Open Source Software for Simulating Active Distribution Systems

Roger C. Dugan
Sr. Technical Executive
EPRI, USA

OpenDSS

- EPRI has released its Distribution System Simulator (DSS) program as open source
- Called “OpenDSS”
- Can be found at:
  - WWW.SOURCEFORGE.NET
  - (Search for OpenDSS)
Why was DSS Developed?

- DSS was developed to provide a very flexible research platform and a foundation for special distribution analysis applications such as DG analysis.
- Fills gaps left by other distribution system analysis tools.
- Study new approaches to distribution system analysis.

Why OpenDSS?

- EPRI has made the DSS open source to:
  - Cooperate with other open source efforts in the USA in Smart Grid research
  - To encourage new advancements in distribution system analysis
  - To provide Smart Grid researchers a tool for testing algorithms
Current Related EPRI Activities

- Intelligrid
  - Distribution Fast Simulation & Modeling
  - DSE – Distribution State Estimator
- CIM/DCIM
- OpenDSS – Distribution System Simulator
  - Multipurpose distribution system analysis tool
  - Open source version released – 5 Sept 2008
  - Official release – November 2008

DSS Background

- Under development for more than 10 Years
  - Started at Electrotrek Concepts in 1997
  - Purchased by EPRI in 2004
- Objectives in 1997
  - Tool to support all distribution planning aspects of distributed generation
  - Implement a flexible research platform
  - Incorporate object-oriented data ideas
- Key Future work
  - Platform for DSE for North American Systems
  - Research platform for reliability tools
Distribution System Simulator (DSS)

The DSS is designed to simulate utility distribution systems in arbitrary detail for most types of analyses related to distribution planning.

- It performs its analysis types in the *frequency domain*,
  - Power flow,
  - Harmonics, and
  - Dynamics.
- It does NOT perform electromagnetic transients (*time domain*) studies.

Overall Model Concept
Example DSS Applications

- Neutral-to-earth (stray) voltage simulations.
- Loss evaluations due to unbalanced loading.
- Development of DG models for the IEEE Radial Test Feeders.
- High-frequency harmonic and interharmonic interference.
- Losses, impedance, and circulating currents in unusual transformer bank configurations.
- Transformer frequency response analysis.
- Distribution automation control algorithm assessment.
- Impact of tankless water heaters on flicker and distribution transformers.
- Wind farm collector simulation.
- Wind farm impact on local transmission.
- Wind generation and other DG impact on switched capacitors and voltage regulators.
- Open-conductor fault conditions with a variety of single-phase and three-phase transformer connections.

Annual Losses

Peak load losses are not necessarily indicative of annual losses.
Solar PV Simulation

Power Distribution Efficiency

Light Load Week

Peak Load Week
Wind Plant 1-s Simulation

Active and Reactive Power

Feeder Voltage and Regulator Tap Changes

Architecture
DSS Structure

DSS Object Structure
Power Delivery Elements

\[ I_{term} = [Y_{prim}] V_{term} \]

Power Conversion Elements

\[ f_{scan}(t) = F(V_{scan}(State), t) \]
Load (a PC Element)

(One-Line Diagram)

Putting it All Together

\[ Y = I_{\text{inj}} \]

\[ \text{PC Elements} \]

\[ I_1 \]
\[ I_2 \]
\[ I_m \]

\[ \text{Node Voltages} \]

\[ \text{Iteration Loop} \]
Scripting Basics

A Basic Script

New Circuit.Simple ! Creates voltage source (Vsource.Source)
Edit Vsource.Source BasekV=115 pu=1.05 ISC3=3000 ISC1=2500 !Define source V and Z
New Linecode.336ACSR R1=0.058 X1=.1206 R0=.1784 X0=.4047 C1=3.4 C0=1.6 Units=kft
New Line.LINE1 Bus1=Sub_Bus Bus2=LoadBus Linecode=336ACSR Length=1 Units=Mi
New Load.LOAD1 Bus1=LoadBus kV=12.47 kW=1000 PF=.95
Solve
Show Voltages
Show Currents
Show Powers kVA elements
Solution Modes

Distribution System Analysis Tools

• DSS has the basic tools for Planning built in:
  – Power Flow
  – Short Circuit Calculations
• In Addition, it has Several Advanced Capabilities
  – “Dynamic” Power Flow
  – Other power flow modes
  – Dynamics
  – Harmonics
• If it is not built in, you can drive it from another program such as Matlab
  – For example: Reliability Analysis
Classes of Solution Modes

- Power Flow
  - Snapshot
  - Direct
- Dynamic Power Flow
  - Daily
  - Yearly
  - DutyCycle
  - Peakday
- Dynamics
- Harmonics

- Other Power Flow
  - LD1
  - LD2
  - Monte Carlo
    - M1
    - M2
    - M3
- Short Circuit
  - Faultstudy
  - MF - Monte Carlo Fault

Introduction to Driving the COM Server from another Application
Active objects concept

- There is one registered In-Process COM interface:
  - `OpenDSSEngine.DSS`
    - That is, the DSS interface is the one you instantiate
    - The DSS interface creates all the others.
- The interfaces generally employ the idea of an **ACTIVE object**
  - Active circuit,
  - Active circuit element,
  - Active bus, etc.
  - The interfaces generally point to the active object
    - To work with another object, change the active object.

DSS Interface

This interface is instantiated upon loading `OpenDSSEngine.DSS` and then instantiates all other interfaces.

Call the Start(0) method to initialize the DSS.

DSS Class Functions (methods) and Properties.
Instantiate the DSS Interface and Attempt Start

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 the Text interface for easier access
    Set DSSText = DSSobj.Text
End If

End Sub

Accessing the COM Server

- In MATLAB:
  - DSSobj = actxserver('OpenDSSEngine.DSS');

- In VBA:
  - Public DSSobj As OpenDSSEngine.DSS
  - Set DSSobj = New OpenDSSEngine.DSS

- In PYTHON:
  - self.engine = win32com.client.Dispatch("OpenDSSEngine.DSS")
TEXT Interface

Interfaces as Exposed by VBA
Object Browser in MS Excel

Text interface is simplest

Text has two Properties

---

Assign a Variable to the Text Interface

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 the Text interface for easier access
    Set DSSText = DSSObj.Text
End If

End Sub
Now Use the Text Interface …

- You can issue any of the DSS script commands from the Text interface

  - 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
  - Set regulator tap change limits for IEEE 123 bus test case
    With DSSText
      .Command = "RegControl.creg1a.maxtapchange=1  Delay=15  !Allow only one tap change per solution."
      .Command = "RegControl.creg2a.maxtapchange=1  Delay=30  !Allow only one tap change per solution"
      .Command = "RegControl.creg3a.maxtapchange=1  Delay=30  !Allow only one tap change per solution"
      .Command = "RegControl.creg3c.maxtapchange=1  Delay=30  !Allow only one tap change per solution"
      .Command = "RegControl.creg4a.maxtapchange=1  Delay=30  !Allow only one tap change per solution"
      .Command = "RegControl.creg4b.maxtapchange=1  Delay=30  !Allow only one tap change per solution"
      .Command = "RegControl.creg4c.maxtapchange=1  Delay=30  !Allow only one tap change per solution"
      .Command = "Set MaxControlIter=30"
    End With

Result Property

- The Result property is a Read Only property that contains any result messages the most recent command may have issued.
  - Error messages
  - Requested values

  - Example: Query line length
    DSSText.Command = ""Line.L1.Length"
    S = DSSText.Result  ' Get the answer
    MsgBox S  ' Display the answer
User-Written Controls

From the COM Interface

Basic Control Loop Flow Chart

- Initialize Loop
- Solve Circuit
- Check Controls
- Next Time Step

If you set Number=1, you can break in here.

You can single-step through this.

Control Actions Done?

NO YES
Control Loop  (Actual Pascal Code)

FUNCTION TSolutionObj.SolveSnap:Integer;  // solve for now once
VAR
  TotalIterations :Integer;
Begin
  SnapShotInit;
  TotalIterations    := 0;
  REPEAT
    Inc(ControlIteration);
    Result := SolveCircuit;  // Do circuit solution w/o checking controls (Now check controls)
    CheckControls;
    {For reporting max iterations per control iteration}
    If Iteration > MostIterationsDone THEN MostIterationsDone := Iteration;
    TotalIterations := TotalIterations + Iteration;
    UNTIL ControlActionsDone or (ControlIteration >= MaxControlIterations);
    If Not ControlActionsDone and (ControlIteration >= MaxControlIterations) then Begin
      DoSimpleMsg('Warning Max Control Iterations Exceeded. ' + CRLF + 'Tip: Show Eventlog to debug control settings.', 485);
      SolutionAbort := TRUE;   // this will stop this message in dynamic power flow modes
    End;
    If ActiveCircuit.LogEvents Then LogThisEvent('Solution Done');
    Iteration := TotalIterations;  { so that it reports a more interesting number }
  End;

External Script and COM Interface Options

• Take Immediate action or keep track of time yourself
  – Set Number=1
  – Sample after solution step
  – Execute command to change element state

• Use the DSS Control Queue through COM Proxy
  – Set Number=1
  – Step through solution
  – Push control commands onto DSS control queue
    • (Allows DSS to keep track of when control actions happen)
  – Write routines to handle pending actions
Control Proxy in COM Interface

COM Interface Control Proxy Operation

CONTROL OBJECT

ACTION 1

ACTION 2

CONTROL DISPATCHER

DoPendingAction

POP

COM CONTROL Proxy

DoPendingAction

Action List

USER CODE

POP

PUSH

ALL PUSHES ARE SORTED BY TIME

CONTROL OBJECT

ACTION 3

ACTION N

CONTROL OBJECT

DoPendingAction

Time

Action Code

Proxy Handle

Control Addr

DSS Control Queue

For More Information

See Wiki

Download “Training” from SOURCEFORGE site