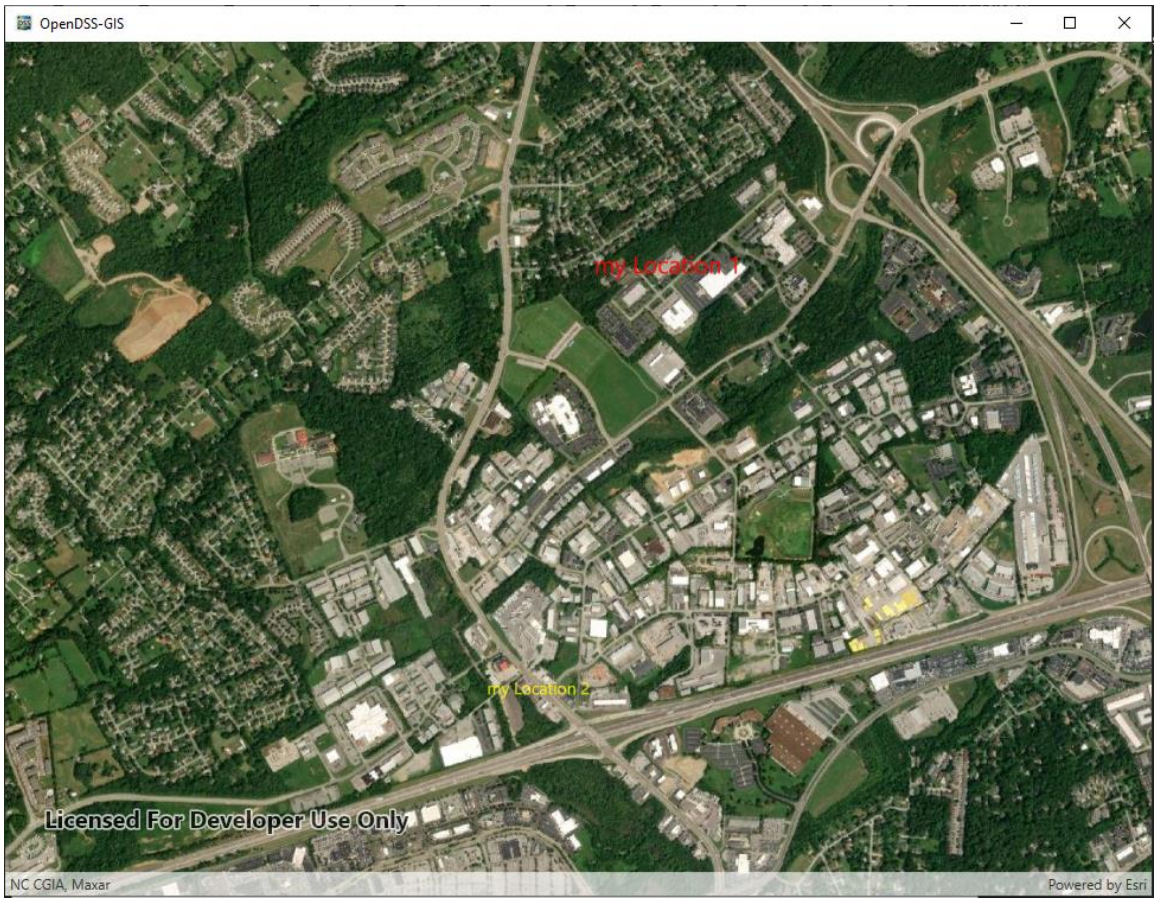



**Figure 38. Printing text from a file on the map**

# General OpenDSS Object Property Descriptions

General Object are objects common to all circuits in the OpenDSS. Any circuit can reference the data contained in the General DSS Objects and they are treated like libraries of common data.

The following describes several object classes 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.

Note that command lines are parsed and executed from left to right. You can use a General class object to provide the base definition and override specific properties by setting them after (to the right of) referencing the General class object like LineCode, for example.

## LINECODE

LineCode objects are general library objects that contain impedance characteristics for lines and cables. The term "line code" is an old distribution system analysis program term that simply refers to a code that was made up by programmers to describe the impedances of a line construction. In most distribution analysis programs, one can describe a line by its LineCode and its length. LineCode objects were defined in a separate file. This collection of objects emulates the old linecode files, except that the concept here is a little 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 in the Line object definition and one does not need to use a LineCode object, although the linecode will be more convenient most of the time and would be easier to read. There may be hundreds of Line objects, but only a few different kinds of line constructions.

A LineCode object definition can also perform a Kron reduction, reducing out the last conductor in the impedance matrices, which would then be assumed to be a neutral conductor, with zero volts at each end (i.e., grounded). This applies only if the impedance is specified as a matrix. If the impedance is defined using symmetrical components, this function does not apply because symmetrical component values already assume the reduction.

This assumes the impedance matrix is constructed as follows, which is typical for power system analysis:

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

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.)

By specifying the values of Rg, Xg, and rho properties used to compute the impedances supplied, OpenDSS 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 use the LineGeometry class and let OpenDSS compute the impedance matrices. Or you can insert a frequency-dependent Reactor model in series with the line to accound for the variation of impedance with frequency.

The LineCode properties, 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. See description above. For making better adjustments of line impedance values for frequency for harmonics studies. Default= 0.01805 ohms per 1000 ft at 60 Hz. If you do not wish to adjust the earth return impedance for frequency, set both Rg and Xg to zero. Generally avoid Kron reduction if you will be solving at frequencies other than the base frequency and wish to adjust the earth return impedance.

**Xg**= Carson earth return reactance per unit length used to compute impedance values at base frequency. See description above. For making better adjustments of line impedance values for frequency for harmonics studies. Default= 0.155081 ohms per 1000 ft at 60 Hz. If you do not wish to adjust the earth return impedance for frequency, set both Rg and Xg to zero. Generally avoid Kron reduction if you will be solving at frequencies other than the base frequency and wish to adjust the earth return impedance.

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

**Neutral =** Designates which conductor is the "neutral" conductor that will be eliminated by Kron reduction. Default is the last conductor (nphases value). After Kron reduction this property is set to 0. Subsequent issuing of Kron=Yes will not do anything until this property is set to a legitimate value. Applies only to LineCodes defined by R, X, and C matrix.

**B1, B0 =** Alternate way to specify C1 and C0. Micro-siemens per unit length

**Seasons=** Defines the number of ratings to be defined for the wire, to be used only when defining seasonal ratings using the "Ratings" property.

**Ratings=** An array of ratings to be used when the seasonal ratings flag is True. It can be used to insert multiple ratings to change during a QSTS simulation to evaluate different ratings in lines.

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

## LINEGEOMETRY

This class of data is used to define the conductors and positions of the conductors in a type of line construction.

**Nconds**= Number of conductors in this geometry. Default is 3. Triggers memory allocations. So, define this first!

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

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

**Wire**= Code for a WireData-class object. MUST BE PREVIOUSLY DEFINED. no default.

**X**= x coordinate. This is a relative coordinate value and can have an arbitrary reference point. For convenience, one of the conductors is uaually assigned the X=0 position.

**H**= Height of conductor above earth.

**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 from Nconds to Nphases by Kron Reduction. Reduce out neutral conductors assumed to be grounded.

**Spacing=** Reference to a LineSpacing object for use in a line constants calculation. Alternative to x, h, and units. MUST BE PREVIOUSLY DEFINED. Must match "nconds" as previously defined for this geometry. Must be used in conjunction with the **Wires** property**.**

**Wires=** Array of WireData names for use in a line constants calculation. Alternative to individual wire inputs. ALL MUST BE PREVIOUSLY DEFINED. Must match "nconds" as previously defined for this geometry, unless TSData or CNData were previously assigned to phases, and these wires are neutrals. Must be used in conjunction with the **Spacing** property.

**CNcable=** Code from CNData. MUST BE PREVIOUSLY DEFINED. no default. Specifies use of Concentric Neutral cable parameter calculation.

**TSCable=** Code from TSData. MUST BE PREVIOUSLY DEFINED. no default. Specifies use of Tape Shield cable parameter calculation.

**CNCables**= Array of CNData names for cable parameter calculation. All must be previously defined, and match "nphases" for this geometry. You can later define "nconds-nphases" wires for bare neutral conductors.

**TSCables=** Array of TSData names for cable parameter calculation. All must be previously defined, and match "nphases" for this geometry. You can later define "nconds-nphases" wires for bare neutral conductors.

**Seasons=** Defines the number of ratings to be defined for the wire, to be used only when defining seasonal ratings using the "Ratings" property.

**Ratings=** An array of ratings to be used when the seasonal ratings flag is True. It can be used to insert multiple ratings to change during a QSTS simulation to evaluate different ratings in lines.

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

New Capacitor.C2 like=c1 ...

### *Line Constants 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.
```

## LINESPACING

A LineSpacing object defined the spacing of a group of conductors in a relative coordinate system. These are usually the main phase conductors, but could be any group of conductors such as conductors in a bundle. Simply define the position of each conductor. The Spacing property in LineGeometry and Line object definitions points to a LineSpacing object.

**Nconds**= Number of conductors (wires) in this LineSpacing object. Default = 3. This triggers the memory allocations, so define this propery first!

**Nphases**= Number of retained phase conductors. If less than the number of wires, list the retained phase coordinates first in the following arrays.

**x**= Array of x coordinates for each wire.

**h=** Array of wire heights corresponding to x array.

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

## LOADSHAPE

A Loadshape object is very important for all types of sequential-time power flow solutions. This is a very powerful capability of OpenDSS and users should become familiar with defining and using them. A LoadShape object consists of a series of multipliers, typically 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, in 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 properties 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, define the interval as 0.0.

**mInterval**= Specify Interval in minutes.

**sInterval**= Specify Interval in seconds.

**Pmult or Mult**= (Mult was the original name of this property)

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:

```
PMult=[file=myfile.txt]
Mult=[dblfile=myfile.dbl]
PMult=[sngfile=myfile.sng]
!for multicolumn CSV files
mult = (file=MyCSVFile.CSV, col=3, header=yes)
```

This syntax will work on almost any property of any class of element in the OpenDSS where an array of numbers is expected. The addition of the capability to handle the two packed binary file types should significantly speed up the importing of AMI data into Loadshape objects. (see UseActual property) Also, the way the Mean and Std

Deviation calculation is done has been changed to delay calculation until these values are actually needed. This should minimize the computing overhead on reading loadshapes back in.

**QMult**= Array of multiplier values. Same property rules as the **PMult** (or **Mult**) property. If specified, the multiplier is applied to the Load (or Generator) kvar value. If omitted, the value of **PMult** is applied to the kvar value.

**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. In prior versions, the mean and standard deviation were always computed after an array of points are entered or normalized (see below). From version 7.3, the mean and standard deviation are computed only if requested to save time when processing many AMI curves. 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 properties 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 *active power* 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 active power 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 active power 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.

**PQCSVFile=** Switch input to a CSV text file containing both active and reactive power (P, Q) multiplier pairs, one per row.

If the interval=0, there should be 3 items on each line: (hour, Pmult, Qmult)

**Action**= {**Normalize** | **DblSave** | **SngSave**} After defining load curve data, setting action=normalize will modify the multipliers so that the peak is 1.0. The mean and std deviation are recomputed. Setting action=**DblSave** or **SngSave** will cause the present **mult** and **qmult** values to be written to either a packed file of double or single, respectively. The filename is the loadshape name. The **mult** array will have a "_P" appended on the file name and the **qmult** array, if it exists, will have "_Q" appended.

**UseActual=** {Yes | No* | True | False*} If true, signals to Load, Generator, or other objects to use the value of the multiplier as the actual kW, kvar value rather than a multiplier. Nominally for entering AMI data. Do not use Action=Normalize with UseActual=Yes.

**Pmax, Qmax=** If you define the LoadShape object with UseActual=Yes, when you define any of the Duty, Daily, or Yearly properties of a load or generator, the kW property is redefined to the kW and kvar values at the time of the peak kW in the loadshape. This will be the value used for the initial Snapshot solution. If you define more than one loadshape object, the last one overrides any previous definition, as with all OpenDSS properties. You can query the Pmax and Qmax properties of the Loadshape object to see what was computed. Keep in mind that the Qmax value is the value of kvar at the time of maximum kW, if defined using the Qmult property. If you want something different, you may override the computed values by setting either or both of these properties after reading in the loadshapes. If you want something else, you will need to redefine kW and kvar (or kW, PF) after you set the Duty, Daily, or Yearly properties.

**Pbase=** Base P value for normalization. Default is zero, meaning the peak will be used.

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

**MemoryMapping**= {Yes | No* | True | False*} If true, enables the memory mapping method for handling the load shape. See *Memory mapping load shapes* for details.

## *Memory mapping load shapes*

A memory-mapped file is a file that contains the contents of a file in virtual memory. Through this mapping between a file and memory space, multiple processes can modify the file by reading and writing directly to memory without having to preprocess the file locally (https://docs.microsoft.com/en-us/dotnet/standard/io/memory-mapped-files).

Through this technique, which is provided by the Operating system, OpenDSS reduces the memory consumption by using a map to the file located in the hard drive to extract the required data when required. There are advantages and disadvantages when using this technique. By default, it is desactivated in OpenDSS.

It is expected that users dealing with a significant amount of large load shapes will benefit from memory nmapping. However, when using memory mapping the source file (the one containing the load shape data) needs to fullfil certain conditions depending on the format:

- ASCII files: If the source file is delivered in ASCII format (string), each line of the file (CSV, txt, etc.) each row needs to have the same amount of characters independently of the number of columns. For example:

```
0.455371653
0.477029779
0.51211543     <- This row has less characters
0.528053513
```

In the example above, the number of characters in line 3 is less than the previous ones, which will trigger an error message and cancel the creation of the load shape when declared. The lack of characters can be easily solved by adding zeros (0) to the line to make it match with the rest of the file as shown below.

```
0.455371653
0.477029779
0.512115430
0.528053513
```

The same condition applies to files containing multiple colmuns as shown in the example below.

```
0.477029779,0.477029779
0.51211543,0.5121154300    <- this line was completed with 00
0.528053513,0.528053513
0.541378127,0.541378127
```

- DBL and SNG files: These file types don't have any special requirement.

The load shape declaration in OpenDSS when using memory mapping requires to specify that the memory mapping feature will be used before declaring the multipliers. For example:

```
New Loadshape.LS_PhaseA npts=8760 interval=1 MemoryMapping=Yes
mult=(file=LS_Phase_AOK.txt)
```

There are also features that cannot be used from the LoadShape class while using memory mapping:

- Normalize: Load shapes cannot be normalized since they are not available in memory with this technique and are given READ_ONLY access since, it may have to share it with other instances (e. g. when using parallel processing).
- Since the load profile is not pre-processed, in case of defining an actual load shape (UseActual=Yes) the user must include Pmax and Qmax when declaring the load shape (Pmax=X Qmax=Y). These values will affect the outcome when performing snapshot simulations. It will not affect time-based simulations (daily, yearly, time, duty, etc.). define Pmax and Qmax if planning to do snapshot simulations when defining load shapes with memory mapping.

### *What to expect with memory mapping?*

When using memory mapping, the time required for uploading a model containing a significant amount of load shapes will be drastically reduced without compromising the simulation performance (depending on the source format). In a model with 2768 buses (3952 nodes) with 724 load shapes in SNG format, loading the model into memory without memory mapping takes about 157 seconds. Otherwise, by using memory mapping the loading time gets reduced to 760 ms as shown in Figure 39.

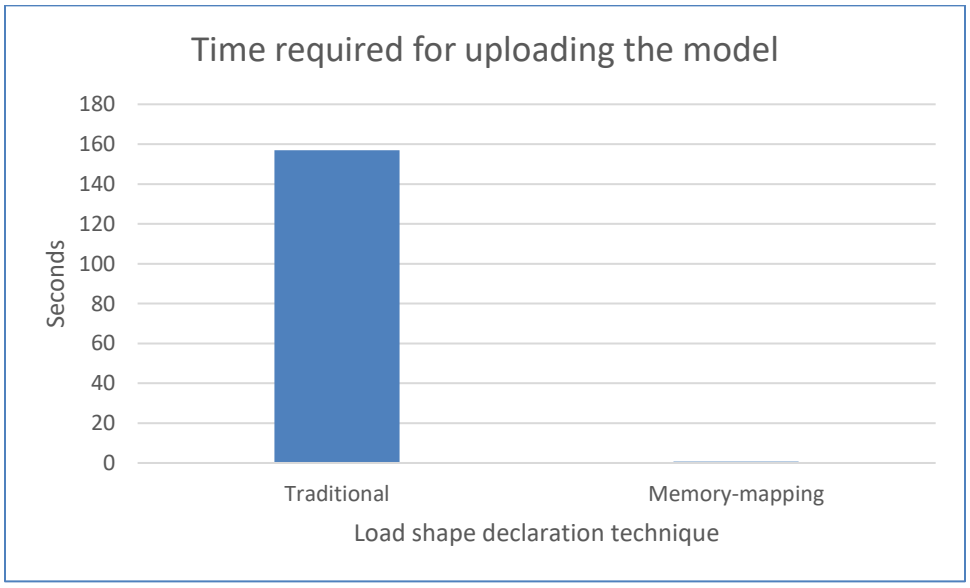



**Figure 39. Time required to compile a model**

In terms of performance, depending on the format of the source the simulation overhead will be

more noticeable. If the source file is in ASCII format (string) the simulation performance will be mostly affected due to the parsing process taking place during the simulation. On the other hand, DBL and SNG files will add little or none overhead to the simulation.

To demonstrate this concept, consider the example proposed at https://sourceforge.net/p/electricdss/code/HEAD/tree/trunk/Version8/Distrib/Examples/MemoryMappingLoadShapes/. In this example we are proposing different multiple ways for declaring the same load shape, one of the lines declares the model as an ASCII file, others as DBL and SNG, and other formats.

```
…
! These are different ways for defining the same load shape using memory
mapped files
New    Loadshape.LS_PhaseA    npts=8760    interval=1    MemoryMapping=Yes
mult=(file=LS_Phase_AOK.txt)
!New    Loadshape.LS_PhaseA    npts=8760    interval=1    MemoryMapping=Yes
mult=(dblfile=myDBL.dbl)
!New    Loadshape.LS_PhaseA    npts=8760    interval=1    MemoryMapping=Yes
mult=(sngfile=mySGL.sng)
…
```

Once the model is loaded into memory a yearly simulation is performed for each case and compared to the original simulation (with no memory mapping). The results are shown below.

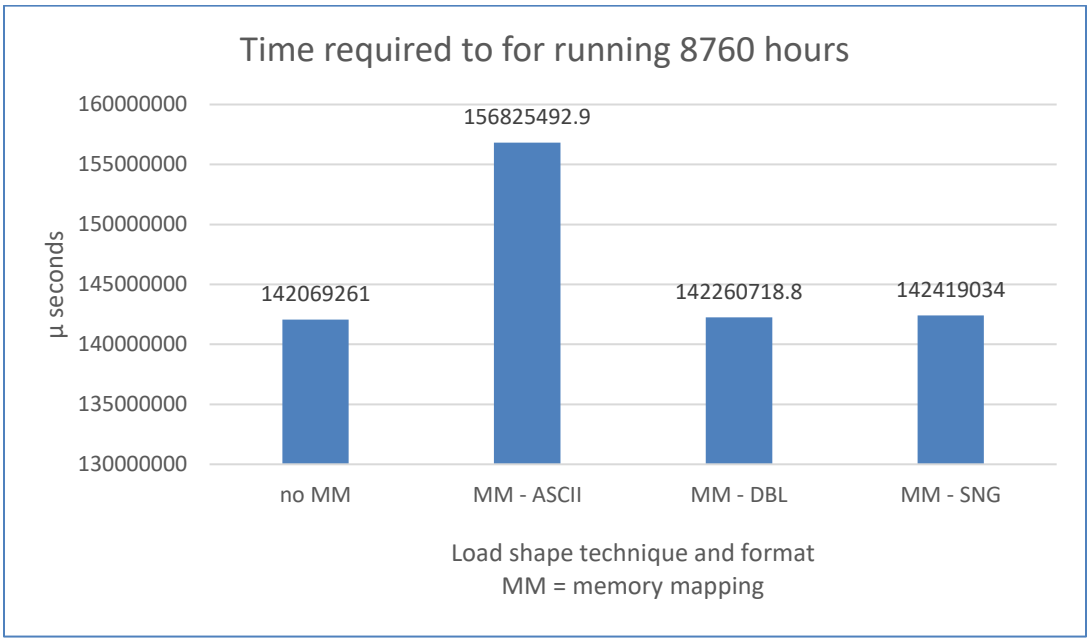



**Figure 40. Yearly QSTS simulation performance with different load shapes format when using memory mapping**

The performance is also tied to the hard drive technology in the host computer. SSD are the best choice in this case, while classic electro mechanic devices may add more overhead to the simulation.

## *Aggregating profiles in OpenDSS*

OpenDSS incorporates a functionality for aggregating profiles to reduce the memory required for simulating models with many load shapes linked to the model's loads. This could be the case when the model contains deatalied Ami data linked to every load in the model, which can represent a significant amount of memory if the load shapes contain high granularity data.

OpenDSS provides a command for aggregating the load profiles within the model by tearing the interconnected model into several sub-zones defined by the user. The tearing algorithm (MeTIS - http://glaros.dtc.umn.edu/gkhome/metis/metis/overview) objective is to balance the model partitioning trying to ensure that each zone will have the same number of buses. After tearing the model, OpenDSS will estimate an aggregate load profile for all the loads in each zone, assingning the new load profile to them.

Finally, OpenDSS will save the aggregated model into a folder called "Aggregated_model" within the original model's folder. The new model includes only the new aggregated load profiles, reducing the size of the model in the hard drive and when loaded into memory.

**Important:** Before using this functionality, make sure that all the loads in the model have a yearly load shape linked, otherwise, it may lead to an error message.

The steps required for using this functionality are: Define the number of zones, call the command specifying the use of the load shape multiplier (Actual/pu). Thiscan be achieve through the following commands:

```
set Num_SubCircuits=X
AggregateProfiles myUse
```

Where X is the number of zones in which the circuit will be torn and "myUse" is Actual or pu for enabling the flag UseActual when defining the new load shapes in the aggregated model. The following example aggregate profiles in pu using 10 zones:

```
Clear
New Circuit.ieee123

Redirect Vsource.dss
Redirect LineCode.DSS
Redirect LoadShape.DSS
Redirect GrowthShape.DSS
...
Redirect RegControl.DSS
Redirect EnergyMeter.DSS
MakeBusList
Redirect BusVoltageBases.dss  ! set voltage bases
Buscoords buscoords.dss
set maxcontroliter=20
solve

set Num_SubCircuits=10
AggregateProfiles
get LinkBranches
```

In the previous example, the line “get LinkBranches” returns the list of PDE that were used as the head of the zone after tearing the model into n-zones. On the other hand, the following example aggregates the load profiles and adds the property “UseActual=true” when creating the new aggregated model.

```
Clear
New Circuit.ieee123

Redirect Vsource.dss
Redirect LineCode.DSS
Redirect LoadShape.DSS
Redirect GrowthShape.DSS
...
Redirect RegControl.DSS
Redirect EnergyMeter.DSS
MakeBusList
Redirect BusVoltageBases.dss  ! set voltage bases
Buscoords buscoords.dss
set maxcontroliter=20
solve

set Num_SubCircuits=10
AggregateProfiles Actual
get LinkBranches
```

## GROWTHSHAPE

A GrowthShape object is like 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 properties 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.

## CNDATA, TSDATA

Data for Concentric Neutral (CN) and Tape Shield (TS) cables commonly used in power distribution systems.

See the file *TechNote CableModelling.pdf* in the Doc folder installed with the OpenDSS program.

## 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.

**Capradius-** Equivalent conductor radius for capacitor calcs. Specify this for **bundled** conductors. Defaults to same value as radius.

**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 of this class:

For bundled conductors, you define an equivalent conductor with

$$GMR_B = [N(GMR_C)A^{(N-1)}]^{1/N}$$

Where

N = number of conductors in the bundle

$GMR_C$ = GMR of each subconductor

A = bundle radius

For capacitive reactance calculations, the effective radius (**Capradius**) of the bundle is

$$r_B = (NrA^{N-1})^{1/N}$$

Where

N = number of conductors in the bundle

r = radius of each conductor

A = bundle radius

When the bundle size is specified by the bundle spacing (distance between adjacent conductors in the bundle, S,

$A = S / [2 \sin(\pi/N)]$ for N>1

A = 0 for N=1 (recall that $0^0 = 1$)

## XFMRCODE

Like LineCode for Line objects, you can define transformer objects and use them simply by referring to them in the Transformer object definition using the XfmrCode property. The XfmrCode properties generally have the same name and definition as the Transformer object. Refer to the Transformer object definition.

An example use of an XfmrCode to define a bank of 1-phase transformers to make up a 3-winding Y-D-Y connection:

```
// The Transformer
// Model as 3 1-phase transformers so we can see the delta currents

New XfmrCode.YDY1Phase phases=1 windings=3
~ conns=[wye  wye delta ]
~ kVs=[39.83, 7.62,  2.4]
~ kVAs=[5000 3333 1666]    ! 15 MVA 3-phase
~ XHL=7.5   XHT=36  XLT=28
~ %Rs = [0.11  0.11  0.275]

/*******************
To define the Transformers, only need to specify buses and any other
properties that are different.
*******************/

// Connect up the three 1-ph transformer in YYD
New Transformer.YDYA XfmrCode=YDY1Phase ppm=0   !<--------neglect ppm
~ Buses=[B2.1.0 B3.1.0 B5.1.2]

New Transformer.YDYB  XfmrCode=YDY1Phase  ppm=0
~ Buses=[B2.2.0 B3.2.0 B5.2.3]

New Transformer.YDYC  XfmrCode=YDY1Phase ppm=0
~ Buses=[B2.3.0 B3.3.0 B5.3.1]
```

See Help in the EXE version of the program for more General objects.

# DSS Circuit Element Object Descriptions

The following are descriptions of key DSS circuit elements. This section is by no means complete and often lags well behind implementations of new elements. However, 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.

# Sources and Other Objects

## VSOURCE OBJECT

Voltage source. A Vsource object is a **two-terminal**, multi-phase Thevenin (short circuit) equivalent. That is, it is a voltage source behind an impedance. The data are specified as it would commonly be for a power system source equivalent: Line-line voltage (kV) and short circuit MVA.

The most common way to use a voltage source object is with the first terminal connected to the bus of interest with the second terminal connected to ground (the voltage reference). In this usage, the connection of the second terminal may be omitted. The second terminal connection defaults to *BusName*.0.0.0 for a 3-phase source connected to *BusName*. In 2009, the voltage source was changed from a single-terminal device to a two-terminal device. This allows for the connection of a voltage source between two buses, which is convenient for some types of studies.

The properties are, in order:

**Bus1**= Name of bus to which the source's first terminal is connected. Remember to specify the node order if the terminals are connected in some unusual manner. Side effect: The processing of this property results in the setting of the Bus2 property so that all conductors in terminal 2 are connected to ground. For example,

```
Bus1=busname
```

Has the side effect of setting Bus2=busname.0.0.0

**Bus2**= Name of bus to which the source's second terminal is connected. If omitted, the second terminal is connected to ground (node 0) at the bus designated by the Bus1 property.

**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.

**MVAc3**= 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.

**ScanType=** {pos*| zero | none} Maintain specified symmetrical component sequence to assume for Harmonic mode solution. Default is positive sequence. Otherwise, angle between phases rotates freely with harmonic.

**Sequence=** {pos*| neg | zero} Set the phase angle relationships for the specified symmetrical component sequence for solution modes other than Harmonics. Default is positive sequence.

**Spectrum=** Name of harmonic spectrum for this source. Default is "defaultvsource", which is defined when the DSS starts.

**Z1** Positive-sequence impedance, ohms, as a 2-element array representing a complex number. Example:

**Z1=[1, 2] ! represents 1 + j2**

If defined, Z1, Z2, and Z0 are used to define the impedance matrix of the Vsource. Z1 MUST BE DEFINED TO USE THIS OPTION FOR DEFINING THE MATRIX.

*Side Effect*: Sets Z2 and Z0 to same values unless they were previously defined. (Same rules as the Reactor element.)

**Z2** Negative-sequence impedance, ohms, as a 2-element array representing a complex number. Example:

**Z2=[1, 2] ! represents 1 + j2**

Used to define the impedance matrix of theVsource.

Note: Z2 defaults to Z1 if it is not specifically defined. If Z2 is not equal to Z1, the impedance matrix is asymmetrical. If the Vsource is close to a generator or represents a generator, you may want to set Z2 somewhat lower than Z1 to show the proper behavior for harmonics and unbalanced loading.

**Z0** Zero-sequence impedance, ohms, as a 2-element array representing a complex number. Example:

**Z0=[3, 4] ! represents 3 + j4**

Used to define the impedance matrix of the Vsource if Z1 is also specified.

Note: Z0 defaults to Z1 if it is not specifically defined.

**puZ1** = Per-unit positive-sequence impedance on base of Vsource BasekV and BaseMVA. See Z1 definition. Transmission system short circuit equivalents are often expressed in per unit and this offers a convenient way to enter those values. Be sure to specify BaseMVA property if different than the common 100 MVA.

**puZ2** = See Z2 property. Per-unit negative-sequence impedance on base of Vsource BasekV and BaseMVA.

**puZ0** = See Z0 property. Per-unit zero-sequence impedance on base of Vsource BasekV and

BaseMVA.

**baseMVA** Default value is 100. Base used to convert values specifiied with puZ1, puZ0, and puZ2 properties to ohms on kV base specified by BasekV property.

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

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

## ISOURCE OBJECT

Current source. This is a one-terminal current source object that can be connected to any bus. Its most common use is likely to be used to represent harmonic sources and to be used in frequency response scans of circuit models. You can perform positive- or zero-sequence scans. Isource objects can also be controlled through the COM interface or other APIs and controlled to represent many different kinds of circuit element for various studies. A very flexible circuit element.

Note that if the device you are trying to model produces or consumes power, it is generally better to model it with a Load or Generator object. The power flow algorithm will automatically determine the phase angle of the current source in the Norton equivalent used to represent these devices. This is sometimes difficult to do correctly with an Isource object.

You can generally attach as many Isource objects to a bus as you want. An Isource object is assumed to be ideal and its Yprim matrix is zero.

Properties:

**phases** Number of phases. Defaults to 3. For 3 or less, phase shift defaults to 120 degrees.

**bus1** Name of bus to which source is connected.
bus1=busname
bus1=busname.1.2.3

**amps** Magnitude of current source, each phase, in Amps.

**angle** Phase angle in degrees of first phase: e.g.,Angle=10.3.
Phase shift between phases defaults to 120 degrees when number of phases <= 3

**frequency** Source frequency. Defaults to circuit fundamental frequency.

**scantype** {pos*| zero | none} Maintain specified sequence for harmonic solution. Default is positive sequence. Otherwise, angle between phases rotates with harmonic.

**sequence** {pos*| neg | zero} Set the phase angles for the specified symmetrical component sequence for solution modes other than Harmonics. Default is positive sequence.

**spectrum** Harmonic spectrum assumed for this source. Default is "default".

Inherited properties

**basefreq** Base Frequency for ratings.

**enabled** {Yes|No or True|False} Indicates whether this element is enabled.

**like** Make like another object, e.g.:

New Isource.Is2 like=Is1 ...

## FAULT OBJECT

A Fault object is a resistor network that can be configured in a variety of ways. It is nominally designed with the same connection philosophy as the Capacitor. That is, it is a two-terminal device in which the second terminal defaults to ground. This is often what is desired when simulating a fault for short-circuit studies. However, the OpenDSS Fault object may be configured to do much more and can be configured to represent any type of fault. For example, it can be connected between transmission overbuild and distribution underbuild to simulate the transmission falling on the distribution circuit.

A Fault object is a standard linear Power Delivery component, completely defined by its primitive admittance matrix. You can have as many Fault objects on the circuit as you wish (some may cause the power flow solution to diverge – if so, switch to a direct solution or a dynamic solution). For Monte Carlo fault mode (MF) you will distribute Fault objects all over the circuit is some proportion. The solver will enable one at a time for each solution.

Since the Fault object is nothing more than a resistor network, you may use it for purposes other than modeling short circuit faults – anything that requires a resistor model. With the Gmatrix property a very complex resistive network can be modeled. For example, you may wish to represent a fault that has one resistance L-L and another L-ground. Note that it is specified as a nodal conductance matrix.

In time mode simulations such as Dynamics, the initiation of the fault can be delayed (ONtime property) and it will automatically clear itself when the current drops below a certain level (MinAmps property) if the fault is declared to be Temporary.

| | |
|---|---|
| **phases** | Number of Phases. **Default is 1**. |
| **bus1** | Name of first bus. Examples:<br>bus1=busname<br>bus1=busname.1.2.3 |
| **bus2** | Name of 2nd bus. Defaults to all phases connected to first bus, node 0, if not specified. (Shunt Wye to ground connection) |
| **R** | Resistance, each phase, ohms. Default is 0.0001. Assumed to be Mean value if gaussian random mode.Max value if uniform mode. A Fault is actually a series resistance that defaults to a wye connection to ground on the second terminal. You may reconnect the 2nd terminal to achieve whatever connection. Use the Gmatrix property to specify an arbitrary conductance matrix. |
| **Gmatrix** | Use this to specify a nodal conductance (G) matrix to represent some arbitrary resistance network. Specify in lower triangle form as usual for DSS matrices. |
| **MinAmps** | Minimum amps that can sustain a temporary fault. Default is 5. |
| **ONtime** | Time (sec) at which the fault is established for time varying simulations. Default is 0.0 (on at the beginning of the simulation) |
| **pctperm** | Percent of failures that become permanent. (not currently used) |
| **Temporary** | {Yes \| No} Default is No. Designate whether the fault is temporary. For Time-varying simulations, the fault will be removed if the current through the fault drops |

below the MINAMPS criteria.

%**stddev** Percent standard deviation in resistance to assume for Monte Carlo fault (MF) solution mode for GAUSSIAN distribution. Default is 0 (no variation from mean).

Standard Inherited Properties (may not apply to all Fault elements)

**normamps** Normal rated current.

**emergamps** Maximum current.

**basefreq** Base Frequency for ratings.

**faultrate** No. of failures per year. This property can be used by such things as a monte carlo fault study to specify how often this fault is likely to occur.

**repair** Hours to repair.

**enabled** {Yes|No or True|False} Indicates whether this element is enabled.

**like** Make like another object, e.g.:

New Fault.F2 like=F1 ...

# Power Delivery Elements (PDelement)

## 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.

These connection rules also apply to Vsource, Reactor, and Fault elements.

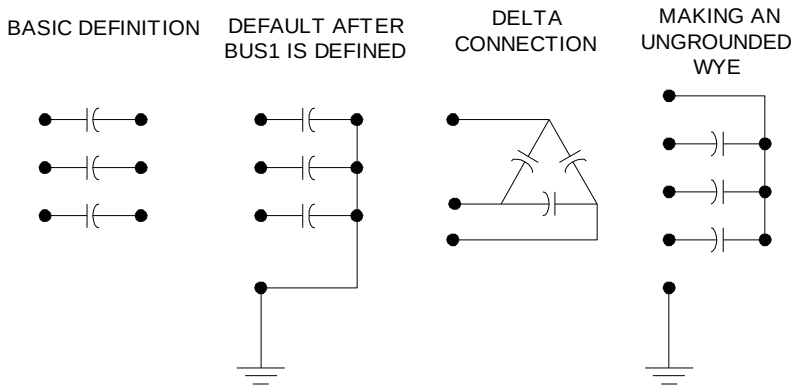



**Figure 41: Definition of Capacitor Object**

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 is frequently 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.

Properties, 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. At each time step, the States array is recorded by a Monitor object connected to the Capacitor with Mode=6.

**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.

## LINE OBJECT

The Line element is used to model most multi-phase, two-port lines or cables. It is a "Pi" model with shunt capacitance. This is a 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. Or you may specify an existing Geometry object and the line impedances will be computed.

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 default Line object is a 1000-ft overhead line with 336 MCM ACSR conductor on a 8-ft crossarm. It is defined by symmetrical component values:

```
R1 = 0.0580  ohms per 1000 ft
X1 = 0.1206  ohms per 1000 ft
R0 = 0.1784  ohms per 1000 ft
X0 = 0.4047  ohms per 1000 ft
C1 = 3.4e-9 nF per 1000ft
C0 = 1.6e-9 nF per 1000ft
```

The corresponding values of Rg and Xg for Rho=100 and 60 Hz are:

```
 Rg = 0.01805  ohms per 1000 ft
Xg = 0.155081  ohms per 1000 ft
```

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.

Symmetrical Component Impedance Definition Properties

If any of the following properties are used, the Line object primitive Y matrix is computed using the present values of the other symmetrical component line properties. This is the default method. One limitation to using the symmetrical component method is you can't correctly represent asymmetrical line constructions. Use the matrix properties or the Geometry property to overcome this limitation.

**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. Setting any of R1, R0, X1, X0, C1, C0 forces the program to use the symmetrical component line definition. See also Cmatrix and B1

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

B1= Alternate way to enter C1, microS per unit length.

B0= Alternate way to enter C0, microS 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.

Matrix Impedance Definition Properties

You may define line impedances in a Line object definition using matrices. If you use any of the following properties and do not supercede it with a symmetrical component property or either a LineCode or Geometry property, the primitive Y matrix is computed assuming you wish to use the quantities defined through the matrix properties below. The default matrix values correspond to the symmetrical component defaults described above. The remain at those values until you change them. Matrix properties allow you great flexibility in modeling all sorts of line asymmetries. If you will be solving at some frequency other than the base frequency, be sure to define Rg and Xg (see LineCode object description).

**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. Applies to line impedances defined using the Line object properties described here or to impedances defined by LineCode objects. Does not apply to impedances defined by Geometry. See explanation in LineCode object. Default values are same as for default LineCode object, which corresponds to the default Line object impedances. Set both Rg and Xg = 0 if you do not wish for earth return correction to be adjusted when the frequency of the solution is different than the base frequency.

**Xg**= Carson earth return reactance per unit length used to compute impedance values at base frequency. For making better frequency adjustments. Applies to line impedances defined using the Line object properties described here or to impedances defined by LineCode objects. Does not apply to impedances defined by Geometry. See explanation in LineCode object. Default values are same as for default LineCode object, which corresponds to the default Line object impedances. Set both Rg and Xg = 0 if you do not wish for earth return correction to be adjusted when the frequency of the solution is different than the base frequency.

**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 cause the number of phases to be redefined.

**EarthModel** One of {Carson | FullCarson | Deri}. Default is the global value established with the **Set EarthModel** option. See the Options Help on EarthModel option. This is used to override the global value for this line. This option applies only when the "geometry" property is used.

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

**Seasons=** Defines the number of ratings to be defined for the wire, to be used only when defining seasonal ratings using the "Ratings" property.

**Ratings=** An array of ratings to be used when the seasonal ratings flag is True. It can be used to insert multiple ratings to change during a QSTS simulation to evaluate different ratings in lines.

**LineType=** Code designating the type of line. One of: OH, UG, UG_TS, UG_CN, SWT_LDBRK, SWT_FUSE, SWT_SECT, SWT_REC, SWT_DISC, SWT_BRK, SWT_ELBOW. OpenDSS currently does not use this internally. For whatever purpose the user defines. Default is OH.

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

## REACTOR OBJECT

The Reactor element is an extremely flexible and powerful circuit element. The Reactor element is implemented with basically the same philosophy as a Capacitor (and a Fault object). It is a constant impedance element that may be configured into a variety of connections. Like the Capacitor, the second terminal defaults to a Wye connection to ground if not specified. In that case, it is flagged as a Shunt element. It is a constant impedance element entirely represented by its primitive Y matrix.

A Reactor object can have frequency-dependent R and L by supplying an appropriate per-unit XYCurve object. You can put a Reactor in series with another circuit element to give the element frequency dependence. This might be common for a Transformer for harmonics studies. If a Line object is defined by a LineGeometry object, frequency-dependence is often automatic. However, if defined by a LineCode object you can use a Reactor element, appropriately defined, to impart frequency dependence.

The reactor may be conceived as a series R-L or a parallel R-L connection (see Parallel property). By default it is an inductance with series resistance. You may also specify a resistor, Rp, in parallel with the entire branch. These options allow the Reactor impedance characteristic to have different frequency response characteristics, which are often useful for some Harmonics-mode simulations, particularly for filters. In the case of a shunt reactor, the Rp value is used for no-load losses.

Zero-impedance devices cannot exist in OpenDSS. However, either R or X can be zero in this model, but not both. A Reactor element can be used for source equivalent or other equivalent impedances where the capacitance of a Line element is unnecessary. Use a 1-phase reactor to model neutral reactors in transformers, generators, or loads.

By default, the Reactor has no coupling between the phases. Shunt reactors would typically be defined by kV and kvar properties, similar to a capacitor. Series reactors without mutual coupling would be defined by the R and X properties. Of course, either could be used in all circumstances.

You can use Reactor objects to represent lines and switches. However, keep in mind that you do not get any shunt capacitance as a Line element would naturally give. Thus, it is easier to accidentally create isolated islands with no conductive path to ground. This will result in Y matrices that will not invert and you will get floating-point errors.

Note that if the connection is specified as “delta” only the first terminal matters; there is no second terminal. This applies only to shunt reactors. By leaving the Conn property as wye and specifying both terminal connections explicitly, nearly any reactor configuration can be achieved. Mutual coupling between phases can be achieved by specifying Rmatrix and Xmatrix properties. Note that the matrix specification is mutually exclusive with the other means of specifying the reactance values. Of course, the Rmatrix and Xmatrix properties may be defined with zero mutual coupling. The Parallel property applies to the Rmatrix and Xmatrix specification.

| | |
|---|---|
| **phases** | Number of phases. |
| **bus1** | Name of first bus. Examples:<br>bus1=busname<br>bus1=busname.1.2.3 |

**bus2** Name of 2nd bus. Defaults to all phases connected to first bus, node 0. (Shunt Wye Connection)
Not necessary to specify for delta (LL) connection

**kv** For 2, 3-phase, kV phase-phase. Otherwise specify actual coil rating.

**kvar** Total kvar, all phases. Evenly divided among phases. Only determines X. Specify R separately

**conn** ={wye | delta |LN |LL} Default is wye, which is equivalent to LN. If Delta, then only one terminal.

**Parallel** {Yes | No} Default=No. Indicates whether Rmatrix and Xmatrix are to be considered to be in parallel. This makes a significant difference in harmonic studies. Default is series. For other models, specify R and Rp.

**R** Resistance (in series with reactance), each phase, ohms.

**Rmatrix** Resistance matrix, lower triangle, ohms at base frequency. Order of the matrix is the number of phases. Mutually exclusive to specifying parameters by kvar or R.

**Rp** Resistance in parallel with R and X (the entire branch). Assumed infinite if not specified.

**X** Reactance, each phase, ohms at base frequency.

**Xmatrix** Reactance matrix, lower triangle, ohms at base frequency. Order of the matrix is the number of phases. Mutually exclusive to specifying parameters by kvar or X.

**Z** Alternative way of defining R and X properties. Enter a 2-element array representing R +jX in ohms. Example**:**

**Z=[5 10] ! equivalent to R=5 X=10**

**Z1** Positive-sequence impedance, ohms, as a 2-element array representing a complex number. Example:

**Z1=[1, 2] ! represents 1 + j2**

If defined, Z1, Z2, and Z0 are used to define the impedance matrix of the REACTOR. Z1 MUST BE DEFINED TO USE THIS OPTION FOR DEFINING THE MATRIX.

*Side Effect*: Sets Z2 and Z0 to same values unless they were previously defined.

**Z2** Negative-sequence impedance, ohms, as a 2-element array representing a complex number. Example:

**Z2=[1, 2] ! represents 1 + j2**

Used to define the impedance matrix of the REACTOR if Z1 is also specified.

Note: Z2 defaults to Z1 if it is not specifically defined. If Z2 is not equal to Z1, the impedance matrix is asymmetrical.

**Z0** Zero-sequence impedance, ohms, as a 2-element array representing a complex number. Example:

**Z0=[3, 4] ! represents 3 + j4**

Used to define the impedance matrix of the REACTOR if Z1 is also specified.

Note: Z0 defaults to Z1 if it is not specifically defined.

**RCurve** Name of **XYCurve** object, previously defined, describing per-unit variation of phase resistance, R, vs. frequency. Applies to resistance specified by R or Z property. If

actual values are not known, R often increases by approximately the square root of frequency.

**LCurve** Name of **XYCurve** object, previously defined, describing per-unit variation of phase inductance, L=X/w, vs. frequency. Applies to reactance specified by X, LmH, Z, or kvar property.L generally decreases somewhat with frequency above the base frequency, approaching a limit at a few kHz.

**LmH** Inductance, mH. Alternate way to define the reactance, X, property

Properties inherited from the circuit element class:

**normamps** Normal rated current.

**emergamps** Maximum current.

**repair** Hours to repair.

**faultrate** No. of failures per year.

**pctperm** Percent of failures that become permanent.

**basefreq** Base Frequency for ratings.

**enabled** {Yes|No or True|False} Indicates whether this element is enabled.

**like** Make like another object, e.g.:

New Reactor.R2 like=R1 ...

## 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.

Note that you can define an **XfmrCode** object to define a Transformer object. This will same some coding in large circuits where many transformers are identical.

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).

Properties, 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 properties to set the winding values using arrays (setting of wdg= … is ignored). The names of these properties are simply the plural form of the property name above.

**Buses** = Array of 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,…].

**%Rs** = Array of percent resistances for windings [1, 2. …]

Use the following propertis to define the reactances of the transformer. For 2- and 3-winding transformers, you may use the conventional XHL, XLT, and XHT (or X12, X23, X13) 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 (or X12)** = Percent reactance high-to-low (winding 1 to winding 2).

**XLT (or X23)** = Percent reactance low-to-tertiary (winding 2 to winding 3).

**XHT (or X13)**= 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 (PPM) 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 an amount equal to rated kVA/1.0e6. Prevents a singular matrix if delta winding left floating. Zig-Zag transformers are also susceptible to this. Set this to zero if you don't need it and the resulting impedance to ground is affecting the

results. Is inconsequential for most cases. Can be negative to represent capacitive ground, if you prefer (but you can create unintentional resonances at very high frequencies.)

**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.

**Sub** = Yes/No. Flag that designates whether this transformer is to be treated as a substation. Default is No. Allows substations to show up differently on circuit plots.

**MaxTap =** Max per unit tap for the active winding. Default is 1.10

**MinTap =** Min per unit tap for the active winding. Default is 0.90

**NumTaps =** Total number of taps between min and max tap. Default is 32 (16 raise and 16 lower taps about the neutral position). The neutral position is not counted.

**SubName =** Substation Name. Optional. Default is null. If specified, printed on plots

**Bank =** Name of the bank this transformer is part of, for CIM, MultiSpeak, and other industry database interfaces.

**XfmrCode =** Name of a library entry of the XfmrCode class for transformer properties. The named XfmrCode must already be defined. You can use this instead of the properties of the same name in the Transformer class.

**XRConst =** {Yes|No*} Default is NO. Signifies whether or not the X/R is assumed contant for harmonic studies, a common assumption. Note: You may also insert a frequency-dependent Reactor object in series with the transformer to impart frequency-dependent characteristics.

**LeadLag =** {Lead | Lag (default) | ANSI (default) | Euro } Designation in mixed Delta-wye connections signifying the relationship between HV to LV winding. Default is ANSI 30 deg lag, e.g., Dy1 of Yd1 vector group. To get typical European Dy11 connection, specify either "lead" or "Euro".

**Seasons=** Defines the number of ratings to be defined for the transformer, to be used only when defining seasonal ratings using the "Ratings" property.

**Ratings=** An array of ratings to be used when the seasonal ratings flag is True. It can be used to insert multiple ratings to change during a QSTS simulation to evaluate different ratings in transformers. Is given in kVA.

Inherited Properties:

**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.

## GICTRANSFORMER OBJECT

The GICTransformer model is used in combination with the GICLine model for the calculation of

geomagnetically induced current (GIC). The GICLine model contains the source representing the induced voltage. Transformers with a connection to ground provide a return path for the resulting current.

Power transformers are represented by their dc equivalent circuits, and only windings with connection to ground are included in the analysis. The model is a resistor network for the purposes of computing GIC in transformers. Only frequency domain models are necessary.

### *Generator Step-Up Banks*

A common transformer connection in large networked power systems is the grounded-wye delta connected generator step-up transformer (GSU). A three-phase model of a GSU used in GIC calculations is shown in Figure 42.

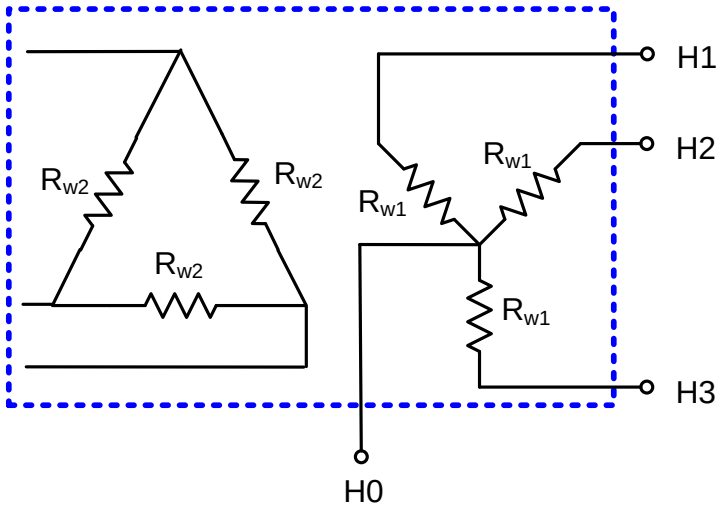



**Figure 42. Three-phase Model of a Generator Step-Up Transformer**

Note that the delta winding does not have any connection to the external network since it is not physically connected to ground and is open to the zero sequence. Thus, it is not included in the OpenDSS model. The HO terminal refers to the neutral point, and is modeled explicitly. $R_{w1}$ and $R_{w2}$ are defined as the dc winding resistance values of the high voltage or extra-high voltage and medium voltage windings, respectively.

Transformer windings are modeled with resistive branch circuits as shown in Figure 43.

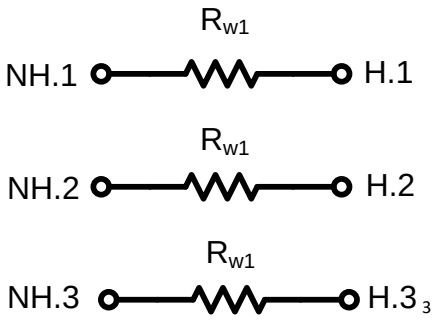



**Figure 43. GSU Model in OpenDSS**

The winding terminals designated as NH.1, NH.2, and NH.3 in Fig. 23 must be connected together to construct the wye winding. The bus number is arbitrary, but typically the name of the high side bus is used with the next available terminal number. An example OpenDSS script with high voltage bus named 'Bus1' and dc winding resistance of 0.1 Ω/phase is as follows:

```
New GICTransformer.T1 busH=Bus1.1.2.3 busNH=Bus1.4.4.4 R1=0.1 type=GSU
```

To illustrate the previous comment of utilizing the next available terminal value for the neutral bus, two identical GSUs in parallel can be described by the following:

```
New GICTransformer.T1 busH=Bus1.1.2.3 busNH=Bus1.4.4.4 R1=0.1 type=GSU
New GICTransformer.T2 busH=Bus1.1.2.3 busNH=Bus1.5.5.5 R1=0.1 type=GSU
```

Naming the neutral buses in this manner allows for the creation of individual neutral buses for each GSU which is necessary when modeling GIC blocking devices.

## *Three-Winding Transformers*

A three-phase model of a three-winding transformer used in the calculation of GIC is shown in Figure 44. Note that the delta tertiary winding is not included in the model since it does not have any physical connection to ground. Thus, the same model can be used for both two winding and three winding transformers. The neutral nodes of both of the wye windings (i.e. X0 and H0) are modeled explicitly. In some cases, either the X0 or H0 bushing may be ungrounded. In the GIC model, the node can be grounded through a large resistance, e.g. 1 MΩ to represent such connections.

[3]

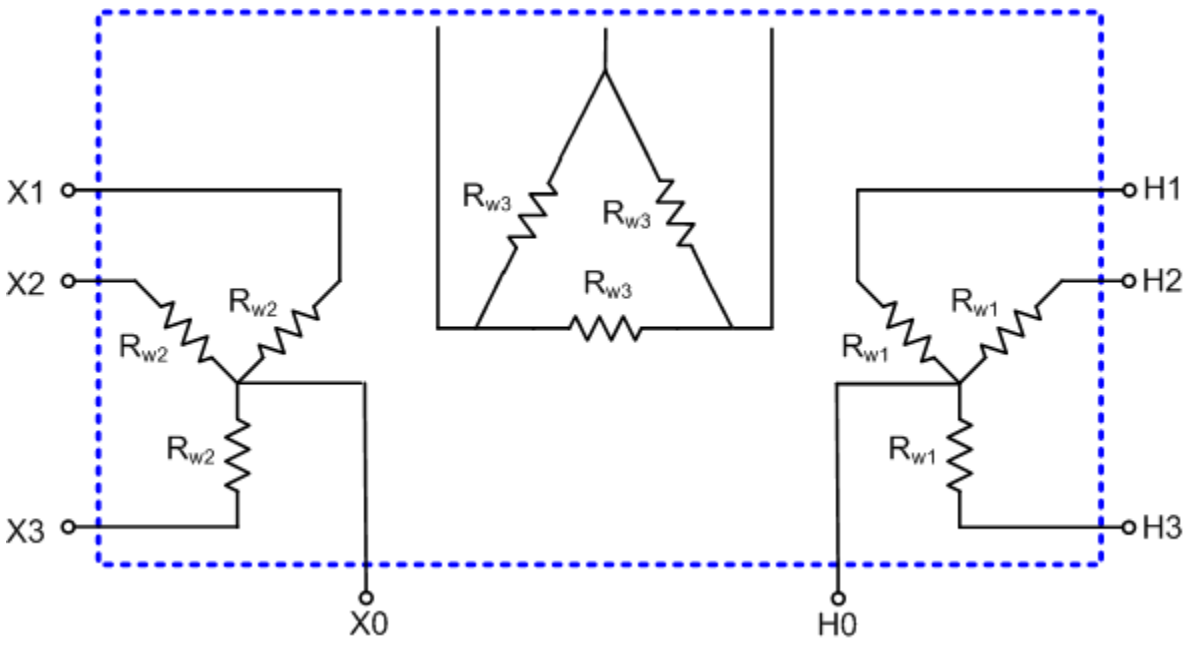



**Figure 44. Three-phase Model of a Three-Winding Transformer (Grounded-Wye, Grounded-Wye, Delta)**

$R_{w1}$ and $R_{w2}$ are defined as the dc winding resistance values of the high voltage or extra-high voltage and medium voltage windings, respectively. $R_{w3}$ is defined as the dc winding resistance values of the low voltage tertiary winding; however, it is not included in the DSS model since it has no physical connection to ground.

Transformer windings are modeled in DSS with resistive branch circuits as shown in Figure 45.

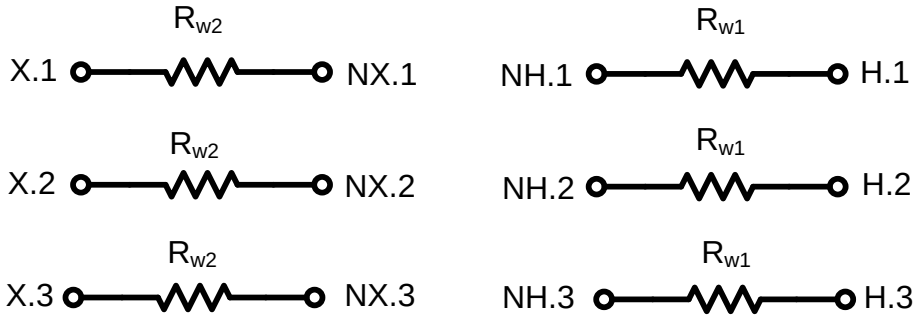



**Figure 45. Two-winding and Three-winding Transformer model in DSS**

The winding terminals designated as NH.1, NH.2, NH.3 and NX.1, NX.2, NX.3 in Figure 45 must be connected together to construct each of the wye windings. The bus numbers are arbitrary, but typically the name of the high side bus is used with the next available terminal number. An example OpenDSS script with high voltage bus named 'Bus1', low voltage bus named 'Bus2', dc winding resistance of 0.2 Ω/phase for the H winding and 0.1 Ω/phase for the X winding is as

follows:

```
New GICTransformer.T1 busH=Bus1.1.2.3 busNH=Bus1.4.4.4 busX=Bus2.1.2.3
busNX=Bus1.5.5.5 R1=0.2 R2=0.1 type=YY
```

### *Autotransformers*

A three-phase model of an autotransformer used in determining GIC flow is shown in Figure 46. Note that the delta tertiary winding (if applicable) is not included in the DSS model since it does not have any physical connection to ground. The common autotransformer neutral (H0/X0) is modeled explicitly.

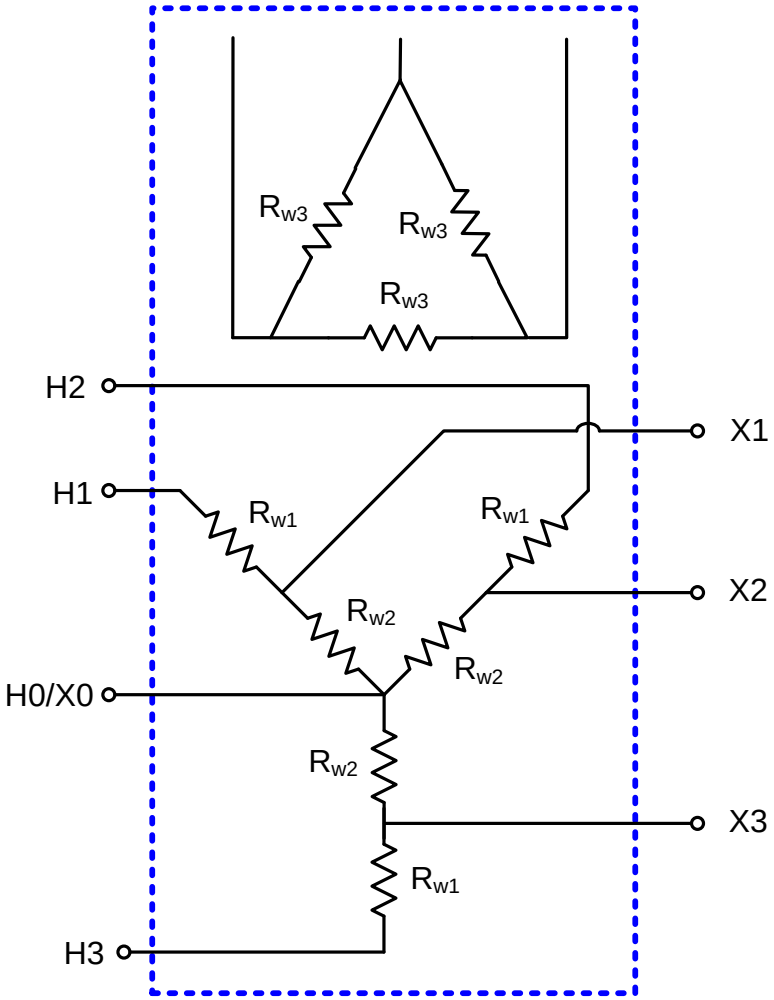



**Figure 46 Three-phase Model of a Three-Winding Autotransformer**

$R_{w1}$ and $R_{w2}$ are defined as the dc resistance values of the series and common windings, respectively. Transformer windings are modeled in DSS with resistive branch circuits as shown in Figure 47.

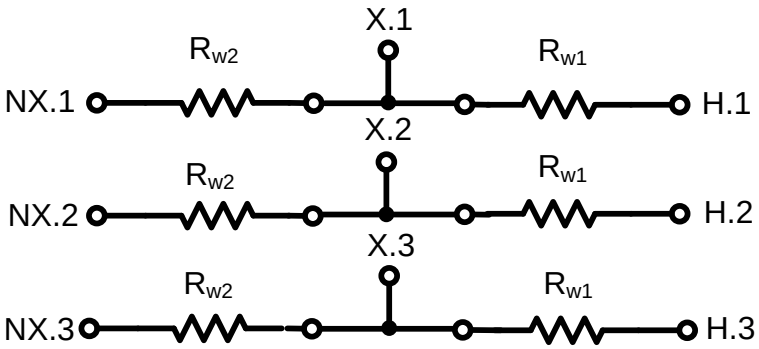



**Figure 47. Two-winding and Three-winding Autotransformer model in DSS**

The winding terminals designated as NX.1, NX.2, and NX.3 in Figure 47 must be connected together to construct the neutral connection. The bus numbers are arbitrary, but typically the name of the high side bus with the next available terminal number is used to designate the neutral bus. An example OpenDSS script with high voltage bus named 'Bus1', low voltage bus named 'Bus2', dc winding resistance of 0.1 Ω/phase for the series winding and 0.2 Ω/phase for the common winding is as follows:

```
New GICTransformer.T1 busH=Bus1.1.2.3 busX=Bus2.1.2.3 busNX=Bus1.4.4.4 R1=0.1
R2=0.2 type=auto
```

The properties of the GICTransformer model, in order, are:

**Basefreq=** Inherited Property for all PCElements. Base frequency for specification of reactance value.

**busH**= Name of bus High-side (H) bus. Node order definitions optional.

**busNH**= Name of Neutral bus for H, or first, winding. Defaults to all phases connected to H-side bus, node 0, if not specified and transformer type is either GSU or YY. (Shunt Wye Connection to ground reference)For Auto, this is automatically set to the X bus.

**busNX**= Name of Neutral bus for X, or Second, winding. Defaults to all phases connected to X-side bus, node 0, if not specified. (Shunt Wye Connection to ground reference).

**busX**= Name of bus Low-side (X) bus. Node order definitions optional.

**emergamps=** Maximum current. Typically not specified in GIC calculations.

**enabled=** {Yes|No or True|False} Indicates whether this element is enabled. Default is Yes/True.

**phases=** Number of phases. Default is 3.

**R1=** Resistance, each phase, ohms for H winding, (Series winding, if Auto). Default is 0.0001.

**R2=** Resistance, each phase, ohms for X winding, (Common winding, if Auto). Default is 0.0001.

**Type=** Type of transformer: {GSU* | Auto | YY}. Default is GSU.

**MVA=** Optional. MVA Rating assumed Transformer. Default is 100. Used for computing vars due to GIC and winding resistances if kV and MVA ratings are specified.

**KVLL1=** Optional. kV LL rating for H winding (winding 1). Default is 500. Required if you are going to export vars for power flow analysis or enter winding resistances in percent.

**KVLL2=** Optional. kV LL rating for X winding (winding 2). Default is 138. Required if you are going to export vars for power flow analysis or enter winding resistances in percent..

**%R1=** Optional. Percent Resistance, each phase, for H winding (1), (Series winding, if Auto). Default is 0.2. Alternative way to enter R1 value. It is the actual resistances in ohmns that matter. MVA and kV should be specified.

**%R2=** Optional. Percent Resistance, each phase, for X winding (2), (Common winding, if Auto). Default is 0.2. Alternative way to enter R2 value. It is the actual resistances in ohms that matter. MVA and kV should be specified.

**K=** Mvar K factor. Default way to convert GIC Amps in H winding (winding 1) to Mvar. Default is 2.2. Commonly-used simple multiplier for estimating Mvar losses for power flow analysis.

Mvar = K * kvLL * GIC per phase / 1000

Mutually exclusive with using the VarCurve property and pu curves.If you specify this (default), VarCurve is ignored.

**VarCurve=** Optional. *XYCurve* object name. Curve is expected as TOTAL pu vars vs pu GIC amps/phase. Vars are in pu of the MVA property. No Default value. Required only if you are going to export vars for power flow analysis using curves. See K property.

**like=** Make like another object, e.g. New GICTransformer.T2 like=T1 ...

The following properties are inherited from the Power Delivery element class, but are ignored for GIC calculations

**normamps=** Normal rated current. Typically not specified in GIC calculations.

**emergamps=** Maximum current. Typically not specified in GIC calculations.

**pctperm=** Percent of failures that become permanent. Typically not specified in GIC calculations.

**repair=** Hours to repair. Typically not specified in GIC calculations.

# Power Conversion Elements (PCelement)

## GICLine Object

The GICLine model is used in the calculation of geomagnetically induced current (GIC). Normally, one would think of a Line element as a power delivery device, but in GIC analysis a Line is actually a source. Each phase of the transmission line is represented by a quasi-dc induced voltage (0.1 Hz) in series with the dc resistance of the line. The resulting model is depicted in Figure 48.

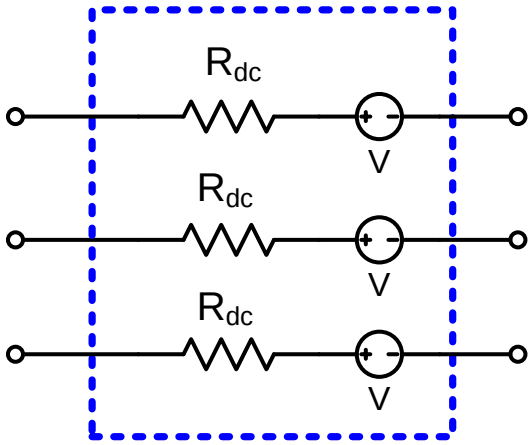



**Figure 48. GICLine Model**

The induced voltage shown in Figure 48 can be computed internally by the program or supplied via the Volts property of the GICLine model. The following describes the procedure utilized by OpenDSS to compute the induced voltage.

The induced voltage shown in Figure 48 is determined by an application of Faraday's Law:

$$V = \oint \vec{E} \circ d\vec{l}$$

where $\vec{E}$ is the electric field vector at the location of the transmission line, and $d\vec{l}$ is the incremental line segment length including direction. Because the distance between the source of the induced geoelectric field (electrojet) and the earth's surface is generally on the order of 100-200 km, the electric field at the height of the transmission line can assumed to be the same as that of the earth's surface. If the geoelectric field is assumed constant in the geographical area of a transmission line, then only the coordinates of the end point of the line are important, regardless of routing twists and turns. The resulting incremental length vector $d\vec{l}$, becomes $\vec{L}$. The induced voltage can therefore be computed as follows:

$$V = \oint \vec{E} \circ d\vec{l} = \vec{E} \circ \vec{L}$$

The vector $\vec{L}$, representing the length and direction of the line between end points can be constructed using an arbitrary reference; however, such methods can introduce error. An improved method is to construct the vector $\vec{L}$, by computing the distance in the Northward and Eastward directions independently as depicted in Figure 49.

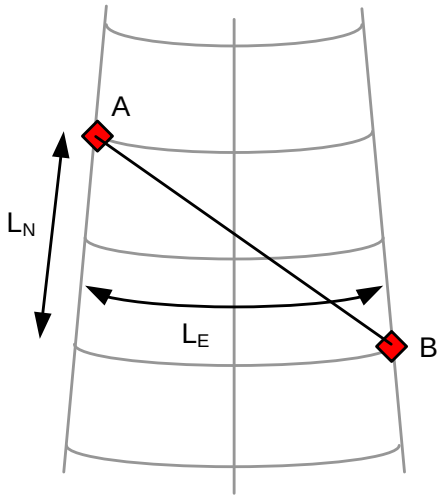



**Figure 49. Substation Location Coordinates**

Recall that the dot product of two vectors A and B can be computed using:

$$\vec{A} \circ \vec{B} = A_x B_x + A_y B_y$$

Thus, the dot product of the induced electric field and the length vector can be approximated by:

$$\vec{E} \circ \vec{L} = E_N L_N + E_E L_E$$

where $E_N$ is the northward electric field (V/km), $E_E$ is the eastward electric field (V/km), $L_N$ is the northward distance (km), and $L_E$ is the eastward distance (km).

The following procedure can be used to compute northward and eastward distances. Consider a transmission line between substations A and B as shown in Figure 49. The northward distance can be computed using:

$$L_N = 111.2 \cdot \Delta lat$$

where Δlat is the difference in latitude (degrees) between the two substations A and B. The eastward distance can be computed using:

$$L_E = 111.2 \cdot \Delta long \cdot sin(90 - \alpha)$$

where Δlong is the difference in longitude (degrees) between the two substations A and B, and $\alpha$ is defined as the average of the two latitudes:

$$\alpha = \frac{LatA + LatB}{2}$$

The properties of the GICLine model, in order, are:

**Angle=** Phase angle in degrees of first phase. Default=0.0. See Voltage property

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

**bus2**= Name of bus for terminal 2.

**C=** Value of line blocking capacitance in microfarads. Default = 0.0, implying that there is no line blocking capacitor.

**EE=** Eastward Electric field. If specified, Voltage and Angle are computed from EN, EE, lat and lon values.

**EN=** Northward Electric field. If specified, Voltage and Angle are computed from EN, EE, lat and lon values.

**frequency=** Source frequency. Defaults to 0.1 Hz.

**Lat1=** Latitude of Bus 1 (degrees)

**Lat2=** Latitude of Bus 2 (degrees)

**Lon1=** Longitude of Bus 1 (degrees)

**Lon2=** Longitude of Bus 2 (degrees)

**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.

**R=** Resistance of line, ohms of impedance in series with GIC voltage source.

**Volts=** Voltage magnitude, in volts, of the GIC voltage induced across this line. When spedified, voltage source is assumed defined by Voltage and Angle properties. Specify this value OR EN, EE, lat1, lon1, lat2, lon2. Not both!! Last one entered will take precedence. Assumed identical in each phase of the Line object.

**X=** Reactance at base frequency, ohms. Default = 0.0. This value is generally not important for GIC studies but may be used if desired.

The following properties are common to all Power Conversion elements and inherited by GICLine.

**like=** Make like another GICLine object

**Basefreq=** Inherited Property for all PCElements. Base frequency for specification of the reactance value, X, if defined.

**enabled=** {Yes|No or True|False} Indicates whether this element is enabled. Inherited Property for all PCElements.

**Spectrum=** Inherited Property for all PCElements. Name of harmonic spectrum for this source. Default is "defaultvsource", which is defined when the DSS starts. Not used for GIC analysis.

## 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 active power, kW, for the load. Total of all phases. See kVA.

**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 (see "Power Conversion Elements" for more details). The load models currently implemented are:

1: Constant P and constant Q (Default): Commonly used for power flow studies

2: Constant Z (or constant impedance)

3: Constant P and quadratic Q

4: Exponential: $P/P_0 = \left(V/V_0\right)^{CVRwatts}$ and $Q/Q_0 = \left(V/V_0\right)^{CVRvars}$ (see CVRwars and CVRvar)

5: Constant I (or constant current magnitude) Sometimes used for rectifier load

6: Constant P and fixed Q (at the nominal value)

7: Constant P and quadratic Q (i.e., fixed reactance)

8: ZIP (see ZIPV)

"Constant" power value (either P or Q) may be modified by loadshape multipliers. "Fixed" power 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. See kVA property below.

**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. Legal ways to define base load (kW and kvar):

kW, PF

kW, kvar

kVA, PF

XFKVA * Allocationfactor, PF

kWh/(kWhdays*24) * Cfactor, PF

**%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** =.

Exponential paramater that defines the relationship between voltage ($V$) and active power ($P$) based on the following: ${}^{P}/_{P_0} = \left({}^{V}/_{V_0}\right)^{CVRwatts}$. $P_0$ is the nominal power of the load at the base voltage $V_0$. $CVR_{watts}$ default=1 (linear relationship between voltage and active power). Typical values range from 0.4 to 0.8. Applies to Model=4 only.

**CVRvars** =

Exponential paramater that defines the relationship between voltage ($V$) and reactive power ($Q$) based on the following: ${}^{Q}/_{Q_0} = \left({}^{V}/_{V_0}\right)^{CVRvars}$. $Q_0$ is the nominal power of the load at the base voltage $V_0$. $CVR_{vars}$ default=2 (quadratic relationship between voltage and reactive power). Typical values range from 2 to 3. Applies to Model=4 only.

**kWh** = kWh billed for this period. Default is 0. See help on kVA and Cfactor and kWhDays.

**kWhDays** = Length of kWh billing period in days (24 hr days). Default is 30. Average demand is computed using this value.

**CFactor** = Factor relating average kW to peak kW. Default is 4.0. See kWh and kWhdays. See kVA.

**CVRCurve** = Default is NONE. Curve describing both watt and var factors as a function of time.

Refers to a LoadShape object with both Mult and Qmult defined. Define a Loadshape to agree with yearly or daily curve according to the type of analysis being done. If NONE, the CVRwatts and CVRvars factors are used and assumed constant for the simulation period.

**NumCust** = Number of customers, this load. Default is 1.

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

**ZIPV**= Array of 7 coefficients:

1. First 3 are ZIP weighting factors for active power (should sum to 1)
2. Next 3 are ZIP weighting factors for reactive power (should sum to 1)
3. Last 1 is cut-off voltage in p.u. of base kV; load is 0 below this cut-off

No defaults; all coefficients must be specified if using model=8.

**%SeriesRL**= Percent of load that is series R-L for Harmonic studies. Default is 50. Remainder is assumed to be parallel R and L. This has a significant impact on the amount of damping observed in Harmonics solutions.

**RelWeight=** Relative weighting factor for reliability calcs. Default = 1. Used to designate high priority loads such as hospitals, etc.

Is multiplied by number of customers and load kW during reliability calcs.

**Vlowpu** = Default = 0.50. Per unit voltage at which the model switches to same as constant Z model (model=2). This allows more consistent convergence at very low voltages due to opening switches or solving for fault situations.

**puXharm** = Special reactance, pu (based on kVA, kV properties), for the series impedance branch in the load model for HARMONICS analysis. Generally used to represent motor load blocked rotor reactance. If not specified (that is, set =0, the default value), the series branch is computed from the percentage of the nominal load at fundamental frequency specified by the %SERIESRL property.

Applies to load model in HARMONICS mode only.

A typical value would be approximately 0.20 pu based on kVA * %SeriesRL / 100.0.

**XRharm** = X/R ratio of the special harmonics mode reactance specified by the puXHARM property at fundamental frequency. Default is 6.

## Inherited Properties

**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.

For power flow studies, the generator is essentially a negative load that can be dispatched. For Harmonics mode, the generator is converted to a voltage source behind the value specified for Xd" that approximately matches the power flow solution. For dynamics mode, the generator is converted to a voltage source behind an impedance with the impedance dependent on the model chosen.

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.

The Generator object is an excellent host for user-defined PCElement models supplied as User-written DLLs (generator model=6). You can use this to research models of various DER elements that produce power as well as those that consume power for which specific OpenDSS models do not exist. A template is provided with the installation of the OpenDSS program in the IndMach012a.DLL. The source code for this DLL is available.

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 for power flow simulations are:

1. Constant P, Q (* dispatch curve, if appropriate).

2. Constant Z (For simple, approximate solution)
3. Constant P, |V| somewhat like a standard power flow with voltage magnitudes and angles as the variables instead of P and Q.
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.
6. User-written model
7. Current-limited constant P, Q model (like some inverters).

Most of the time you will use #1 for planning studies, assuming you want to specify a specific power. All load models can follow *Loadshapes*. Some follow only the P component while the Type 1 can follow both a P and Q characteristic.

The default is for the generator to be a current injection source (Norton equivalent). Thus, its primitive Y matrix is similar to a Load object with a nominal equivalent admittance at rated voltage included in the primitive Y matrix and the injection current representing the amount required to compensate the Norton equivalent to achieve the desired terminal current. However, if the generator model is switched to Admittance from PowerFlow (see Set Mode command), the generator terminal current is computed simply from equivalent admittance that is included in the system Y matrix.

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

The properties, in numerical 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 is limited when voltage is 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 user-defined class. Can be any number.

**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. For Model 7, this is used to determine the upper current limit. For example, if Vminpu is 0.90 then the current limit is (1/0.90) = 111%.

**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.

**DutyStart =** Starting time offset [hours] into the duty cycle shape for this generator, defaults to 0.

**Balanced =** {Yes | No*} Default is No. For Model=7, force balanced current only for 3-phase generators. Force zero- and negative-sequence to zero.

**XRdp =** Default is 20. X/R ratio for Xdp property for FaultStudy and Dynamic modes. Required to match published textbook fault study cases that have generators with both R and X specifiedd.

## Inherited Properties

**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.

## *Generator Dynamics Model*

The Generator object automatically converts into a form of Thevenin equivalent model when the program switches to Dynamics mode. This would occur on a "set mode=dynamics" or "solve mode=dynamics …" statement as well a Faultstudy mode. Figure 50 shows the nominal form of the model for a delta-connected generator (no zero sequence modeled in the Generator). It consists of a voltage source behind subtransient reactance, *Xd'* (Xdp property), in the positive sequence network while the negative sequence is modeled as an impedance using the value specified for *Xd"* (Xdpp property). The voltage, $E_1$, is computed to approximately match the positive sequence power flow computed just prior to entering Dynamics mode.

The machine dynamics are governed by the traditional swing equations for a single mass. This simple model works reasonably well for such analyses as generator infeed to faults, open-conductor simulations, and short DG islanding simulations. The different reactance value for negative sequence yields a more accurate result for unbalanced circuit conditions than a simple voltage source behind three single-phase reactances. There are no built-in models for exciter and governor controls, which would obviously be needed for extended dynamics studies and microgrid simulations. Nevertheless, this model satisfies many needs of distribution planners for evaluating DG interconnections.

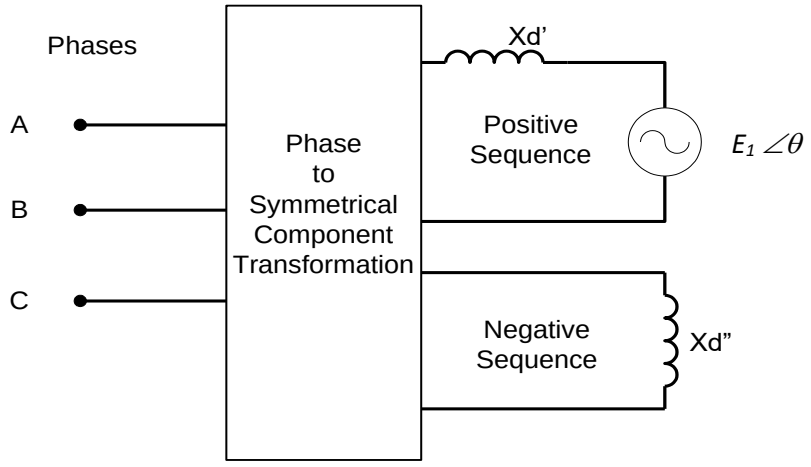



**Figure 50. Nominal Generator Model in Dynamics Mode**

When the zero sequence must be included, such as for grounded-wye connected alternators, it has been found that the OpenDSS performs better by inserting a Yg/Delta transformer with the same kVA rating in front of the model shown in Figure 50. The impedance of the transformer is set to the zero sequence impedance of the machine, which is typically about 5% or so. Then the Xdp and Xdpp properties are decreased by that amount. The impedance of the transformer provides a little linear system isolation from the nonlinear machine that assists in the convergence of the solution algorithm.

For Generator model=7, which is to approximate the nominal behavior of some inverter-based systems, the values specified for the Xdp and Xdpp properties are actually placed in the *real* part of the complex impedance. This yields a current contribution from the generator that is more in phase with $E_1$ up to the max current for the model (see *Vminpu* property above). This

approximates inverters that tend to produce a current in phase with the voltage, and is often good enough for planning purposes.

## INDMACH012 OBJECT

This is an induction machine (asynchronous machine) model that was previously supplied as a separate user-defined DLL for the Generator model called IndMach012a.dll. The functionality is virtually identical. Simply connect the model to a bus. Both power flow and dynamic mode models are provided. It will demonstrate the appropriate behavior expected from induction machines for various unbalanced circuit conditions.

An induction machine can be either a generator or load

The properties are:

**Phases**= No. of phases this induction machine. Default is 3.

**bus1**= Name of bus to which the machine is connected. Include node definitions if the terminal conductors are connected unusually. 3-phase Wye-connected machines have 4 conductors; Delta-connected have 3. 1-phase Delta (or Line-Line) has 2 conductors; 2-phase has 3. The remaining Delta, or line-line, connections have the same number of conductors as phases.

**Kv**= Base voltage for the inducation machine. 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**= Shaft Power, kW, for the Induction Machine. A positive value denotes power for a load. Negative value denotes an induction generator. Total of all phases.

**Pf**= (Read Only) nominal Power Factor for machine under present conditions. You cannot specify PF for an induction machine; this is determined by circuit conditions. If negative, the watts and vars are flowing in opposite directions. Reactive power in an induction machine is almost always INTO the machine.

**Conn =** Connection of stator: Delta or Wye. Default is Delta. Generally, delta is better for simulations in OpenDSS. Ungrounded Wye can lead to numerical instability.

**kVA=** Rated kVA for the machine.

**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 in Dynamics mode.

**puRs =** Per unit stator resistance. Default is 0.0053.

**puXs =** Per unit stator leakage reactance. Default is 0.106.

**puRr =** Per unit rotor resistance. Default is 0.007.

**puXr =** Per unit rotor leakage reactance. Default is 0.12.

**puXm =** Per unit magnetizing reactance.Default is 4.0.

**Slip =** Initial slip value. Default is 0.007

**MaxSlip =** Max slip value to allow. Default is 0.1. Set this before setting slip. In Dynamics simulations, the slip may exceed this value. This is mainly to allow the power flow

solution to converge.

**SlipOption =** Option for slip model. One of {fixedslip | variableslip* } Variable slip is the default and slip will be computed to satisfy the power specification. If fixed slip, kW is computed to match the slip specification.

**Spectrum =** Name of harmonic voltage or current spectrum for this IndMach012. Voltage behind Xd", or blocked rotor, for machine - default. Not usually much of an issue for an induction machine. This model is still under development.

**(The usual Daily, Yearly, Duty Loadshape Options)**

## STORAGE OBJECT

| Property | Description |
|---|---|
| **%Charge** | Charging rate (input power) in Percent of rated kW. Default = 100. |
| **%Discharge** | Discharge rate (output power) in Percent of rated kW. Default = 100. |
| **%EffCharge** | Percent efficiency for CHARGING the storage element. Default = 90. |
| **%EffDischarge** | Percent efficiency for DISCHARGING the storage element. Default = 90.Idling losses are handled by %IdlingkW property and are in addition to the charging and discharging efficiency losses in the power conversion process inside the unit. |
| **%Idlingkvar** | Percent of rated kW consumed as reactive power (kvar) while idling. Default = 0. |
| **%IdlingkW** | Percent of rated kW consumed while idling. Default = 1. |
| **%R** | Equivalent percent internal resistance, ohms. Default is 0. Placed in series with internal voltage source for harmonics and dynamics modes. Use a combination of %IdlekW and %EffCharge and %EffDischarge to account for losses in power flow modes. |
| **%reserve** | Percent of rated kWh storage capacity to be held in reserve for normal operation.<br><br>Default = 20.<br><br>This is treated as the minimum energy discharge level unless there is an emergency. For emergency operation set this property lower. |

| | |
|---|---|
| | Cannot be less than zero. |
| **%stored** | Present amount of energy stored, % of rated kWh. Default is 100%. |
| **%X** | Equivalent percent internal reactance, ohms. Default is 50%. Placed in series with internal voltage source for harmonics and dynamics modes. (Limits fault current to 2 pu.) Use %Idlekvar and kvar properties to account for any reactive power during power flow solutions. |
| **basefreq** | Base Frequency for ratings. |
| **bus1** | Bus to which the Storage element is connected. May include specific node specification. |
| **ChargeTrigger** | Dispatch trigger value for charging the storage.<br><br>If = 0.0 the Storage element state is changed by the State command or StorageController object.<br><br>If <> 0 the Storage element state is set to CHARGING when this trigger level is GREATER than either the specified Loadshape curve value or the price signal or global Loadlevel value, depending on dispatch mode. See State property. |
| **class** | An arbitrary integer number representing the class of Storage element so that Storage values may be segregated by class. |
| **conn** | ={wye\|LN\|delta\|LL}. Default is wye. |
| **daily** | Dispatch shape to use for daily simulations. Must be previously defined as a Loadshape object of 24 hrs, typically. In the default dispatch mode, the Storage element uses this loadshape to trigger State changes. |
| **debugtrace** | {Yes \| No } Default is no. Turn this on to capture the progress of the Storage model for each iteration. Creates a separate file for each Storage element named "STORAGE_name.CSV". |
| **DischargeTrigger** | Dispatch trigger value for discharging the storage.<br><br>If = 0.0 the Storage element state is changed by the State command or by a StorageController object.<br><br>If <> 0 the Storage element state is set to DISCHARGING when this trigger level is EXCEEDED by either the specified Loadshape curve value or the price signal or global Loadlevel value, depending on dispatch mode. See State property. |

| | |
|---|---|
| **DispMode** | {DEFAULT \| FOLLOW \| EXTERNAL \| LOADLEVEL \| PRICE } Default = "DEFAULT". Dispatch mode.<br><br>In DEFAULT mode, Storage element state is triggered to discharge or charge at the specified rate by the loadshape curve corresponding to the solution mode.<br><br>In FOLLOW mode the kW and kvar output of the STORAGE element follows the active loadshape multipliers until storage is either exhausted or full. The element discharges for positive values and charges for negative values. The loadshapes are based on the kW and kvar values in the most recent definition of kW and PF or kW and kvar properties.<br><br>In EXTERNAL mode, Storage element state is controlled by an external Storage controller. This mode is automatically set if this Storage element is included in the element list of a StorageController element.<br><br>For the other two dispatch modes, the Storage element state is controlled by either the global default Loadlevel value or the price level. |
| **duty** | Load shape to use for duty cycle dispatch simulations such as for solar ramp rate studies. Must be previously defined as a Loadshape object. Typically would have time intervals of 1-5 seconds. Designate the number of points to solve using the Set Number=xxxx command. If there are fewer points in the actual shape, the shape is assumed to repeat. |
| **DynaData** | String (in quotes or parentheses if necessary) that gets passed to the user-written dynamics model Edit function for defining the data required for that model. |
| **DynaDLL** | Name of DLL containing user-written dynamics model, which computes the terminal currents for Dynamics-mode simulations, overriding the default model. Set to "none" to negate previous setting. This DLL has a simpler interface than the UserModel DLL and is only used for Dynamics mode. |
| **enabled** | {Yes\|No or True\|False} Indicates whether this element is enabled. |
| **kv** | Nominal rated (1.0 per unit) voltage, kV, for Storage element. For 2- and 3-phase Storage elements, specify phase-phase kV. Otherwise, specify actual kV across each branch of the Storage element. If wye (star), specify phase-neutral kV. If delta or phase-phase connected, specify phase-phase kV. |
| **kVA** | kVA rating of power output. Defaults to rated kW. Used as the base for |

| | |
|---|---|
| | Dynamics mode and Harmonics mode values. |
| **kvar** | Get/set the present kvar value. Alternative to specifying the power factor. Side effect: the power factor value is altered to agree based on present value of kW. |
| **kW** | Get/set the present kW value. A positive value denotes power coming OUT of the element, which is the opposite of a Load element. A negative value indicates the Storage element is in Charging state. This value is modified internally depending on the dispatch mode. |
| **kWhrated** | Rated storage capacity in kWh. Default is 50. |
| **kWhstored** | Present amount of energy stored, kWh. Default is same as kWh rated. |
| **kWrated** | kW rating of power output. Base for Loadshapes when DispMode=Follow. Side effect: Sets KVA property. |
| **like** | Make like another object, e.g.:<br><br>New Storage.S2 like=S1 ... |
| **model** | Integer code (default=1) for the model to use for powet output variation with voltage. Valid values are:<br><br>1:Storage element injects a CONSTANT kW at specified power factor.<br><br>2:Storage element is modeled as a CONSTANT ADMITTANCE.<br><br>3:Compute load injection from User-written Model. |
| **pf** | Nominally, the power factor for discharging (acting as a generator). Default is 1.0.<br><br>Setting this property will also set the kvar property.Enter negative for leading powerfactor (when kW and kvar have opposite signs.)<br><br>A positive power factor for a generator signifies that the Storage element produces vars as is typical for a generator. |
| **phases** | Number of Phases, this Storage element. Power is evenly divided among phases. |
| **spectrum** | Name of harmonic voltage or current spectrum for this Storage element. Current injection is assumed for inverter. Default value is "default", which is defined when the DSS starts. |
| **State** | {IDLING \| CHARGING \| DISCHARGING} Get/Set present operational state. In DISCHARGING mode, the Storage element acts as a generator |

| | |
|---|---|
| | and the kW property is positive. The element continues discharging at the scheduled output power level until the storage reaches the reserve value. Then the state reverts to IDLING. In the CHARGING state, the Storage element behaves like a Load and the kW property is negative. The element continues to charge until the max storage kWh is reached and Then switches to IDLING state. In IDLING state, the kW property shows zero. However, the resistive and reactive loss elements remain in the circuit and the power flow report will show power being consumed. |
| **TimeChargeTrig** | Time of day in fractional hours (0230 = 2.5) at which storage element will automatically go into charge state. Default is 2.0. Enter a negative time value to disable this feature. |
| **UserData** | String (in quotes or parentheses) that gets passed to user-written model for defining the data required for that model. |
| **UserModel** | Name of DLL containing user-written model, which computes the terminal currents for both power flow and dynamics, overriding the default model. Set to "none" to negate previous setting. |
| **Vmaxpu** | Default = 1.10. Maximum per unit voltage for which the Model is assumed to apply. Above this value, the load model reverts to a constant impedance model. |
| **Vminpu** | Default = 0.90. Minimum per unit voltage for which the Model is assumed to apply. Below this value, the load model reverts to a constant impedance model. |
| **yearly** | Dispatch shape to use for yearly simulations. Must be previously defined as a Loadshape object. If this is not specified, the Daily dispatch shape, if any, is repeated during Yearly solution modes. In the default dispatch mode, the Storage element uses this loadshape to trigger State changes. |

## *Examples*

```
New storage.JO0235 phases=1 bus1=X_637.1 yearly=Phasealoadshape
~ kV=0.24 kwrated=25 pf=1.0 kwhrated=25
~ state=IDLING  DischargeTrigger=0.8  ChargeTrigger=0.6

// 3-phase version
New Storage.Store1 phases=3 Bus1=LVBus kV=0.400 conn=Delta kVA=60
~ kWrated=60 kWHrated=   0.20833  %reserve=50
~ state=discharge
~ kW=50  PF=1

// 1-phase L-L connected version with dynamics DLL
```

```
New Storage.Store1 phases=1 Bus1=LVBus.1.2  kV=0.400  conn=delta kVA=60
~ kWrated=60  kWHrated=   0.20833  %reserve=50
~ state=discharge
~ kW=50  PF=1
// connect to user-written dynamics model
~ DynaDLL="C:\Users\prdu001\OpenDSS\Source\DESS1\Dess1.DLL"
~ DynaData=(file=DESSModel_Test.Txt)
```

# Control Elements

One of the distinctive capabilities of the OpenDSS is that control elements are modeled separately from the power-carrying elements. This provides significant flexibility to create new models. Initially, control objects reflected only standard utility distribution system control devices. More recently, new, innovative models have been developed. Currently, there are nine controls provided with the program with more planned soon. This section gives some idea of how the control objects work and how you might exploit them.

Control elements are polled after a power flow solution has been obtained. If any control needs to operate, it places a message on the Control Queue. When it comes time for the control to operate, the message is popped off the queue and the command executed. This allows for the simulation of controls that have time delays in their operation.

The typical execution of the Solve command for a single snapshot power flow solution is:

1. Initialize Snapshot (**_InitSnap**)
2. Repeat until converged:
    a. Solve Circuit (**_SolveNoControl**)
    b. Sample control devices (**_SampleControls**)
    c. Do control actions, if any (**_DoControlActions**)

The names in parentheses after the step is the corresponding command (see command reference above) that you would use if you wanted to "roll your own" solution method.

Control elements typically have a *Sample* function that samples the voltage and current at the terminal that the control is monitoring. Each element of the Control element class also has a *DoPendingAction* function that is called from the control queue at the appropriated time. The Control element then takes the prescribed action or decides that it doesn't need to (as is the case for many distribution system controls).

Refer to the Doc folder on the OpenDSS website for additional information.

## CapControl Object

The capacitor control monitors the voltage and current at a terminal of a PDelement or a PCelement and sends switching messages to a Capacitor object. The CapControl contains essential features of many typical utility capacitor controls.

Example:

```
New CapControl.C1ctrl element=Line.L1 Capacitor=C1
~ Type=Current ON=400 OFF=300 Delay=30
```

This control monitors the current in Line.L1 and switches Capacitor.C1 based on current magnitude. This is one of the simpler controls to implement. The capacitor will switch ON when the current in the monitored phase (defaults to 1) after a delay of 30 s. If the current drops below 400 A before 30 s has elapsed, the control resets (ignores the message from the control queue).

After energization, the capacitor switches off when the current drops below 300 with the default DelayOFF.

**Element** Full object name of the circuit element, typically a line or transformer, to which the capacitor control's PT and/or CT are connected.There is no default; must be specified.

**Capacitor** Name of Capacitor element which the CapControl controls. No Default; Must be specified.Do not specify the full object name; "Capacitor" is assumed for the object class. Example:

Capacitor=cap1

**Type** {Current | voltage | kvar | PF | time } Control type. Specify the ONsetting and OFFsetting appropriately with the type of control. (See help for ONsetting)

**CTPhase** Number of the phase being monitored for CURRENT control or one of {AVG | MAX | MIN} for all phases. Default=1. If delta or L-L connection, enter the first or the two phases being monitored [1-2, 2-3, 3-1]. Must be less than the number of phases. Does not apply to kvar control which uses all phases by default.

**CTratio** Ratio of the CT from line amps to control ampere setting for current and kvar control types.

**DeadTime** Dead time after capacitor is turned OFF before it can be turned back ON. Default is 300 sec.

**Delay** Time delay, in seconds, from when the control is armed before it sends out the switching command to turn ON. The control may reset before the action actually occurs. This is used to determine which capacity control will act first. Default is 15. You may specify any floating point number to achieve a model of whatever condition is necessary.

**DelayOFF** Time delay, in seconds, for control to turn OFF when present state is ON. Default is 15.

**EventLog** {Yes/True* | No/False} Default is YES for CapControl. Log control actions to Eventlog.

**OFFsetting** Value at which the control arms to switch the capacitor OFF. (See help for ONsetting)For Time control, is OK to have Off time the next day ( < On time)

**ONsetting** Value at which the control arms to switch the capacitor ON (or ratchet up a step).

Type of Control:

Current: Line Amps / CTratio
Voltage: Line-Neutral (or Line-Line for delta) Volts / PTratio
kvar: Total kvar, all phases (3-phase for pos seq model). This is directional.
PF: Power Factor, Total power in monitored terminal. Negative for Leading.
Time: Hrs from Midnight as a floating point number (decimal). 7:30am would be entered as 7.5.

**PTPhase** Number of the phase being monitored for VOLTAGE control or one of {AVG | MAX | MIN} for all phases. Default=1. If delta or L-L connection, enter the first or the two phases being monitored [1-2, 2-3, 3-1]. Must be less than the number of phases. Does not apply to kvar control which uses all phases by default.

**PTratio** Ratio of the PT that converts the monitored voltage to the control voltage. Default is 60. If the capacitor is Wye, the 1st phase line-to-neutral voltage is monitored. Else, the line-to-line voltage (1st - 2nd phase) is monitored.

**terminal** Number of the terminal of the circuit element to which the CapControl is connected. 1 or 2, typically. Default is 1.

**VBus** Name of bus to use for voltage override function. Default is bus at monitored terminal. Sometimes it is useful to monitor a bus in another location to emulate various DMS control algorithms.

**Vmax** Maximum voltage, in volts. If the voltage across the capacitor divided by the PTRATIO is greater than this voltage, the capacitor will switch OFF regardless of other control settings. Default is 126 (goes with a PT ratio of 60 for 12.47 kV system).

**Vmin** Minimum voltage, in volts. If the voltage across the capacitor divided by the PTRATIO is less than this voltage, the capacitor will switch ON regardless of other control settings. Default is 115 (goes with a PT ratio of 60 for 12.47 kV system).

**VoltOverride** {Yes | No} Default is No. Switch to indicate whether VOLTAGE OVERRIDE is to be considered. Vmax and Vmin must be set to reasonable values if this property is Yes.

## REGCONTROL OBJECT

This control is designed to emulate a standard utility voltage regulator or LTC control. It is attached to a particular winding of a transformer as the winding to monitor. It generally also adjusts the taps in that winding but could also be directed to control the taps in another winding.

The control has line drop compensator modeling by setting the R, X, CTprim, and ptratio properties. The control can also monitor the voltage at a remote bus to emulate various Smart Grid devices. This is a useful function for performing volt/var optimization.

To understand how regulator controls work, refer to W. H. Kersting's book on Distribution System Modeling. A simple example of a regulator on a 12.47 kV system:

```
New RegControl.Reg1 Transformer=T1 Winding=2 Vreg=122 band=3 ptratio=60
```

With a line-drop compensator, the definition might look like

```
New RegControl.Reg1 Transformer=T1 Winding=2 Vreg=122 band=3
~ ptratio=60 CTprim=300  R=2  X=0
```

Controlling a bus at the end of the feeder to 118 V:

```
New RegControl.Reg1 Transformer=T1 Winding=2 Vreg=118 band=2 bus=MyEndBus
```

**transformer** Name of Transformer element to which the RegControl is connected. Do not specify the full object name; "Transformer" is assumed for the object class. Example:

Transformer=Xfmr1

**winding** Number of the winding of the transformer element that the RegControl is monitoring. 1 or 2, typically. Side Effect: Sets TAPWINDING property to the same winding.

**vreg** Voltage regulator setting, in VOLTS, for the winding being controlled. Multiplying this value times the ptratio should yield the voltage across the WINDING of the controlled transformer. Default is 120.0

**band** Bandwidth in VOLTS for the controlled bus (see help for ptratio property). Default is 3.0

**delay** Time delay, in seconds, from when the voltage goes out of band to when the tap changing begins. This is used to determine which regulator control will act first. Default is 15. You may specify any floating point number to achieve a model of whatever condition is necessary.

**ptratio** Ratio of the PT that converts the controlled winding voltage to the regulator voltage. Default is 60. If the winding is Wye, the line-to-neutral voltage is used. Else, the line-to-line voltage is used.

**CTprim** Rating, in Amperes, of the primary CT rating for converting the line amps to control amps.The typical default secondary ampere rating is 0.2 Amps (check with manufacturer specs).

**R** R setting on the line drop compensator in the regulator, expressed in VOLTS.

**X** X setting on the line drop compensator in the regulator, expressed in VOLTS.

**PTphase** For multi-phase transformers, the number of the phase being monitored or one of {

MAX | MIN} for all phases. Default=1. Must be less than or equal to the number of phases. Ignored for regulated bus.

**tapwinding** Winding containing the actual taps, if different than the WINDING property. Defaults to the same winding as specified by the WINDING property.

**bus** Name of a bus (busname.nodename) in the system to use as the controlled bus instead of the bus to which the transformer winding is connected or the R and X line drop compensator settings. Do not specify this value if you wish to use the line drop compensator settings. Default is null string. Assumes the base voltage for this bus is the same as the transformer winding base specified above. Note: This bus (1-phase) WILL BE CREATED by the regulator control upon SOLVE if not defined by some other device. You can specify the node of the bus you wish to sample (defaults to 1). If specified, the RegControl is redefined as a 1-phase device since only one voltage is used.

**debugtrace** {Yes | No* } Default is no. Turn this on to capture the progress of the regulator model for each control iteration. Creates a separate file for each RegControl named "REG_name.CSV".

**EventLog** {Yes/True* | No/False} Default is YES for regulator control. Log control actions to Eventlog.

**inversetime** {Yes | No* } Default is no. The time delay is adjusted inversely proportional to the amount the voltage is outside the band down to 10%.

**maxtapchange** Maximum allowable tap change per control iteration in STATIC control mode. Default is 16.

Set this to 1 to better approximate actual control action.

Set this to 0 to fix the tap in the current position.

**revband** Bandwidth for operating in the reverse direction.

**revDelay** Time Delay in seconds (s) for executing the reversing action once the threshold for reversing has been exceeded. Default is 60 s.

**reversible** {Yes |No*} Indicates whether or not the regulator can be switched to regulate in the reverse direction. Default is No.Typically applies only to line regulators and not to LTC on a substation transformer.

**revNeutral** {Yes | No*} Default is no. Set this to Yes if you want the regulator to go to neutral in the reverse direction.

**revR** R line drop compensator setting for reverse direction.

**revThreshold** kW reverse power threshold for reversing the direction of the regulator. Default is 100.0 kw.

**revvreg** Voltage setting in volts for operation in the reverse direction.

**revX** X line drop compensator setting for reverse direction.

**tapdelay** Delay in sec between tap changes. Default is 2. This is how long it takes between changes after the first change.

**TapNum** An integer number indicating the tap position that the controlled transformer winding tap position is currently at, or is being set to. If being set, and the value is outside the range of the transformer min or max tap, then set to the min or max tap position as appropriate. Default is 0, which is the neutral tap. A conventional 32-

step regulator has a range from -16 (Lower) to 16(Raise). With tap 0, there are actually 33 taps. You can query the tap position as follows:

**? RegControl.MyRegulator.TapNum**

The result will go into the Result window or interface.

**vlimit** Voltage Limit for bus to which regulated winding is connected (e.g. first customer). Default is 0.0. Set to a value greater then zero to activate this function.

# Meter Elements

## 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 may use trapezoidal integration (system option), 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, for example for a 22 kV system:

```
Hour
"kWh"
"kvarh"
"MaxkW"
"MaxkVA"
"ZonekWh"
"Zonekvarh"
"ZoneMaxkW"
"ZoneMaxkVA"
"OverloadkWhNormal"
"OverloadkWhEmerg"
"LoadEEN"
"LoadUE"
"ZoneLosseskWh"
"ZoneLosseskvarh"
"ZoneMaxkWLosses"
"ZoneMaxkvarLosses"
"LoadLosseskWh"
"LoadLosseskvarh"
"NoLoadLosseskWh"
"NoLoadLosseskvarh"
"MaxkWLoadLosses"
"MaxkWNoLoadLosses"
"LineLosses"
"TransformerLosses"
"LineModeLineLosses"
"ZeroModeLineLosses"
```

```
"3-phaseLineLosses"
"1-and2-phaseLineLosses"
"GenkWh"
"Genkvarh"
"GenMaxkW"
"GenMaxkVA"
"22kVLosses"
"Aux1"
"Aux6"
"Aux11"
"Aux16"
"Aux21"
"Aux26"
"22kVLineLoss"
"Aux2"
"Aux7"
"Aux12"
"Aux17"
"Aux22"
"Aux27"
"22kVLoadLoss"
"Aux3"
"Aux8"
"Aux13"
"Aux18"
"Aux23"
"Aux28"
"22kVNoLoadLoss"
"Aux4"
"Aux9"
"Aux14"
"Aux19"
"Aux24"
"Aux29"
"22kVLoadEnergy"
"Aux5"
"Aux10"
"Aux15"
"Aux20"
"Aux25"
"Aux30"
```

Registers are frequently added for various purposes. You can view the present meters simply by solving and taking a sample. Then execute a **Show Meters** command. The Aux registers listed in the example above are used for keep track of losses at up to 7 different voltage levels in the meter zone.

### *Meter 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:

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).

*(Note: These rules imply that the meter should usually be placed at the source end of the circuit element it is monitoring.)*

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
- An 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.

In order to apply the Reconductor command, both lines must be in the same meter zone. The upline/downline orientation of the line segments is established when the zones are built. Otherwise, the DSS has no concept of radiality.

The Parent property available in the Lines interface in the COM interface is set when the zone is established. This allows users to programmatically trace back up the circuit toward the Energymeter. The total number of customers served downline is determined during the establishment of the meter zone.

### *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 51.

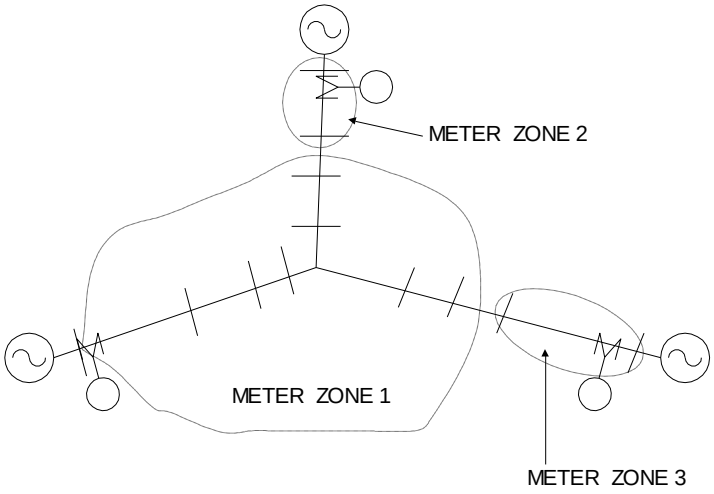



**Figure 51. Default Meter Zones for a Simple Network**

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 52.

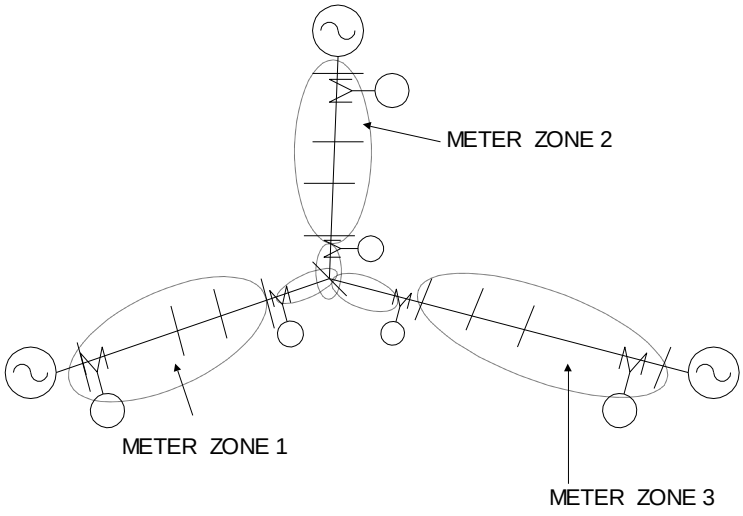



**Figure 52. Using Additional Meters to Control the Definition of Meter Zones**

## *Sampling*

The sampling algorithms are as follows:

Local Energy and Power Values: 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.

Losses in Zone: Accumulate the kW losses in each power delivery element in the zone.

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

Overload Energy in Zone: 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.

EEN and UE in Zone: 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:

The voltage at the load bus is below minimum ratings

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 (1$^{st}$ 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, normally the source end.

**Action**= Optional action to execute. One of

**Clear** = reset all registers to zero

**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.

**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).

(**C**)ombined: (default) Load UE or EEN are computed from both overload and undervoltage criteria.

(**V)**oltage: Load UE/EEN are computed only from the undervoltage criteria.

Example: option=(E, R, C)

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.

**3PhaseLosses** = {Yes | No} Default is YES. Compute Line losses and segregate by 3-phase and other (1- and 2-phase) line 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 of the buses are assigned BEFORE defining the EnergyMeter to ensure that it will automatically pick up the voltage bases. Or, issue the CalcVoltageBases command after defining the EnergyMeter. Each voltage base has four(4) loss registers: total, line, load, and no-load, respectively. There are sufficient registers to report losses in five (5) different voltage levels.

**BaseFreq** = Base frequency for ratings.

**Enabled** = {Yes|No or True|False} Indicates whether the element is enabled.

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

## MONITOR OBJECT

A monitor is a benign circuit element that is connected to 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. Other quantities may be saved depending on the setting of the Mode property. The data are saved in a file stream (a separate one for each monitor) at the conclusion of each step 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 generally save two basic types of quantities: 1) Voltage and current; 2) Power. Can also save other selected quantities as defined below. The Mode codes are defined as follows:

0: Standard mode - V and I, each phase, complex

1: Power each phase, complex (kw and kvars)

2: Transformer taps (connect Monitor to a transformer winding)

3: State variables (connect Montor to a PCElement)

4: Flicker level and severity index (Pst) for voltages. No adders apply. Flicker level at simulation time step, Pst at 10-minute time step.

5: Solution variables (Iterations, etc). Normally, these would be actual phasor quantities from solution.

6: Capacitor Switching (connect Monitor to a capacitor, this mode records all the steps defined for the capacitor in separate channels)

7: Storage state variables (Storage devices only). Variables include kWhstored (SOC), kW, kvar, kWlosses, inverter losses, idling losses, efficiency, etc. Name of the variable is in the header for the channel, which may be retrieved in the Show command or through Header property of the COM interface.

8: All Transformer winding currents (Transformer or AutoTransformer elements). Useful for getting delta winding currents rather than the usual terminal currents.

9: Losses (watts, vars) of the monitored element.

10: All Transformer winding voltages (Transformer or AutoTransformer elements). Useful for getting voltages across actual windings rather than just the node

voltages.

Normally, the values would be actual phasor quantities, but for many modes, the mode number can be combined with the adders below to obtain other quantities.

11: All terminal node voltages and line currents of monitored device.

+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 property forces clearing of the monitor's buffer, or saving to disk.

The name of the file created appears in the Result window.

# 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.

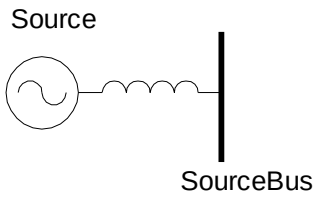



**Figure 53. Default Circuit**

The default values for the source are (see Vsource object definition):

115 kV

3000 MVA short circuit

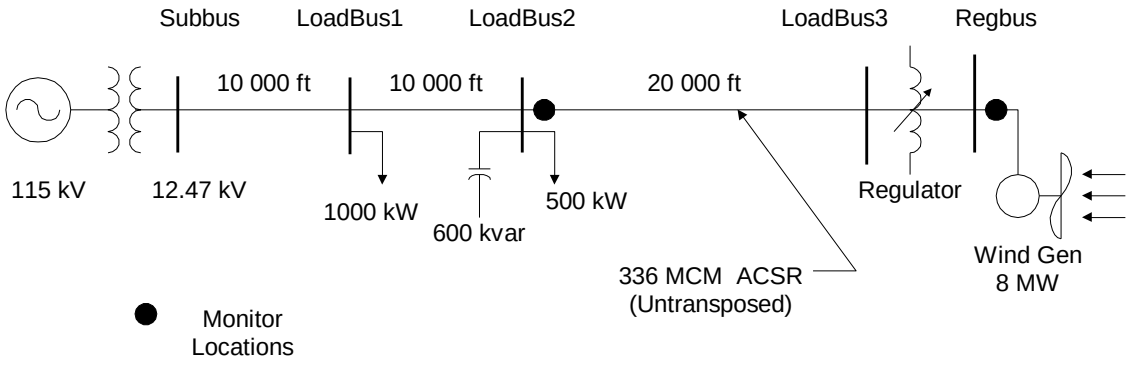
Thus, the circuit can be immediately solved, albeit with a trivial result.

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

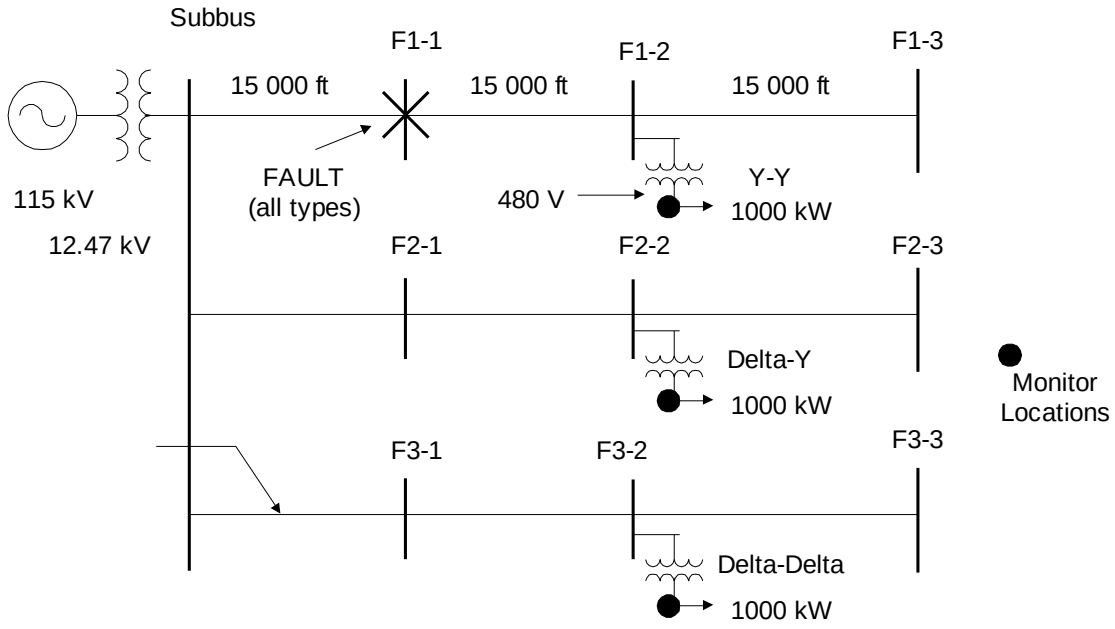



### *DSS Circuit Description Script*

```
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
new load.load2 bus1=Genbus2 phases=3 kv=0.48 kw=1000.0 pf=0.88 model=1
new load.load3 bus1=Genbus3 phases=3 kv=0.48 kw=1000.0 pf=0.88 model=1
conn=delta


! 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

While a lot can be done with the standard text scripting interface, knowledgable users can open up an entirely new world of applications by learning to effectively use the COM interface. Algorithms can be implemented in another computer program and used to drive the OpenDSS to do something that is not currently implemented within it. An example would be a specific optimization algorithm. Only a few simple ones are currently implemented inside OpenDSS, but external ones of great complexity can be written. The algorithms would rely on the OpenDSS to represent the behaviour of the distribution system while adjusting whatever variables are being optimized.

One good use of the COM interface is to create looping scripts. There is no looping capabilities in the OpenDSS scripting language. The closest thing is the Next command, which can simplify scripts that increment time. Even if there were a looping capability, it would execute relatively slowly because it would be interpreted. Looping scripts are relatively easy to write in other languages and they generally run quickly.

Documentation of the OpenDSS COM interfaces have lagged behind the implementations. This is one reason the program has been made open source. If the use of an interface is not obvious from its name or with experimentation, the user can view the source code and determine what the interface function or property will do when invoked. The code behind the implementation of the interfaces may be found in files named *Implxxxx*.pas in the Source\DLL folder, where *xxxx* is the name of the interface. Some interface documentation may be found in the DOC folder, with the caveat that even these might be out of date since functions (methods) and properties may be added at any time. The Wiki has additional information.

There are several choices for tools that can provide documentation of the type library (TLB) for the COM interface. Most users will have at least one of the Microsoft Office programs on their computer. Since COM is a Microsoft invention, the VBA developer support for these programs is pretty good for exposing the classes, properties, and methods in the TLB to the user. They have code completion, which is useful for developing code that must access the OpenDSS COM interfaces. Software like MATLAB is not generally as helpful. Some users will lay out their code in VBA first and then paste it into the MATLAB editor. Another strategy is to obtain one of a number of TLB documenters and execute it on the OpenDSSEngine.DLL file.

We will document the DSS interface here and then provide some examples of writing code for driving the OpenDSS program. The DSS interface is the only one that you must instantiate to start the program. There are several other interfaces, all created when the Start function of the DSS interface is executed.

The most important of the other interfaces is probably the Circuit class interface. This is documented in the file named *OpenDSS Circuit Interface*. Both a doc file and a pdf file are on the OpenDSS website.

Besides setting variables for the DSS object itself, it is common to also define specific variables for the Text interface and the Circuit interface. For example, in VBA:

```
Option Explicit
```

```
' Declare variables
Public DSSobj As OpenDSSengine.DSS
Public DSSText As OpenDSSengine.Text
Public DSSCircuit As OpenDSSengine.Circuit
…

' Create a new instance of the DSS
    Set DSSobj = New OpenDSSengine.DSS
    DSSObj.Start(0)
' Assign a variable to the Text interface for easier access
    Set DSSText = DSSobj.Text
' Assign a variable to the Circuit interface for easier access
    Set DSSCircuit = DSSobj.ActiveCircuit
```

## DSS INTERFACE

Assume DSSObj is an object defined as type OpenDSSEngine.DSS. Then the methods and properties of the DSS interface are as described below:

### *bOK = DSSObj.Start(0)*

Start is a function that returns a Boolean value indicating success in starting the OpenDSS.

(Insert "DSSObj." in front of each property name below.)

### *ActiveCircuit*

Returns a type Circuit interface that points to the active circuit. This is one of the most-used interface and a variable in the user's program is often set to this interface to simplify programming. For example, in VBA:

```
Set DSSCircuit = DSSObj.ActiveCircuit
```

### *ActiveClass*

Returns a type ActiveClass interface to the class of the active element in the present problem. This interface is basically for reporting the names of the classes in the present version of the program.

### *AllowForms*

This property is used to suppress message forms from popping up during the execution of the user's program. Set this to False to suppress forms. Cannot be set back to True.

### *Circuits*

This interface is provided for future use when the program can support more than one circuit object in memory at a time. Currently, this is not allowed.

### *Classes*

Variant array containing a list of intrinsic OpenDSS class names.

### *ClearAll*

Executing this function clears all circuit parameters from memory.

### *DataPath*

Read/write string describing the path name for OpenDSS data.

### *DefaultEditor*

Read only property returning the path name of the Editor being used by OpenDSS.

### *DSSProgress*

Returns interface to the Progress Bar,

### *Error*

Returns the Error interface, which can give the error number and error description

### *Events*

Returns interface to DSSEvents, which is used to manage Windows events raised within OpenDSS.

### *Executive*

Returns DSS_Executive interface, which gives access to the names of all OpenDSS commands, options, and related help strings. This is the same information that is displayed in the standard OpenDSS Help form from the Help command.

### *NewCircuit*

This function makes a new circuit object and returns the Circuit interface to the active Circuit.

### *NumCircuits*

Returns the number of circuit defined. At present, either 0 or 1.

### *Reset*

This function executes the Reset command.

### *SetActiveClass(Classname as String)*

This function sets the active class to the class specified by Classname. Same as SetActiveClass function in Circuit interface.

### *ShowPanel*

Shows a version of the main control panel for OpenDSS.

### *Text*

Returns the Text interface through which standard OpenDSS text command may be sent and results messages retrieved. This is generally a commonly-used interface and most user-written programs should have a variable assigned to this interface to aid coding. See examples that follow this section. The Text interface has two properties: Command and Result.

### *UserClasses*

Variant array of strings with a list of names of user-written classes.

### *Version*

Read-only property returning a string describing the present version of the program. This is useful for making sure the version being accessed is compatible with the version the user-written program expects.

# EXAMPLE MS EXCEL VBA CODE FOR DRIVING THE COM INTERFACE

The code below was extracted from a simple Microsoft Excel example for driving the OpenDSS for compiling an arbitrary circuit script file.

## *Declarations*

This section of the code declares some public variable for accessing the OpenDSS object and directly accessing two of the interfaces once the OpenDSS is instantiated. The DSSobj variable is required. DSSText and DSSCircuit are optional but allow for shorthand access to the Text interface and the active circuit interface.

```
Option Explicit

Public DSSobj As OpenDSSengine.DSS
Public DSSText As OpenDSSengine.Text
Public DSSCircuit As OpenDSSengine.Circuit
```

## *Startup*

This subroutine first instantiates the OpenDSS object, setting it to the DSSobj variable. The DSSText variable can be set at this time, so that is done. Note that the DSSCircuit variable cannot be set because no circuit exists yet. Then the OpenDSS is started by invoking the Start method. The argument is ignored in this version. This initializes the OpenDSS and gets it ready for doing its work.

```
Public Sub StartDSS()

' Create a new instance of the DSS
    Set DSSobj = New OpenDSSengine.DSS

' Assign a variable to the Text interface for easier access
    Set DSSText = DSSobj.Text

' Start the DSS
    If Not DSSobj.Start(0) Then MsgBox "DSS Failed to Start"

End Sub
```

## *Compiling the Circuit*

This subroutine illustrates using the Text interface Command property to compile the circuit script file the user has specified. The filename is specified in a cell in the workbook labelled “fname”.

When the Compile has completed, there will be an active circuit and the current directory will have been switched to the script file directory. Unless changed in the script, this is where the results files will reside. Now that there is an active circuit, the DSSCircuit variable can be assigned to it.

```
Public Sub LoadCircuit(fname As String)

' Always a good idea to clear the DSS when loading a new circuit
    DSSText.Command = "clear"

' Compile the script in the file listed under "fname" cell on the main form
    DSSText.Command = "compile " + fname
```

```
' The Compile command sets the current directory the that of the file
' Thats where all the result files will end up.

' Assign a variable to the Circuit interface for easier access
    Set DSSCircuit = DSSobj.ActiveCircuit

End Sub
```

## *Solving the Circuit*

Solving can be accomplished by sending "solve" to the DSSText.Command interface. However, this example illustrates directly accessing the Solve method in the Solution object of the active circuit. This shows the expanded code. Since the DSSCircuit interface has been set, one could also invoke the Solve method by "`DSSCircuit.Solution.Solve`" which is a bit shorter.

```
Public Sub Solve()

' Execute the Solve method in the Solution interface of the active circuit
    DSSobj.ActiveCircuit.Solution.Solve

End Sub
```

## *Bringing Results into the Spreadsheet*

There are several ways to bring results from inside the OpenDSS into a spreadsheet. One obvious method is to export a CSV file and open the file in MS Excel. The two subroutines below illustrate how to do a couple of things programmatically. The first function loads the sequence voltages for all buses into a worksheet. The second loads the complex per unit voltages into another worksheet.

These routines take advantage of the ability to transfer variant arrays across the COM interface that is inherent in VBA and supported in OpenDSS. With one simple line of code, such as:

```
V = DSSBus.SeqVoltages
```

A whole array of values are brought into the program to be exploited however the user desires. The SeqVoltages property is associated with the Bus interface. When invoked, it causes the sequence voltages to be computed for the bus for the most recent solution and marshalled into a variant array of doubles. (The OpenDSS does not work in symmetrical components, so these values must be computed post-solution.)

In these examples, the values are simply deposited into certain cells along with the bus name. The bus name is extracted from the DSSCircuit.ActiveBus interface.

```
Public Sub LoadSeqVoltages()

' This Sub loads the sequence voltages onto Sheet1 starting in Row 2

    Dim DSSBus As OpenDSSengine.Bus
    Dim iRow As Long, iCol As Long, i As Long, j As Long
    Dim V As Variant
    Dim WorkingSheet As Worksheet

    Set WorkingSheet = Sheet1   'set to Sheet1 (target sheet)
```

```
    iRow = 2
    For i = 1 To DSSCircuit.NumBuses  ' Cycle through all buses

        Set DSSBus = DSSCircuit.Buses(i)  ' Set ith bus active

    ' Bus name goes into Column 1
        WorkingSheet.Cells(iRow, 1).Value = DSSCircuit.ActiveBus.Name

    ' Load sequence voltage magnitudes of active bus into variant array
        V = DSSBus.SeqVoltages

    ' Put the variant array values into Cells
    ' Use Lbound and UBound because you don't know the actual range
        iCol = 2
        For j = LBound(V) To UBound(V)
            WorkingSheet.Cells(iRow, iCol).Value = V(j)
            iCol = iCol + 1
        Next j
       iRow = iRow + 1
    Next i

End Sub


Public Sub LoadVoltageMags()

' This Sub loads the per unit complex voltages onto Sheet3 starting in Row 2

    Dim DSSBus As OpenDSSengine.Bus
    Dim iRow As Long, iCol As Long, i As Long, j As Long
    Dim V As Variant
    Dim WorkingSheet As Worksheet

    Set WorkingSheet = Sheet3   'For Example

    iRow = 2
    For i = 1 To DSSCircuit.NumBuses    ' Cycle through all buses

        Set DSSBus = DSSCircuit.Buses(i)  ' Set i-th bus active

    ' Bus name goes into Column 1
        WorkingSheet.Cells(iRow, 1).Value = DSSCircuit.ActiveBus.Name

    ' Loads pu voltages (complex) at active bus as variant array of doubles
        V = DSSBus.puVoltages

    ' Put values in Variant array into cells in sequence provided by DSS
        iCol = 2
        For j = LBound(V) To UBound(V)
            WorkingSheet.Cells(iRow, iCol).Value = V(j)
            iCol = iCol + 1
        Next j

       iRow = iRow + 1
    Next i

End Sub
```

In MS Excel VBA, you may easily view the other properties and methods available through the

various interfaces in the OpenDSSEngine by including the OpenDSSEngine on the Tools | References list and displaying the Object Browser (see Figure 54 and Figure 55). A simple one-line help string is available for each item.

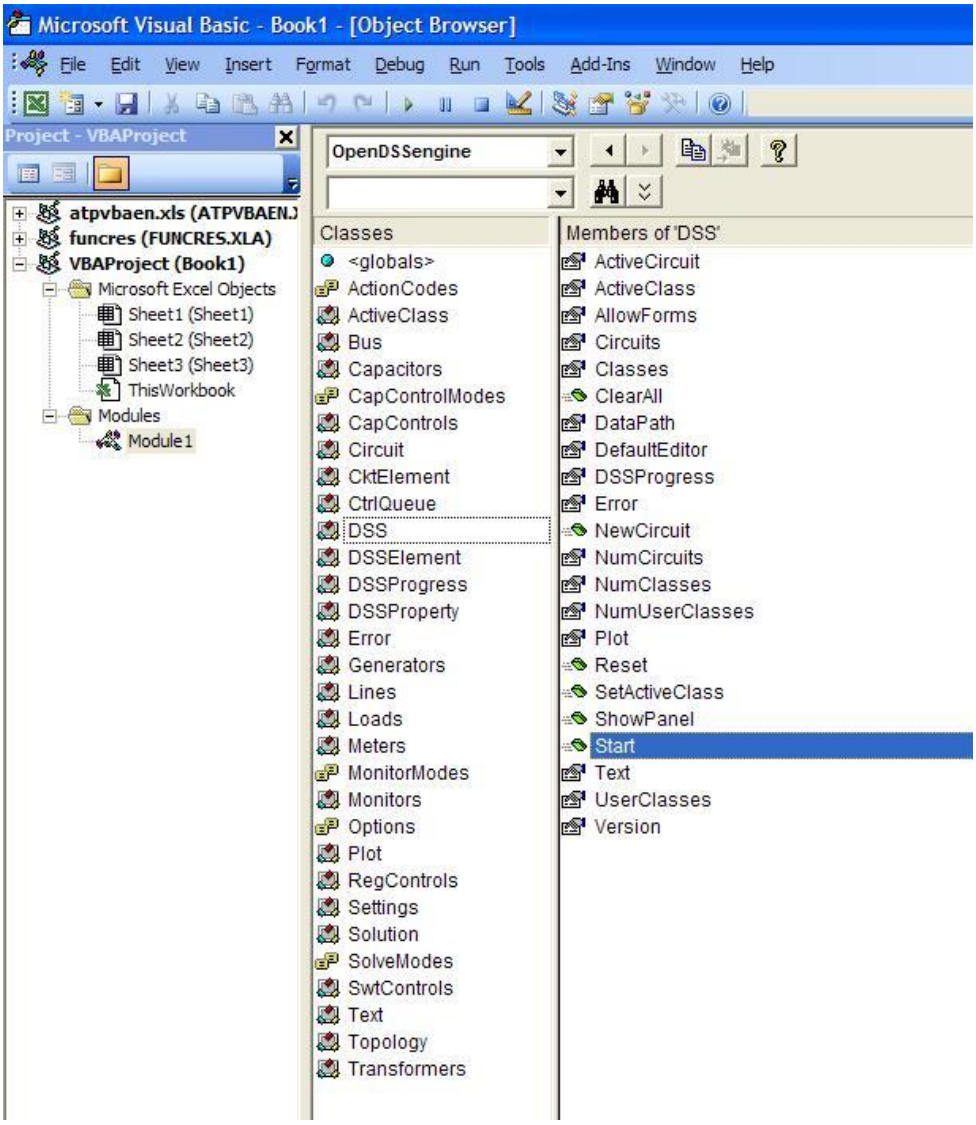



**Figure 54. OpenDSSEngine COM interface as seen with MS Excel VBA**

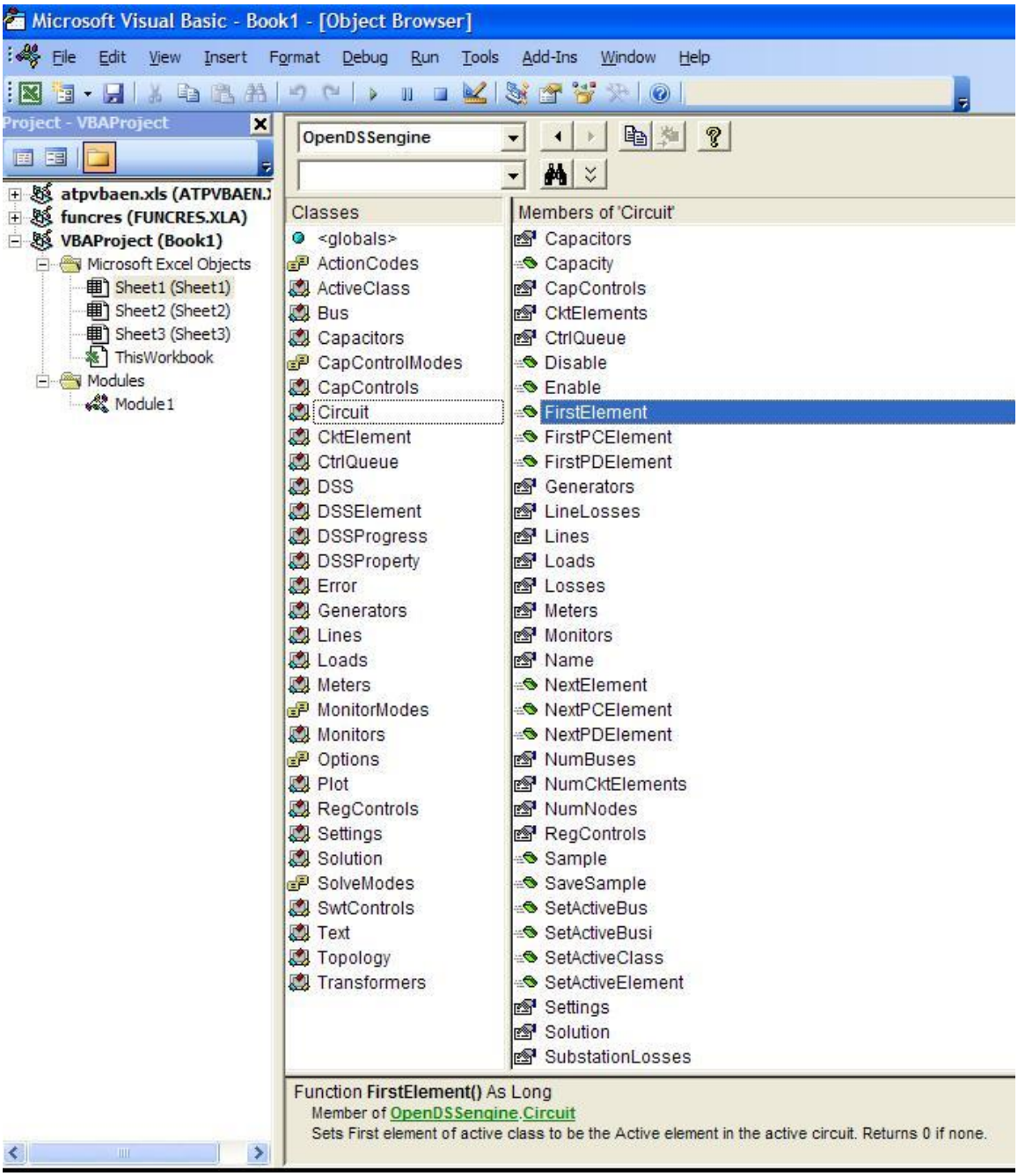



**Figure 55. VBA Object Browser listing of the OpenDSS Circuit Interface**

## EXAMPLE: OBTAINING ALL CAPACITOR NAMES AND LOCATIONS IN MICROSOFT EXCEL VBA

This Excel subroutine will cycle through all the Capacitor objects in the circuit and put the names and bus connections in an Excel spreadsheet. It uses Ctk24 from the EPRI Test Circuits (on the OpenDSS site).

```
Attribute VB_Name = "Module1"
Option Explicit

Public DSSObj As OpenDSSengine.DSS
Public DSSText As OpenDSSengine.Text

Public Sub LoadInCapLocations()

    Set DSSObj = New OpenDSSengine.DSS
    Set DSSText = DSSObj.Text

    If DSSObj.Start(0) Then

        DSSText.Command = "Compile
[C:\Users\prdu001\OpenDSS\EPRITestCircuits\ckt24\master_ckt24.dss]"
        DSSObj.ActiveCircuit.Solution.Solve

        DSSObj.ActiveCircuit.SetActiveClass "Capacitor"

        Dim MyCap As Integer, iRow As Integer
        Dim BusNameArray As Variant

        ActiveSheet.Cells(1, 1).Value = DSSObj.ActiveClass.ActiveClassName
        ActiveSheet.Cells(1, 2).Value = "Bus1"
        ActiveSheet.Cells(1, 3).Value = "Bus2"

        iRow = 2

        MyCap = DSSObj.ActiveClass.First
        Do While MyCap > 0
            ActiveSheet.Cells(iRow, 1).Value = DSSObj.ActiveClass.Name
            BusNameArray = DSSObj.ActiveCircuit.ActiveCktElement.BusNames  '
gets all bus names for this device
            ActiveSheet.Cells(iRow, 2).Value = BusNameArray(0)    ' Bus1
            ActiveSheet.Cells(iRow, 3).Value = BusNameArray(1)    ' Bus2
            iRow = iRow + 1
            MyCap = DSSObj.ActiveClass.Next
        Loop

    End If

    Set DSSObj = Nothing     ' Be a good citizen

End Sub
```

## EXAMPLE: GIC CALCULATION

The example system represents a hypothetical 20 bus EHV network consisting of 500 kV and 345 kV lines and transformers[4]. Figure 56 shows a single-line diagram of the network. The network includes single transmission lines as well as some that occupy the same transmission corridor. The substations feature both conventional transformers and autotransformers. Also included are series (series capacitors) and neutral connected GIC blocking devices.

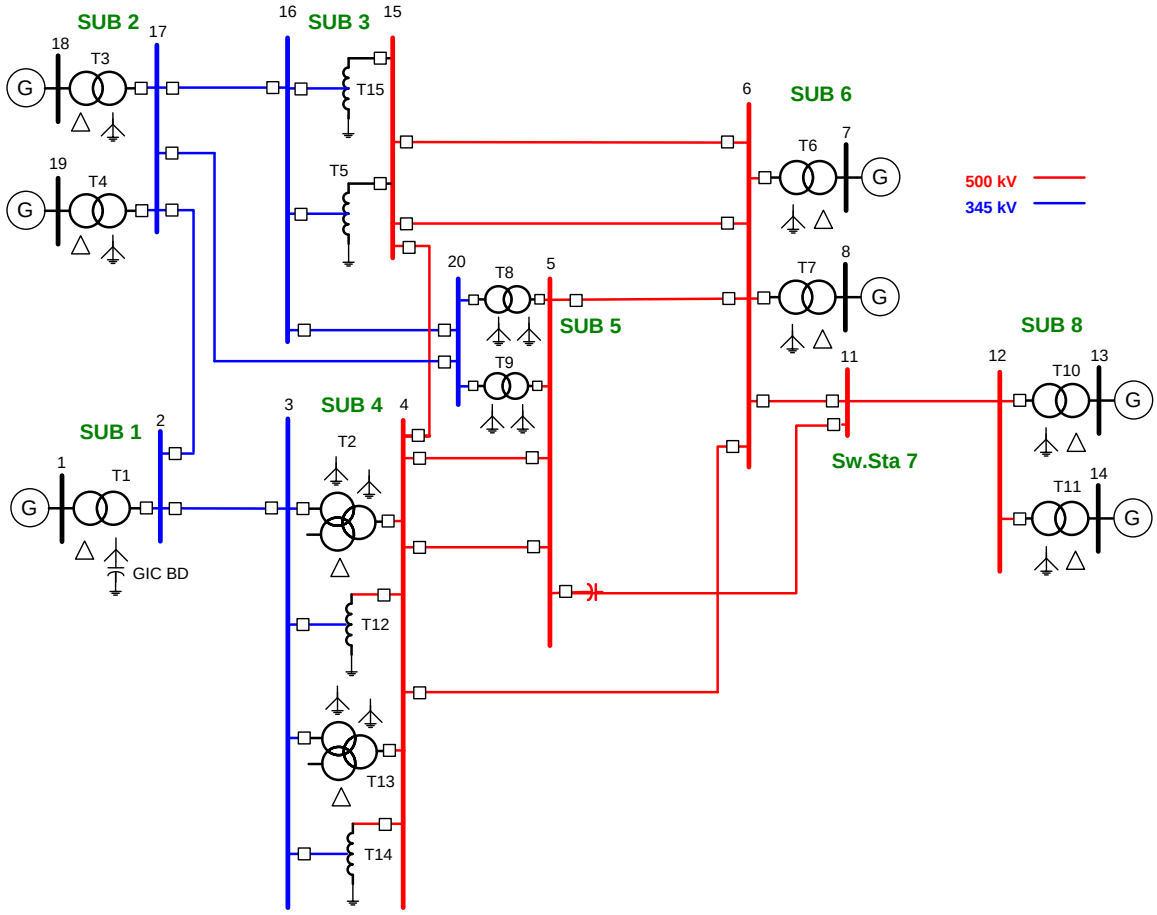



**Figure 56. One Line Diagram of Example GIC System**

Data for the system described in Figure 56 are provided in Tables 1-3.

[4] R. Horton, D.H. Boteler, T.J. Overbye, R.J. Pirjola, R.Dugan, "A Test Case for the Calculation of Geomagnetically Induced Currents," *IEEE Transactions on Power Systems*, October 2012.

**Table 2.**
**GIC Example Substation Location and Grid Resistance**

| Name | Latitude | Longitude | Grounding Resistance (Ohm) |
|---|---|---|---|
| Sub 1 | 33.613499 | -87.373673 | 0.2 |
| Sub 2 | 34.310437 | -86.365765 | 0.2 |
| Sub 3 | 33.955058 | -84.679354 | 0.2 |
| Sub 4 | 33.547885 | -86.074605 | 1.0 |
| Sub 5 | 32.705087 | -84.663397 | 0.1 |
| Sub 6 | 33.377327 | -82.618777 | 0.1 |
| Sub 7 | 34.252248 | -82.836301 | N/A |
| Sub 8 | 34.195574 | -81.098002 | 0.1 |

**Table 3.**
**GIC Example Transmission Line Data**

| Line | From Bus | To Bus | Voltage (kV-LL) | Length (miles) | Resistance (ohm/phase) |
|---|---|---|---|---|---|
| 1 | 2 | 3 | 345 | 77.18 | 3.512 |
| 2 | 2 | 17 | 345 | 77.47 | 3.525 |
| 3 | 15 | 4 | 500 | 87.51 | 1.986 |
| 4 | 17 | 16 | 345 | 102.54 | 4.665 |
| 5 | 4 | 5 | 500 | 103.31 | 2.345 |
| 6 | 4 | 5 | 500 | 103.31 | 2.345 |
| 7 | 5 | 6 | 500 | 131.05 | 2.975 |
| 8 | 5 | 11 | 500 | 154.57 | 3.509 |
| 9 | 6 | 11 | 500 | 63.59 | 1.444 |
| 10 | 4 | 6 | 500 | 205.57 | 4.666 |
| 11 | 15 | 6 | 500 | 128.81 | 2.924 |
| 12 | 15 | 6 | 500 | 128.81 | 2.924 |
| 13 | 11 | 12 | 500 | 102.39 | 2.324 |
| 14 | 16 | 20 | 345 | 88.98 | 4.049 |
| 15 | 17 | 20 | 345 | 152.53 | 6.940 |

**Table 4.**
**GIC Example Transformer Data**

| Name | Type | Resistance W1 (Ohms/phase) | Bus No. | Resistance W2 (Ohms/phase) | Bus No. |
|---|---|---|---|---|---|
| T1 | GSU w/ GIC BD | 0.1 | 2 | N/A | 1 |
| T2 | GY-GY-D | 0.2 | 4 | 0.1 | 3 |
| T3 | GSU | 0.1 | 17 | N/A | 18 |
| T4 | GSU | 0.1 | 17 | N/A | 19 |
| T5 | Auto | 0.04 | 16 | 0.06 | 15 |
| T6 | GSU | 0.15 | 6 | N/A | 7 |
| T7 | GSU | 0.15 | 6 | N/A | 8 |
| T8 | GY-GY | 0.04 | 5 | 0.06 | 20 |
| T9 | GY-GY | 0.04 | 5 | 0.06 | 20 |
| T10 | GSU | 0.1 | 12 | N/A | 13 |
| T11 | GSU | 0.1 | 12 | N/A | 14 |
| T12 | Auto | 0.04 | 4 | 0.06 | 3 |
| T13 | GY-GY-D | 0.2 | 4 | 0.1 | 3 |
| T14 | Auto | 0.04 | 4 | 0.06 | 3 |
| T15 | Auto | 0.04 | 15 | 0.06 | 16 |

## *OpenDSS Script for Example System*

The following scripts can be used to model the example system in OpenDSS with an Eastward electric field of 1 V/km. See Examples\GICExample on the OpenDSS Website.

**Transmission Lines**

Note: Lines appear wrapped in this document due to Word limitations, but are all one OpenDSS statement in the actual DSS script file.

```
!GIC Line Data
New GICLine.1-Bus2-Bus3 bus1=2 bus2=3 R=3.512 Lat1=33.613499 Lon1=-87.373673
Lat2=33.547885 Lon2=-86.074605 EE=1.00 EN=0.00
New GICLine.2-Bus2-Bus17 bus1=2 bus2=17 R=3.525 Lat1=33.613499 Lon1=-87.373673
Lat2=34.310437 Lon2=-86.365765 EE=1.00 EN=0.00
New GICLine.3-Bus15-Bus4 bus1=15 bus2=4 R=1.986 Lat1=33.955058 Lon1=-84.679354
Lat2=33.547885 Lon2=-86.074605 EE=1.00 EN=0.00
New GICLine.4-Bus17-Bus16 bus1=17 bus2=16 R=4.665 Lat1=34.310437 Lon1=-
86.365765 Lat2=33.955058 Lon2=-84.679354 EE=1.00 EN=0.00
New GICLine.5-Bus4-Bus5 bus1=4 bus2=5 R=2.345 Lat1=33.547885 Lon1=-86.074605
Lat2=32.705087 Lon2=-84.663397 EE=1.00 EN=0.00
New GICLine.6-Bus4-Bus5 bus1=4 bus2=5 R=2.345 Lat1=33.547885 Lon1=-86.074605
Lat2=32.705087 Lon2=-84.663397 EE=1.00 EN=0.00
New GICLine.7-Bus5-Bus6 bus1=5 bus2=6 R=2.975 Lat1=32.705087 Lon1=-84.663397
Lat2=33.377327 Lon2=-82.618777 EE=1.00 EN=0.00
New GICLine.8-Bus5-Bus11 bus1=5 bus2=11 C=32.0 R=3.509 Lat1=32.705087 Lon1=-
```

```
84.663397 Lat2=34.252248 Lon2=-82.836301 EE=1.00 EN=0.00
New GICLine.9-Bus6-Bus11 bus1=6 bus2=11 R=1.444 Lat1=33.377327 Lon1=-82.618777
Lat2=34.252248 Lon2=-82.836301 EE=1.00 EN=0.00
New GICLine.10-Bus4-Bus6 bus1=4 bus2=6 R=4.666 Lat1=33.547885 Lon1=-86.074605
Lat2=33.377327 Lon2=-82.618777 EE=1.00 EN=0.00
New GICLine.11-Bus15-Bus6 bus1=15 bus2=6 R=2.924 Lat1=33.955058 Lon1=-84.679354
Lat2=33.377327 Lon2=-82.618777 EE=1.00 EN=0.00
New GICLine.12-Bus15-Bus6 bus1=15 bus2=6 R=2.924 Lat1=33.955058 Lon1=-84.679354
Lat2=33.377327 Lon2=-82.618777 EE=1.00 EN=0.00
New GICLine.13-Bus11-Bus12 bus1=11 bus2=12 R=2.324 Lat1=34.252248 Lon1=-
82.836301 Lat2=34.195574 Lon2=-81.098002 EE=1.00 EN=0.00
New GICLine.14-Bus16-Bus20 bus1=16 bus2=20 R=4.049 Lat1=33.955058 Lon1=-
84.679354 Lat2=32.705087 Lon2=-84.663397 EE=1.00 EN=0.00
New GICLine.15-Bus17-Bus20 bus1=17 bus2=20 R=6.940 Lat1=34.310437 Lon1=-
86.365765 Lat2=32.705087 Lon2=-84.663397 EE=1.00 EN=0.00
```

**Transformers**

Note: Lines appear wrapped in this document due to Word limitations, but are all one OpenDSS statement in the actual DSS script file.

```
New GICTransformer.T1 busH=2 busNH=2.4.4.4 R1=0.1 type=GSU_
New GICTransformer.T2 busH=4 busNH=4.4.4.4 busX=3 busNX=4.4.4.4 R1=0.2 R2=0.1
type=YY
New GICTransformer.T3 busH=17 busNH=17.4.4.4 R1=0.1 type=GSU
New GICTransformer.T4 busH=17 busNH=17.4.4.4 R1=0.1 type=GSU
New GICTransformer.T5 busH=15 busX=16 busNX=15.4.4.4 R1=0.04 R2=0.06 type=Auto
New GICTransformer.T6 busH=6 busNH=6.4.4.4 R1=0.15 type=GSU
New GICTransformer.T7 busH=6 busNH=6.4.4.4 R1=0.15 type=GSU
New GICTransformer.T8 busH=5 busNH=5.4.4.4 busX=20 busNX=5.4.4.4 R1=0.04
R2=0.06 type=YY
New GICTransformer.T9 busH=5 busNH=5.4.4.4 busX=20 busNX=5.4.4.4 R1=0.04
R2=0.06 type=YY
New GICTransformer.T10 busH=12 busNH=12.4.4.4 R1=0.10 type=GSU
New GICTransformer.T11 busH=12 busNH=12.4.4.4 R1=0.10 type=GSU
New GICTransformer.T12 busH=4 busX=3 busNX=4.4.4.4 R1=0.04 R2=0.06 type=Auto
New GICTransformer.T13 busH=4 busNH=4.4.4.4 busX=3 busNX=4.4.4.4 R1=0.2 R2=0.1
type=YY
New GICTransformer.T14 busH=4 busX=3 busNX=4.4.4.4 R1=0.04 R2=0.06 type=Auto
New GICTransformer.T15 busH=15 busX=16 busNX=15.4.4.4 R1=0.04 R2=0.06 type=Auto
New Capacitor.T1 bus1=2.4 bus2=2.5 phases=1 cuf=10
```

[1]