ELECTRIC POWER RESEARCH INSTITUTE

New User Primer

The Open Distribution System Simulator™ (OpenDSS)

Jason Sexauer
OpenDSS User

License

Table of Contents

Figures

What is OpenDSS?

AT A GLANCE

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 as both a stand-alone executable program and an in-process COM server DLL designed to be driven from a variety of existing software platforms. 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.

Another major strength of OpenDSS is in its "quazi-static" solution modes which lend themselves well to sequential time simulations, like analyzing how a circuit will perform over an entire year. The program has several built-in solution modes, such as

- Snapshot Power Flow
- Daily Power Flow
- Yearly Power Flow
- Harmonics
- Dynamics
- Fault study
- Monte Carlo Fault study
- 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 knowledgeable users to use the features of the program to perform new types of studies.

Through the COM interface, the user is able to 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.

A Brief History

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.

The program’s heritage is closer to a harmonic flow analysis program, or even a dynamics program, than a typical power flow program. One of the most visible products of this representation is the somewhat “back seat” role buses play in the model: while in most power flow programs, buses are the central element on which everything else is built, buses are dynamically created as needed in OpenDSS. 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.

The Purpose of This Document

The goal of this document is to serve as a landing pad for users first starting out with OpenDSS. While it includes much of the same information as the OpenDSS Manual, it is not meant to be nearly as in-depth a documentation to the software. Summary details of the application will be provided with references to where the interested reader can find additional materials as needed.

The OpenDSS Circuit Model

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 basic "building blocks" of the circuit model are "Power Delivery" elements (devices like lines, transformers, and capacitors) and "Power Conversion" elements (devices like generators and loads). Support models – such as control, shape, meter, and parameter abstractions – can be created to further refine Power Delivery and Power Conversion models. Further information on all of these models is available in the OpenDSS Manual. From these elements, the bus and nodes necessary to represent the interconnected system are dynamical created; this is a significant paradigm shift over traditional load flow engines which are more bus centered.

Buses

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.*

It is always assumed that node 0 for a bus is the reference/ground/0 V bus. Additional nodes are traditionally the phases for that bus (for example, node 2 is the B phase). When specifying a bus address, any and applicable nodes should be included (except node 0, which is assumed to always be provided). For example, when specifying the three-phase bus *BUSNAME*, use:

```
BUSNAME.1.2.3
```

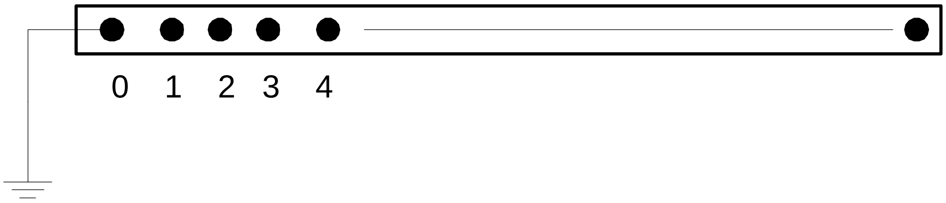


Figure 1: Bus Definition

Power Delivery Elements

Power delivery (PD) elements (also sometimes called power delivery devices) 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).

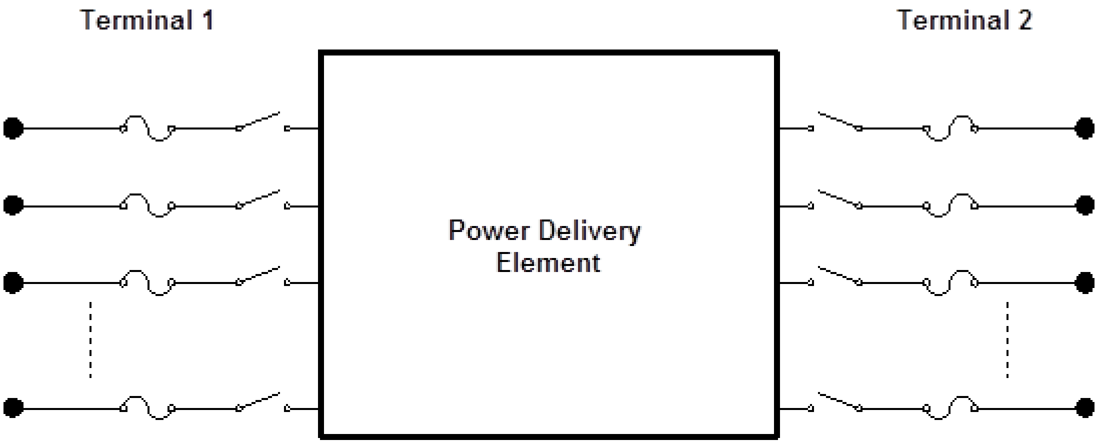


Figure 2: Power Delivery Element Definition

POWER CONVERSION ELEMENTS

Power conversion (PC) elements (also sometimes called power conversion devices) 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 most common power conversion elements are **generators** and **loads**.

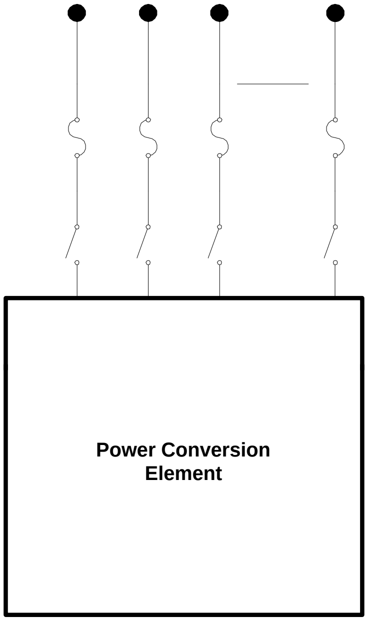


Figure 3: Power Conversion Element Definition

SUPPORT ELEMENTS

A wide array of support elements can be defined to further add to Power Delivery and Power Conversion elements. Support elements provide a convenient abstraction of system parameters, give control functionality, provide monitoring, or provide relevant shape information for time analysis. The most commonly used are:

- **LineCode**– A definition of a line type based on basic data, like the line resistance and reactance.
- **LineGeometry**, **LineSpacing**, and **WireData** – Used together, these define a line type based on fundamental principles and physical parameters like the line’s Geometric Mean Radius (GMR).
- **LoadShape** – A definition of a scaling multiplier (or actual kW/kVAR) values to use in time simulations.
- **Spectrum** – A definition of the harmonic spectrum emitted by a PD element by harmonic order.
- **EnergyMeter** – Used to gather various statistics on a whole feeder.
- **Monitor** – Used to gather power flow results for a specific element.
- **CapControl** and **RegControl** – Emulate the control systems for switched capacitors and regulator tap changers.

Using the OpenDSS GUI

OpenDSS is packaged with a Graphical User Interface (GUI) which provides a structured environment for the creation and analysis of power system cases. The GUI is one of two primary methods users can interact with the OpenDSS solution engine; the other is via the COM interface which is discussed briefly in An Introduction to the COM Interface on page 24 and more extensively in the OpenDSS Manual.

It should be noted that the GUI is more of a tool to aid with analysis of OpenDSS circuits and the creation and debugging of scripts (see The Basics of the OpenDSS Scripting Language on page 18) rather than a replacement for scripting. Communication to the DSS is fundamentally accomplished through text strings passed to the OpenDSS command processor. Scripts or script fragments are executed by:

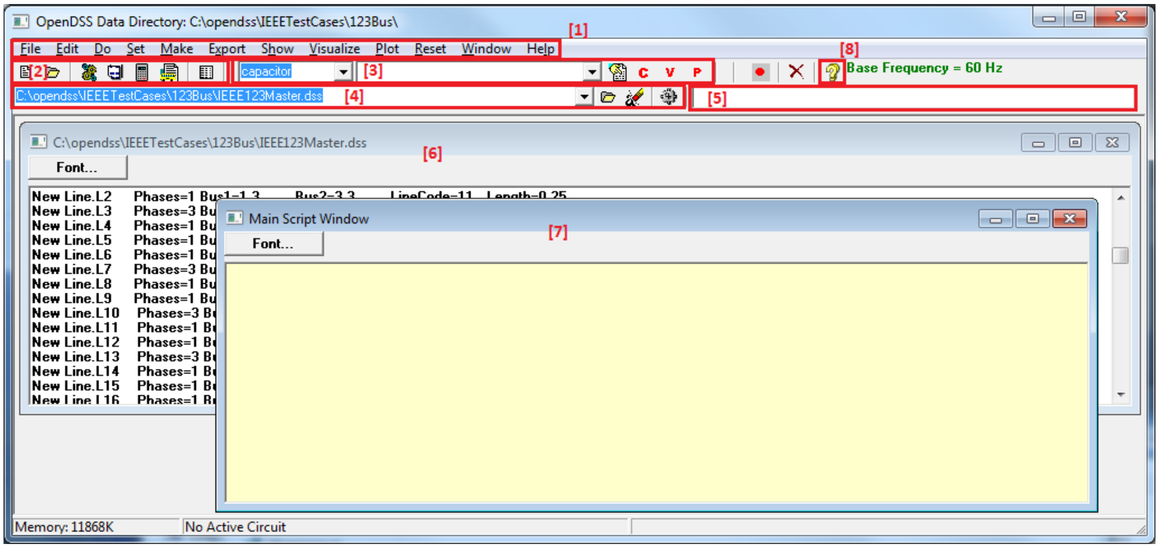
1. Selecting the script lines to be executed.
2. The user can then right-click on the selection and then click on the Do Selected option, which has a short-cut key of Ctrl-D.
3. The selection may also be executed from the Do menu or the speed button directly below the Do menu item.

All commands executed via the GUI have a counterpart in the OpenDSS scripting language. These commands are documented in detail in the OpenDSS Manual and the Command and Element Properties Reference accessible with the GUI through Help > OpenDSS Help. One can also record the actions performed in the GUI to a .dss file using the record commands tool under Edit > Record Script.

Simulation results are returned as arrays of values in text through CSV files. A few standard text file reports are provided by the base OpenDSS software component (see Show and Export menu commands). The intent for users demanding more sophisticated reports is for users to design them through Excel worksheets or whatever application they use to control the OpenDSS in special ways using the COM interface (see An Introduction to the COM Interface on page 24).

The User Interface

When opening OpenDSS, the user is greeted with the following window:


OpenDSS Data Directory: C:\opendss\IEEETestCases\123Bus\
[1]
File Edit Do Set Make Export Show Visualize Plot Reset Window Help
[8]
[2]
capacitor
[3]
C V P
Base Frequency = 60 Hz
C:\opendss\IEEETestCases\123Bus\IEEE123Master.dss
[4]
[5]
C:\opendss\IEEETestCases\123Bus\IEEE123Master.dss
[6]
Font...
Main Script Window
[7]
Font...
Memory: 11868K
No Active Circuit

```
New Line.L2     Phases=1 Bus1=1.3      Bus2=3.3     LineCode=11   Length=0.25
New Line.L3     Phases=3 Bu
New Line.L4     Phases=1 Bu
New Line.L5     Phases=1 Bu
New Line.L6     Phases=1 Bu
New Line.L7     Phases=3 Bu
New Line.L8     Phases=1 Bu
New Line.L9     Phases=1 Bu
New Line.L10    Phases=3 Bı
New Line.L11    Phases=1 Bı
New Line.L12    Phases=1 Bı
New Line.L13    Phases=3 Bı
New Line.L14    Phases=1 Bı
New Line.L15    Phases=1 Bı
New Line.L16    Phases=1 Bı
```

Figure 4. The OpenDSS User Interface

Key components of the interface include,

1. The **menu structure**, which drives most of the workflow in OpenDSS. Menus of interest include:
   - The **Set** menu, which allows one to set any solution parameter that can be set via the `options` scripting command.
   - The **Export** menu, which allows one to save various reports to csv files.
   - The **Show** menu, which contains much of the same information as the reports as the Export menu, but displays them directly in the GUI.
   - The **Visualize** menu, which provides a graphical output of the device selected via the element selector (item 3 in the figure).
   - The **Plot** menu, which provides graphical output relevant to the whole system.
2. The **toolbar**, which provides direct access to many commonly used OpenDSS commands such as “Solve,” “Summary,” and “Do Command.”
3. The **element tools**, which allows the user to select what circuit element (by type) to edit or display visualizations for.
4. The **script tools**, which allows one to select which of the current opened scripts to run.
5. The **results bar**, which provides a condensed version of the Results window which can be accessed through Show > Result Form.
6. **Script Windows** to directly edit various *.dss files
7. The **Main Script Window** is a sort of “notepad” or “interactive window” for OpenDSS. The user can type small commands and run them via the “Do Command” feature (Ctrl-D). The contents of this window are retained between sessions.
8. The **Help** button which brings up the OpenDSS Command and Element Properties Reference which gives a tree-view guide to the various script commands in OpenDSS.

WORKFLOW

In general, the user will want to:

1. **Define the circuit** they wish to study by creating new lines, transformers, loads, generators, etc…
    a. Realistically, the best way to do this is by creating a dss script. More details on scripting are provided in the next chapter starting on page 18
    b. Once a script has been written, use the ⚙ button to run the selected script.
2. **Set up the circuit options**, such as the solution mode (snapshot, daily, harmonic, etc…)
    a. This is accomplished through the commands in the Set menu. The most basic form of analysis is the "snapshot" which is analogous to a traditional power flow. For additional information on circuit options, see the OpenDSS Manual.
3. **Solve** the powerflow problem
    a. First, ensure that the bus list is created and the base voltages found by running Do > Calc Voltage Bases.
    b. Then use the 🖩 button in the toolbar to solve the circuit.
4. **Perform analysis** on the solved circuit – The specifics of how to accomplish this vary from analysis to analysis, but some general tasks include:
    a. *Looking at a branch, transformer, load, or other element in the system* – To do this, first select the element type and then the element from the element toolbar. Then select the C, V, or P buttons to see a current, voltage, or power visualization. Select the 📝 button to edit the element. An example of a voltage visualization for a line is provided in Figure 5.
    b. *Visualize the voltage profile of the system* – Type `plot profile` into the main script window, and press Ctrl-D to Do that command. A voltage profile will come up showing how the voltage progresses as one progresses down the feeder; an example is provided in Figure 6. Additional parameter can be specified via the `plot` command; see the OpenDSS Manual for more details.
    c. *Visualize data onto a one-line of the feeder* – If you provide bus location data via the `buscoords` command (see the OpenDSS Manual for more information), you can superimpose power flow data onto a map/one-line of the system. To do this, go Plot > Circuit Plots > Circuit Plot. An example is provided in Figure 7. Additional parameter cans be specified via the `plot` command; see the OpenDSS Manual for more details.
    d. *Export data for analysis in 3rd party program* – All results obtained through OpenDSS can be exported through the various commands in the "Export" menu. Results are .csv files.

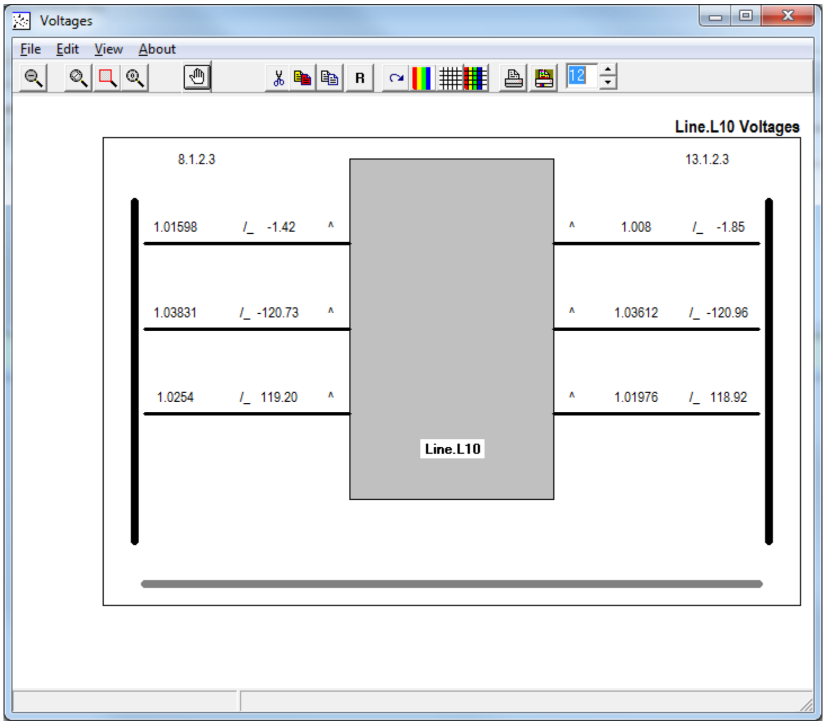


Figure 5. Voltage Visualization for a Line Element

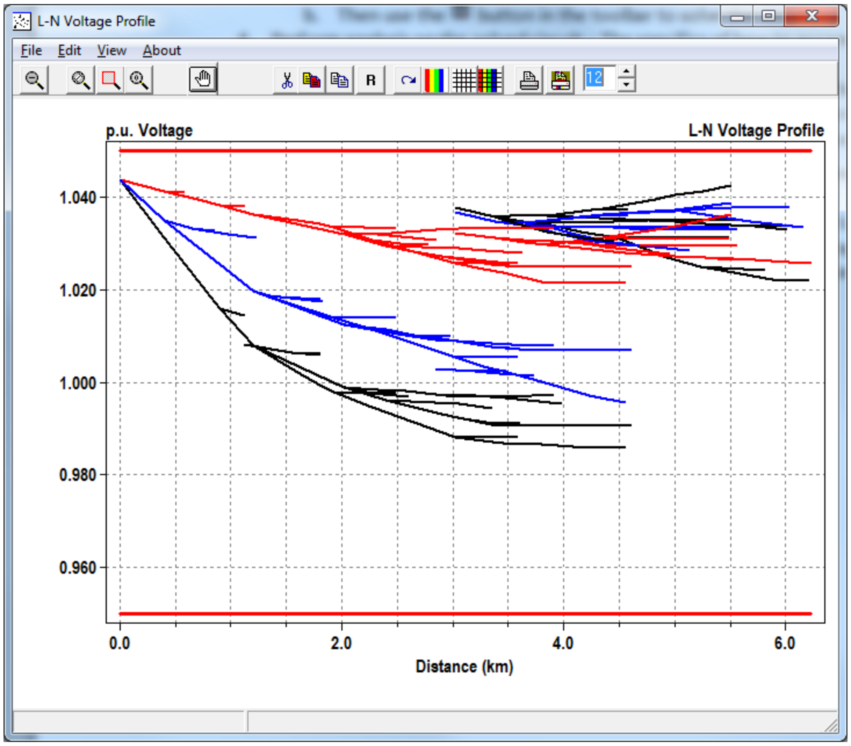


Figure 6. Voltage Profile for a Feeder

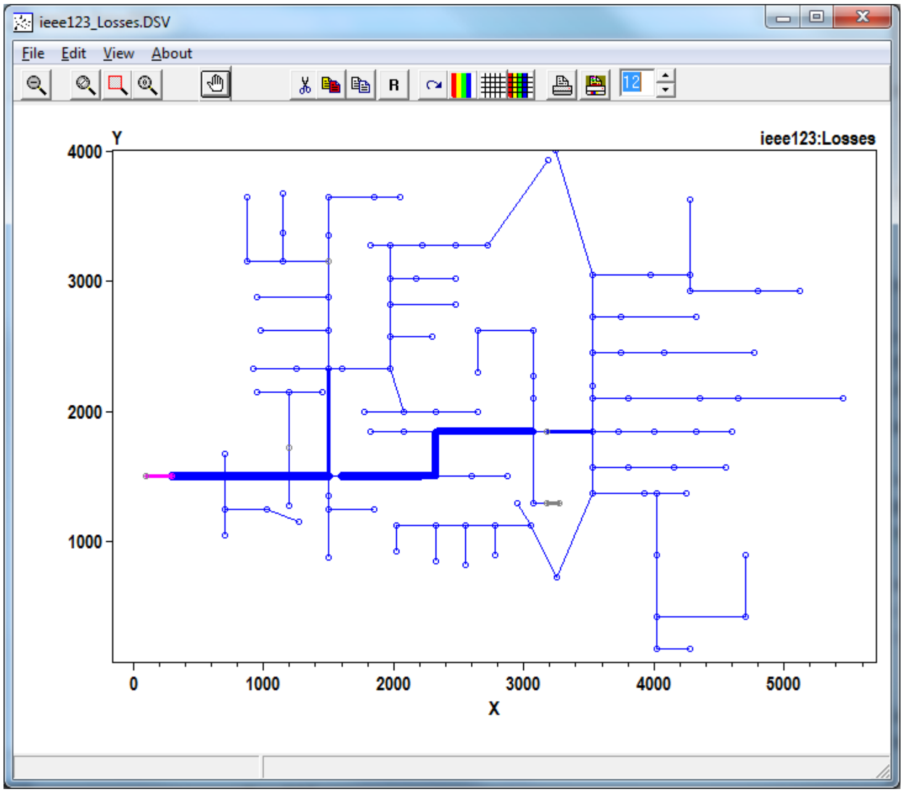


Figure 7. Lines Losses for a Feeder

The Basics of the OpenDSS Scripting Language

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.

The OpenDSS scripting language is case insensitive.

Always refer to the **Help** command or the command documentation (accessible from the ? button) for the latest commands and property names that are recognized by the DSS.

Command Syntax

A command is a single line of a script. A command verb which interacts directly with a circuit element (like the `New` and `Edit` commands) initiates a command that takes the following form:

```
CommandVerb ElementClass.Element.Param1=Val1, Param2=Val2
```

Command verbs which do not interact with circuit elements directly (like the `Plot` and `Export` commands) initiate a command that take the following form:

```
CommandVerb Param1=Val1, Param2='Value 2', Param3=(1 2 3)
```

Command Verbs

A full list of command verbs is available in the “Executive” section of the command documentation, opened via the ? button. Some common command verbs include:

- `New` – Create a new circuit element
- `Edit` – Edit a specified circuit element
- `Set` – Define solution options, like the mode. See the “options” section of the command documentation for more information.
- `Solve` – Perform a solution of the current circuit
- `Show` – Display power flow results in text format. See the “show” section of the command documentation for more information.
- `Export` – Export power flow results to a text/csv file. See the “export” section of the command documentation for more information.
- `Plot` – Plot power flow results. See the “plot” section of the command documentation for more information.

If no command is specified, the `Edit` command is assumed.

PARAMETERS

Parameters/value pairs may be separated by commas (,) or white space (blank, tab). If a value requires delimiters, the following are acceptable. While technically interchangeable, it is encouraged to follow the following stylistic standards:

double quotes	`"..."`	Strings
single quotes	`'...'`	Strings
parentheses	`(...)`	Arrays
brackets	`[...]`	Arrays
curly braces	`{...}`	Inline Math (see OpenDSS Manual for details)

Access to a class's members or element's parameters (also known as properties) is provided through the period (.) symbol. All elements should be specified through their fully qualified name, unless the context can be specifically inferred[1]. The fully qualified name is in the following format:

```
ElementClass.ElementName
```

As an extension, an element's property's fully qualified name is in the following format:

```
ElementClass.ElementName.PropertyName
```

When accessing multiple parameters of the same element within the same command, additional property names beyond the first one do *not* need to be fully qualified. For example[2]:

```
Line.L1.Bus1=1, Bus2=5
```

In addition to being specified in parameter/values pairs, parameters can also be specified in their default order. This is most commonly done when specifying the `element` or `object` property, as it is always the first property. Specification by order and by parameter/value pairs can be mixed in the same line, as shown below:

```
New EnergyMeter.Feeder Line.L115, terminal=1, enabled=false
```

When specifying a new device, many of the values are given reasonable default values. Full documentation of what values must be specified, what values have defaults, and what the default value is set to is provided in the command documentation. In general, element classes which have `element`, `object`, `terminal`, `bus1`, or `bus2` properties do not have default values and must be specified at device creation.

[1] For example, when specifying a capacitor control's `element` property, the fact the `element` is a capacitor is inferred, thus the ElementClass does not have to be included

[2] Also note no command verb is specified, thus the `Edit` command verb is being inferred.

Comments

Comments can be added to scripts using // or ! as demonstrated below:

```
// Edit the voltage regulator control
RegControl.Ctrl1. maxtapchange=1  ! Limit to one tap change
```

Multi-line comments are notated via /* … */. Note that /* must be on the first column of the line. For example:

```
/*   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
*/
```

Multi-Line Commands

If a command needs to span multiple lines, the ~ symbol (which is an alias for the More command) should be used, such as in the example below:

```
New Line.L1   Bus1=1, Bus2=2, Length=1
~   units=mi, geometry=3PH_3/0_Horiz
```

Note that the above code is actually technically two separate commands. As such, any non-default parameters that must be defined at instantiation of an Element Class must be defined in the first line. Thus, the following would generate an error:

```
// This will error because Bus1 and Bus2 are not set in the first line
New Line.L1   Length=1, units=mi
~   Bus1=1, Bus2=2, geometry=3PH_3/0_Horiz
```

Including External Files

There are two ways to include external scripts in an OpenDSS script

1. The Compile command, which inserts another OpenDSS script at that location in the script file. The Compile command changes the default directory to the directory the external file is located in.
2. The Redirect command, which inserts another OpenDSS script at that location in the script file. The redirect command keeps the default directory as the directory of the source script calling the redirect command.

Some properties, like the mult property found in LoadShapes, may require large lists of data which are inconvenient to include directly in the script or which come from an external source. The File capability, which is used in lieu of a traditional value, allows one to read in a file and use its contents as the value stream for an element. For example:

```
LoadShape.LS1 mult=(File='Example.csv')
```

Refer to the OpenDSS Manual for additional information on the File capability and its sister functions sngFile and dblFile.

WORKFLOW

As mentioned previously, any tasks that can be done in the GUI can be done through the scripting language. As such, the tasks presented below are similar to "Workflow" on page 15. In general, the user will want to:

1. **Define the circuit** they wish to study by creating new lines, transformers, loads, generators, etc…
   a. Use the `Clear` command to clear out any circuit that may already be loaded into OpenDSS
      ex: `Clear`
   b. Use the `New` command to define a new circuit
      ex: `New object=circuit.ExampleCircuit`
   c. For each element, use the `New` command to define the elements in the circuit
      ex: `New line.L1 bus1=1 bus2=2 linecode=336ACSR length=1.25`
2. **Set up the circuit options**, such as the solution mode (snapshot, daily, harmonic, etc…)
   a. Use the `Set` command to set the values for circuit options
      ex: `Set mode=snapshot voltagebases=(115, 12.47, .14)`
3. **Solve** the powerflow problem
   a. First, ensure that the bus list is created and the base voltages found using the `CalcVoltageBases` command
      ex: `CalcVoltageBases`
   b. Use the solve command to perform the simulation
      ex: `Solve`
4. **Perform analysis** on the solved circuit – The specifics of how to accomplish this vary from analysis to analysis, but some general tasks include:
   a. *Looking at a branch, transformer, load, or other element in the system* – Use the `Visualize` command
      ex: `Visualize element=Line.L1 what=powers`
   b. *Visualize the voltage profile of the system* – Use the `plot` command
      ex: `Plot profile`
   c. *Visualize data onto a one-line of the feeder* – Use the `Buscords` and `Plot` commands
      ex:
      ```
      Buscoords BusCordsFile.dat
      Plot circuit
      ```
   d. *Export data for analysis in 3rd party program* – Use the `Export` command
      ex: `Export voltages`

EXAMPLE SCRIPT

Consider the following circuit:

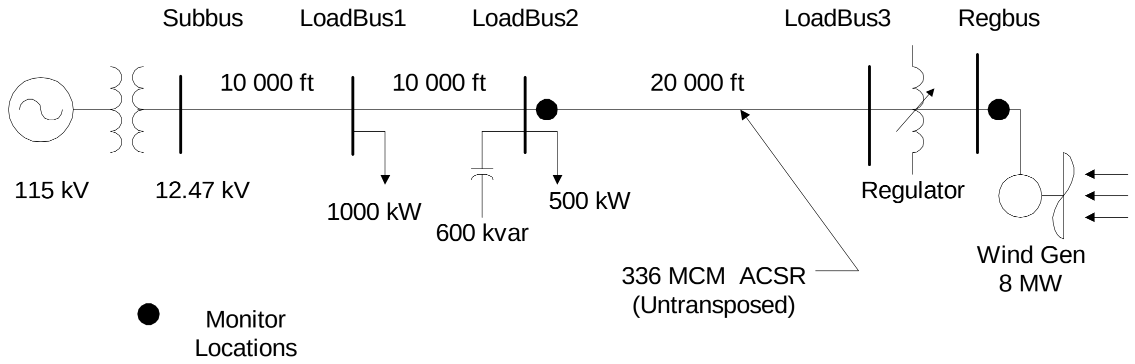


Figure 8. Example Circuit

To create this circuit and perform some basic analysis, the following script could be used:

```
// The first step is always to clear the DSS and instantiate a new circuit
clear
New object=circuit.ExampleCircuit
~ basekv=115  1.00 0.0 60.0 3 20000 21000 4.0 3.0  ! edit the voltage source

// Define some load shapes for the loads and wind
! This is an example of defining parameters via their default order
!  in this case, the num of points and interval
New loadshape.day 8 3.0
~   mult=(.3 .36 .48 .62 .87 .95 .94 .60)
! This is an example of an inline calculation, see OpenDSS Manual for more info
New loadshape.wind 2400 {1 24 /}                ! unit must be hours
~   mult=(file=zavwind.csv) action=normalize    ! wind turbine characteristic

// Define a linecode for the lines - unbalanced 336 MCM ACSR connection
New linecode.336matrix nphases=3   ! horizontal flat construction
! rmatrix, xmatrix, and xmatrix are in lower triangular matrix format.
!   see the OpenDSS Manul for more details on how to specify matrixes.
! In ohms per 1000 ft
~   rmatrix=(0.0868455 |  0.0298305 0.0887966 | 0.0288883 0.0298305  0.0868455)
! In ohms per 1000 ft
~   xmatrix=(0.2025449 |  0.0847210 0.1961452 | 0.0719161 0.0847210  0.2025449)
! In nf per 1000 ft
~   cmatrix=(2.74 | -0.70 2.96| -0.34 -0.71 2.74)
~   Normamps = 400  Emergamps=600

// Define Substation transformer
! Note that the buses property provides an alternate way to specify the
!   buses beyond bus1= and bus2=
New transformer.subxfrm 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 bus1=subbus   bus2=loadbus1 linecode=336matrix length=10
New line.line2 loadbus1 loadbus2 336matrix 10
New line.line3 Loadbus2 loadbus3 336matrix 20
```

```
// Define the loads
New load.load1 bus1=loadbus1 phases=3 kv=12.47 kw=1000.0 pf=0.88 model=1
~   class=1 duty=day
New load.load2 bus1=loadbus2 phases=3 kv=12.47 kw=500.0 pf=0.88 model=1
~   class=1 duty=day conn=delta

// Capacitor with control
New capacitor.Cap1  bus1=loadbus2  phases=3 kvar=600 kv=12.47
New capcontrol.Cap1Ctrl element=line.line3 terminal=1 capacitor=Cap1
~   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.subxfrmCtrl  transformer=subxfrm  winding=2 vreg=125
~   band=3 ptratio=60 delay=10
New regcontrol.reg1Ctrl 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 we can see what's happening
//  (See documentation on how the mode parameter works)
! Monitor the power output of the wind turbine
New Monitor.gen1 element=generator.gen1 terminal=1  mode=1
! Monitor the voltage and currents at the second load bus
New Monitor.loadbus2 load.load2 1 mode=0
! Monitor sequence voltages and currents magnitudes of line 3, terminal 1
New Monitor.line3 line.line3 1 mode=48
! You need an energy meter in order to get line distances for a profile plot
New Energymeter.em1 line.line1

// Define voltage bases so voltage reports come out in per unit
Set voltagebases=(115 12.47 .48)

// Generate the bus list and figure out the voltage bases
Calcvoltagebases

// Simulation options to make the cap and reg controllers operate in sync
// with the rest of the simulation
Set controlmode=time
// Simulation options to do a time based simulation for 24 hours (86400 sec)
//  with a time step of 1 sec starting at hour 0, second 0
Set mode=duty number=86400  hour=0  stepsize=1 sec=0

// Conduct the simulation
Solve

// Show some results
! Plot how the voltage at loadbus1 looked during the day
Plot monitor, object=loadbus2, Channels=(1,3,5)
! Visualize the line's flow as it appear at the last timestep
Visualize element=Line.line1 what=powers
! Show the voltage profile on the feeder as it appeared at the last timestep
Plot profile
```

An Introduction to the COM Interface

While a lot can be done with the standard text scripting interface, knowledgeable users can open up an entirely new world of applications by learning to effectively use the COM interface. The COM interface allows one to develop algorithms in another computer program and then drive the OpenDSS engine 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 are 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.

Detailed documentation of the COM interface is provided in the OpenDSS manual, on the OpenDSS Wiki, and in the TechNotes (see Reference Resources on page 35). However, the most up to date documentation is within the COM interface itself; use a type library browser (TLB) – such as the provided in Microsoft Office's Visual Basic Editor – to expose the library by opening the VBA editor when in Excel, for example, add a reference (under the Tools menu) to OpenDSSEngine. Then you will be able to browse the "library" for OpenDSSEngine using the "Object Browser" (in the View menu) in the VBA editor.

In this primer, Visual Basic code will be used for examples because it is (arguably) the easiest language for anyone to read, regardless of their programming background. It is also widely available: you can try these examples by using the macro editor of Microsoft Excel or in the free Visual Studio Express version of Visual Basic. Code snippets in Mathwork's MATLAB (a language commonly used for scientific computing) and Python (a popular free scripting language which is gaining a good amount of popularity in the scientific computing community) are provided at the end of the chapter. However, users have successful used OpenDSS's COM library in a wide array of languages including C++, C#, and R.

Starting Out with The COM Interface

First of all, make sure that the programming environment you are using is connected to the OpenDSS COM interface. In Visual Basic, this is done by going to Tools > References or Project > Add Reference depending on what version you are using. From here, select the OpenDSSEngine COM type library.

To instantiate an OpenDSS object and create a link to commonly used functions, use the following code:

```
' Declare the OpenDSS related variables
Dim DSSObj As OpenDSSengine.DSS
Dim DSSText As OpenDSSengine.Text
Dim DSSCircuit As OpenDSSengine.Circuit
Dim DSSSolution As OpenDSSengine.Solution

' Instantiate the OpenDSS Object
DSSObj = New OpenDSSengine.DSS

' Start up the Solver
If Not DSSObj.Start(0) Then
    MsgBox("Unable to start the OpenDSS Engine")
    Return
End If

' Set up the Text, Circuit, and Solution Interfaces
DSSText = DSSObj.Text
DSSCircuit = DSSObj.ActiveCircuit
DSSSolution = DSSCircuit.Solution
```

As can be seen, from the DSS parent object several useful children classes exist. These are

- The **Text** interface, which provides access to the command line interpreter interface. Using this object, one can directly execute OpenDSS scripting commands, as found in the previous section “The Basics of the OpenDSS Scripting Language.”
- The **Circuit** interface, which provides access to the elements that make up the circuit. Using members of this object, one can iterate through, edit, and monitor various circuit elements.
- The **Solution** interface, which provides access to the solution. Using members of this object, one can define solution parameters, solve the circuit, and view properties of the solution, like the number of iterations it took.

The next several sections will take a more in-depth look at these interfaces.

THE TEXT INTERFACE

The text interface is the simplest, but one of the most useful, interfaces. It allows one to execute OpenDSS scripting commands from the COM interface. Thus anything a user knows how to do using the scripting engine is instantly available to the COM user!

To use the `Text` interface, set the `Command` property of the `Text` interface to a string with the command to execute:

```
' Load in an example circuit
DSSText.Command = "Compile 'C:\example\IEEE123Master.dss'"
' Create a new capacitor
DSSText.Command = "New Capacitor.C1 Bus1=1 Phases=3 kVAR=1200"
DSSText.Command = "~ Enabled=false" ' You can even use ~
' Change the bus for Line L1
DSSText.Command = "Line.L1.Bus1 = 5"
```

If the command one is executing displays a result, the result can be retrieved using the `Result` property of the `Text` interface:

```
' Export voltage to a csv file
Dim Filename As String
DSSText.Command = "Export Voltages"
Filename = DSSText.Result
MsgBox("File saved to: " & Filename)
```

Introduction to the Circuit Interface

The circuit interface is commonly used to edit the properties of the various elements in a circuit. Nearly all the element classes in OpenDSS (ie, Lines, Loads, Capacitors, CapControls, etc…) have a child object under the circuit interface. These child objects also have convenient functions to allow one to iterate through the member elements which are useful when looping. For example, this script will loop through all the loads in a circuit and scale the kW up by 20%:

```
' Step through every load and scale it up
Dim iLoads As Integer ' Track what load we're on

iLoads = DSSCircuit.Loads.First
While iLoads
    ' Scale load by 120%
    DSSCircuit.Loads.kW = DSSCircuit.Loads.kW * 1.2
    ' Move to next load
    iLoads = DSSCircuit.Loads.Next
End While
```

If one wants to edit a specific element, use the `SetActiveElement` method and the `ActiveDSSElement` interface as shown below:

```
' Set a capacitor's rated kVAR to 1200
DSSCircuit.SetActiveElement("Capacitor.C83")
DSSCircuit.ActiveDSSElement.Properties("kVAR").Val = 1200
```

However, the benefit of this method over simply using the text interface, as shown below, is debatable. Use whichever method makes sense to you.

```
' Does the same thing as the previous snippet
DSSText.Command = "Capacitor.C83.kVAR = 1200"
```

Another useful feature of the Circuit Interface is to retrieve power flow results, such as bus voltages, element losses, etc… This example gets the bus names and voltages:

```
' Get bus voltages
Dim BusNames As String()
Dim Voltages As Double()
BusNames = DSSCircuit.AllBusNames
Voltages = DSSCircuit.AllBusVmagPu

' See what an arbitrary bus's voltage is
MsgBox(BusNames(5) & "'s voltage mag in per unit is: " & Voltages(5))
```

For more on getting power flow results from the circuit interface, see the “All commands” (`AllElementLoses`, `AllNodeVmagByPhase`, etc…) in the COM section of the OpenDSS manual.

INTRODUCTION TO THE SOLUTION INTERFACE

The Solution Interface is used to monitor and control the OpenDSS solution process and control procedures. This includes solving the circuit, setting the solution mode, monitoring convergence, reporting the time interval for time and duty based solution, and other such aspects of solving an OpenDSS circuit.

At its most fundamental level, the solution interface allows one to solve the circuit:

```
' Solve the Circuit
DSSSolution.Solve()
If DSSSolution.Converged Then
       MsgBox("The Circuit Solved Successfully")
End If
```

One can also control the solution at a more granular level. For example, say one wanted to model the effects of a large load pickup 30 seconds into a simulation. The code below could be used:

```
' Model effects of a large load pickup 30 seconds into a simulation
DSSText.Command = _
       "New Monitor.Mon1 element=Line.L100 mode=0"
DSSSolution.StepSize = 1        ' Set step size to 1 sec
DSSSolution.Number = 30         ' Solve 30 seconds of the simulation
' Set the solution mode to duty cycle, which forces loads to use their
'   "duty cycle" loadshape and allows time based simulation
DSSSolution.Mode = OpenDSSengine.SolveModes.dssDutyCycle
DSSSolution.Solve()

DSSCircuit.Enable("Load.L1")    ' Enable the load
DSSSolution.Number = 30         ' Solve another 30 seconds of simulation
DSSSolution.Solve()

MsgBox("Seconds Elapsed: " & DSSSolution.Seconds)
' Plot the voltage for the 60 seconds of simulation
DSSText.Command = "Plot monitor object=Mon1 Channels=(1,3,5)"
```

Finally, the circuit interface grants very specific information into the method and control behind the solution. Using the various “Solve” methods (ex, `Solve`, `SolveDirect`, `SolveNoControl`, etc…) and other functions like `SystemYChanged` and `MostIterationsDone`, granular control of and information on the solution process can be obtained. Using `CheckControls` and other methods in the Solution Interface, along with the `DSSCircuit.CtrlQueue` interface, specialized control schemes can be implemented. The details of this are beyond the scope of this document; more information can be found in the OpenDSS TechNotes (see “Reference Resources” on page 35).

Example in Visual Basic

The code presented below is identical to the snippets presented throughout this chapter. It is provided in one convenient location for reference.

```
' ****************************************************
' * Initialize OpenDSS
' ****************************************************
' Declare the OpenDSS related variables
Dim DSSObj As OpenDSSengine.DSS
Dim DSSText As OpenDSSengine.Text
Dim DSSCircuit As OpenDSSengine.Circuit
Dim DSSSolution As OpenDSSengine.Solution

' Instantiate the OpenDSS Object
DSSObj = New OpenDSSengine.DSS

' Start up the Solver
If Not DSSObj.Start(0) Then
    MsgBox("Unable to start the OpenDSS Engine")
    Return
End If

' Set up the Text, Circuit, and Solution Interfaces
DSSText = DSSObj.Text
DSSCircuit = DSSObj.ActiveCircuit
DSSSolution = DSSCircuit.Solution

' ****************************************************
' * Examples Using the DSSText Object
' ****************************************************
' Load in an example circuit
DSSText.Command = "Compile 'C:\example\IEEE123Master.dss'"
' Create a new capacitor
DSSText.Command = "New Capacitor.C1 Bus1=1 Phases=3 kVAR=1200"
DSSText.Command = "~ Enabled=false" ' You can even use ~
' Change the bus for Line L1
DSSText.Command = "Line.L1.Bus1 = 5"

' Export voltage to a csv file
Dim Filename As String
DSSText.Command = "Export Voltages"
Filename = DSSText.Result
MsgBox("File saved to: " & Filename)

' ****************************************************
' * Examples Using the DSSCircuit Object
' ****************************************************
' Step through every load and scale it up
Dim iLoads As Integer ' Track what load we're on

iLoads = DSSCircuit.Loads.First
While iLoads
    ' Scale load by 120%
    DSSCircuit.Loads.kW = DSSCircuit.Loads.kW * 1.2
    ' Move to next load
    iLoads = DSSCircuit.Loads.Next
End While

' Set a capacitor's rated kVAR to 1200
DSSCircuit.SetActiveElement("Capacitor.C83")
DSSCircuit.ActiveDSSElement.Properties("kVAR").Val = 1200
```

```
' Get bus voltages
Dim BusNames As String()
Dim Voltages As Double()
BusNames = DSSCircuit.AllBusNames
Voltages = DSSCircuit.AllBusVmagPu

' See what an arbitrary bus's voltage is
MsgBox(BusNames(5) & "'s voltage mag in per unit is: " & Voltages(5))

' ****************************************************
' * Examples Using the DSSSolution Object
' ****************************************************
' Solve the Circuit
DSSSolution.Solve()
If DSSSolution.Converged Then
    MsgBox("The Circuit Solved Successfully")
End If

' Model effects of a large load pickup 30 seconds into a simulation
DSSText.Command = _
    "New Monitor.Mon1 element=Line.L100 mode=0"
DSSSolution.StepSize = 1        ' Set step size to 1 sec
DSSSolution.Number = 30         ' Solve 30 seconds of the simulation
' Set the solution mode to duty cycle, which forces loads to use their
'   "duty cycle" loadshape and allows time based simulation
DSSSolution.Mode = OpenDSSengine.SolveModes.dssDutyCycle
DSSSolution.Solve()

DSSCircuit.Enable("Load.L1")    ' Enable the load
DSSSolution.Number = 30         ' Solve another 30 seconds of simulation
DSSSolution.Solve()

MsgBox("Seconds Elapsed: " & DSSSolution.Seconds)
' Plot the voltage for the 60 seconds of simulation
DSSText.Command = "Plot monitor object=Mon1 Channels=(1,3,5)"
```

Example in MATLAB

The code presented below performs the same functions as the code snippets presented throughout this chapter in Mathwork's MATLAB. Much of this code could also be implemented in Octave with little change (see the Octave example directory in the install path).

```
clc
clear all
close all

% ****************************************************
% * Initialize OpenDSS
% ****************************************************
% Instantiate the OpenDSS Object
DSSObj = actxserver('OpenDSSEngine.DSS');

% Start up the Solver
if ~DSSObj.Start(0),
    disp('Unable to start the OpenDSS Engine')
    return
end

% Set up the Text, Circuit, and Solution Interfaces
DSSText = DSSObj.Text;
DSSCircuit = DSSObj.ActiveCircuit;
DSSSolution = DSSCircuit.Solution;

% ****************************************************
% * Examples Using the DSSText Object
% ****************************************************
% Load in an example circuit
DSSText.Command = 'Compile "C:\example\IEEE123Master.dss"';
% Create a new capacitor
DSSText.Command = 'New Capacitor.C1 Bus1=1 Phases=3 kVAR=1200';
DSSText.Command = '~ Enabled=false'; % You can even use ~
% Change the bus for Line L1
DSSText.Command = 'Line.L1.Bus1 = 5';

% Export voltage to a csv file
DSSText.Command = 'Export Voltages';
Filename = DSSText.Result;
disp(['File saved to: ' Filename])

% ****************************************************
% * Examples Using the DSSCircuit Object
% ****************************************************
% Step through every load and scale it up

iLoads = DSSCircuit.Loads.First;
while iLoads,
    % Scale load by 120%
    DSSCircuit.Loads.kW = DSSCircuit.Loads.kW * 1.2;
    % Move to next load
    iLoads = DSSCircuit.Loads.Next;
end
```

```
% Set a capacitor's rated kVAR to 1200
DSSCircuit.SetActiveElement('Capacitor.C83');
DSSCircuit.ActiveDSSElement.Properties('kVAR').val = '1200';

% Get bus voltages
BusNames = DSSCircuit.AllBusNames;
Voltages = DSSCircuit.AllBusVmagPu;

% See what an arbitrary bus's voltage is
disp([BusNames{5} '''s voltage mag in per unit is: '
num2str(Voltages(5))])

% ****************************************************
% * Examples Using the DSSSolution Object
% ****************************************************
% Solve the Circuit
DSSSolution.Solve();
if DSSSolution.Converged,
    disp('The Circuit Solved Successfully')
end

% Model effects of a large load pickup 30 seconds into a simulation
DSSText.Command = ...
    'New Monitor.Mon1 element=Line.L100 mode=0';
DSSSolution.StepSize = 1;        % Set step size to 1 sec
DSSSolution.Number = 30;         % Solve 30 seconds of the simulation
% Set the solution mode to duty cycle, which forces loads to use their
%   'duty cycle' loadshape and allows time based simulation
DSSSolution.Mode = 6;             % Code for duty cycle mode
DSSSolution.Solve();

DSSCircuit.Enable('Load.L1');    % Enable the load
DSSSolution.Number = 30;         % Solve another 30 seconds of simulation
DSSSolution.Solve();

disp(['Seconds Elapsed: ' num2str(DSSSolution.Seconds)])
% Plot the voltage for the 60 seconds of simulation
DSSText.Command = 'Plot monitor object=Mon1 Channels=(1,3,5)';
```

EXAMPLE IN PYTHON

The code presented below performs the same functions as the code snippets presented throughout this chapter in Python.

```python
import win32com.client

# ****************************************************
# * Initialize OpenDSS
# ****************************************************
# Instantiate the OpenDSS Object
try:
    DSSObj = win32com.client.Dispatch("OpenDSSEngine.DSS")
except:
    print "Unable to start the OpenDSS Engine"
    raise SystemExit

# Set up the Text, Circuit, and Solution Interfaces
DSSText = DSSObj.Text
DSSCircuit = DSSObj.ActiveCircuit
DSSSolution = DSSCircuit.Solution


# ****************************************************
# * Examples Using the DSSText Object
# ****************************************************
# Load in an example circuit
DSSText.Command = r"Compile 'C:\example\IEEE123Master.dss'"
# Create a new capacitor
DSSText.Command = "New Capacitor.C1 Bus1=1 Phases=3 kVAR=1200"
DSSText.Command = "~ Enabled=false" # You can even use ~
# Change the bus for Line L1
DSSText.Command = "Line.L1.Bus1 = 5"

# Export voltage to a csv file
DSSText.Command = "Export Voltages"
Filename = DSSText.Result
print "File saved to: " + Filename


# ****************************************************
# * Examples Using the DSSCircuit Object
# ****************************************************
# Step through every load and scale it up
iLoads = DSSCircuit.Loads.First
while iLoads:
    # Scale load by 120%
    DSSCircuit.Loads.kW = DSSCircuit.Loads.kW * 1.2
    # Move to next load
    iLoads = DSSCircuit.Loads.Next

# Set a capacitor's rated kVAR to 1200
DSSCircuit.SetActiveElement("Capacitor.C83")
DSSCircuit.ActiveDSSElement.Properties("kVAR").Val = 1200

# Get bus voltages
BusNames = DSSCircuit.AllBusNames
Voltages = DSSCircuit.AllBusVmagPu
```

```
# See what an arbitrary bus's voltage is
print BusNames[5] + "'s voltage mag in per unit is: " + \
      str(Voltages[5])

# ****************************************************
# * Examples Using the DSSSolution Object
# ****************************************************
# Solve the Circuit
DSSSolution.Solve()
if DSSSolution.Converged:
    print "The Circuit Solved Successfully"

# Model effects of a large load pickup 30 seconds into a simulation
DSSText.Command = \
    "New Monitor.Mon1 element=Line.L100 mode=0"
DSSSolution.StepSize = 1        # Set step size to 1 sec
DSSSolution.Number = 30         # Solve 30 seconds of the simulation
# Set the solution mode to duty cycle, which forces loads to use their
#   "duty cycle" loadshape and allows time based simulation
DSSSolution.Mode = 6            # Code for duty cycle mode
DSSSolution.Solve()

DSSCircuit.Enable("Load.L1")    # Enable the load
DSSSolution.Number = 30         # Solve another 30 seconds of simulation
DSSSolution.Solve()

print "Seconds Elapsed: " + str(DSSSolution.Seconds)
# Plot the voltage for the 60 seconds of simulation
DSSText.Command = "Plot monitor object=Mon1 Channels=(1,3,5)"
```

Additional Resources

As you continue to use OpenDSS and explore its vast capabilities, you will no doubt run into situations where you require more information than is provided in this introductory guide. For these purposes, the OpenDSS developers and community have provided several resources.

WHERE TO GO FROM HERE?

From the information in this manual, you should be ready to get your feet wet and start playing around with the software. An extensive series of examples of all the features of OpenDSS are provided in the Examples directory of the OpenDSS installation path. These include:

- Further OpenDSS scripting examples, provided in the “Examples/Scripts” directory.
- Examples of the various solution modes, such as time based simulations, duty-cycle simulations, annual energy simulations, Monte Carlo analysis, dynamic analysis, harmonic analysis, fault studies, capacitor placement optimization, geomagnetically induced current studies, and many more
- Examples of the various circuit elements, such as
    - Voltage regulators and their controls, capacitors and their controls, and centralized volt-VAR optimization controllers
    - All sorts of generator models, like the classic PV generator model, fixed power factor generators, inverters, photovoltaic systems, induction generators, storage elements, centrally dispatched generation, and others.
    - Fuses, reclosers, relays, and faults
    - Elements specific to modeling, like load shapes, growth curves, harmonic spectrums, line geometries, temperature curves, energy meters, and monitors
- Examples and basic wrappers for driving OpenDSS through the COM interface in several program languages including,
    - MATLAB
    - Python
    - Visual Basic for Applications through Excel
- A series of true-to-life distribution feeder models are provided in the EPRITestCircuits directory.
- A simple transmission model is provided in the Examples/Stevenson directory.

Additional introductory and training materials are also provided in the Training directory of the OpenDSS install path and in the Training directory on SourceForge, located here.

Reference Resources

OpenDSS provides several reference resources to extensively document the various features of the DSS scripting language and COM interface.

- The **in-program help interface**, accessible through Help > DSS Help or the ? button, provides a quick reference to the various properties of OpenDSS circuit elements and commands.
- The **OpenDSS manual**, located in Doc/OpenDSSManual.pdf of the OpenDSS installation directory, gives significantly more in depth information into OpenDSS features and interworkings than presented here as well as a comprehensive COM and OpenDSS language reference.
- The **OpenDSS TechNotes**, located in the Docs directory of the OpenDSS installation path are a set of white papers that give very specific and detailed information on the COM interface, OpenDSS features, and more esoteric applications of the program such as modeling transformer core effects and photovoltaic systems.
- The **OpenDSS wiki**, located at here gives additional information that can also be found in the white papers and manual. It is an especially good source of information on the COM interface.
- A **type library browser** (TLB) – such as the provided in Microsoft Office's Visual Basic Editor – can be used to expose the COM object. To do this, open the VBA editor when in Excel, for example, add a reference (under the Tools menu) to OpenDSSEngine.DSS. Then you will be able to browse the "library" for OpenDSSEngine using the "Object Browser" (in the View menu) in the VBA editor.

Additional Help

If you require additional help, the OpenDSS forums on SourceForge foster a thriving community to address your question. You can find the forums here.