Table of Contents

1 Background

OpenDSS is a distribution system simulator that is generally comparable to GridLAB-D, although each program has advantages and disadvantages over the other [1]. It was developed with EPRI funding as a Windows-only application, using commercial Windows-only tools. This memo summarizes the build process of a console-mode, cross-platform version using GCC, FPC and Lazarus. Tested platforms include Windows 10, Ubuntu 18.04 and Mac OS X High Sierra.

OpenDSS was written in Delphi (a Windows-specific version of Pascal), and it requires the KLUSolve sparse matrix solver, which was written in C/C++. After compiling KLUSolve with GCC, it's then possible to build OpenDSS with Free Pascal (FPC) 3.0.4 on Windows, Linux and Mac OS X. Modern versions of Pascal are object-oriented, but simpler than C++ and Java. There are syntax differences, but a developer who is already comfortable with C++ or Java can be productive in Pascal, too. See http://wiki.freepascal.org/Lazarus_Documentation#Lazarus_and_Pascal_Tutorials. There is a Free Pascal IDE and cross-platform GUI toolkit called Lazarus, but this is not required for OpenDSS.

With a common code base, there can still be differences between the native and cross-platform versions. For example, EPRI has implemented a Windows-only parallelization scheme, and continues to develop Windows-only automation interfaces. NREL uses a shared-library version of OpenDSS called by Python and/or Julia. On the other hand, PNNL is implementing a FNCS interface and command-line history only for the cross-platform version. However, both versions maintain the same core modeling and analysis functions.

[1] R. C. Dugan and T. E. McDermott, "An open source platform for collaborating on smart grid research," in *Power and Energy Society General Meeting, 2011 IEEE*, 2011, pp. 1-7.

1.1 Running Cross-platform OpenDSS

1. Enter "opendsscmd" from a command prompt
   a. The program's >> prompt will appear. Enter any OpenDSS command(s) from this prompt
   b. Up and down arrows navigate through the command history
   c. Enter "help" from the >> prompt for the built-in help
   d. Enter "exit", "q" or an empty line from >> to exit
2. You can enter "opendsscmd filename.dss" from a command prompt. This runs the OpenDSS commands in filename.dss, and then exits immediately.
3. You can enter "opendsscmd –f" from a command prompt; this enters a FNCS time step loop.
4. You can enter "opendsscmd –f filename.dss" from a command prompt. This runs the OpenDSS commands in filename.dss, and then enters a FNCS time step loop.

1.2 Open Issues

1. The regular expressions for the batchedit command, which are implemented in ExecHelper.pas, have become case-sensitive. They need to be made case-insensitive.

1.3 Code Repositories

The source code is under version control at SourceForge and GitHub. The code repositories for OpenDSS, the sparse matrix solver and the command-line history are at:

- https://github.com/pnnl/linenoise-ng.git
- https://sourceforge.net/projects/klusolve/
- https://sourceforge.net/projects/electricdss/

It's assumed that all three repositories will be cloned under a common location, such as ***~/src***

2 Streamlined Build Instructions

These instructions assume that GridLAB-D and (optionally) FNCS already build on the target computer. CMAKE and an SVN client are also required. These instructions were tested on Mac OS X High Sierra, Ubuntu 16.04 and Windows 10. On Windows, if not building GridLAB-D, the required preliminary steps are described in a later section. On Windows, some steps must be done from an MSYS2 terminal, not the regular Command Window. The Lazarus IDE may be of interest for those actively developing for OpenDSS, and this is also described later.

2.1 Prerequisite: GridLAB-D and FNCS are Built

This is the recommended build process for OpenDSS with FNCS. First, complete the steps for your operating system at https://tesp.readthedocs.io/en/latest/Building_TESP.html

1. Install FPC 3.0 or higher:
   a. Linux: ***sudo apt-get install fpc***
   b. Windows: from https://sourceforge.net/projects/freepascal/files/Win32/3.0.4, you need the base win32 installation and then the x86_64 cross-compiler
   c. Mac: from https://sourceforge.net/projects/freepascal/files/Mac OS X/3.0.4/, you need the intel-macosx.dmg installer
2. Build linenoise-ng (starting in ~/src or c:) (use MSYS2 on Windows)

mkdir linenoise-ng

cd linenoise-ng

git clone https://github.com/pnnl/linenoise-ng.git .

mkdir build

cd build

cmake -DCMAKE_BUILD_TYPE=Release ..

on Windows, add ***-G"MSYS Makefiles"***

make

sudo make install *on Windows, this writes to c:\Program Files (x86); not what we want*

a. Linux only (may defer this to step 3f below): ***ldconfig***

3. Build KLUSolve (use MSYS2 on Windows; **pacman -S --needed svn unzip**)
   a. ***mkdir KLUSolve***
   b. cd KLUSolve
   c. svn checkout https://svn.code.sf.net/p/klusolve/code/ .
   d. Manually ***mkdir Lib*** or ***md Lib***
   e. Mac OS X only: manually ***mkdir KLUSolve/Obj***
   f. Issue ***make all***
      i. On Mac OS X ***make all*** usually fails to link the first time due to a race condition on copying files. If that happens, just ***make all*** again.
   g. Manually install the shared object library (TODO: put this in the Makefile)
      i. Linux static: ***cp Lib/libklusolve.so /usr/local/lib*** and then ***ldconfig***
      ii. Windows: copy Lib/libklusolve.dll to the target location, either the opendsscmd location or a Windows system directory
      iii. Mac: ***cp Lib/libklusolve.dylib /usr/local/lib***
   h. Test the demo program:
      i. Linux, Mac OS X, or MSys prompt: from Test subdirectory ***bash run_concat.sh***
         1. If the unzip command is not available on Windows MSYS, try step ii below
      ii. Windows Command Prompt: from Test subdirectory, manually unzip kundert_test_matrices.zip and then invoke ***run_concat.bat***
4. Linux: symbolic links are probably required, based on the contents of /usr/lib. For example:
   a. sudo ln -sv /usr/lib/x86_64-linux-gnu/libstdc++.so.6.0.25 /usr/lib/x86_64-linux-gnu/libstdc++.so
   b. sudo ln -sv /lib/x86_64-linux-gnu/libgcc_s.so.1 /lib/x86_64-linux-gnu/libgcc_s.so
5. Extract the selected OpenDSS source code (starting in ~/src or c:). Because EPRI keeps large build products, Windows-only artifacts and copy-paste code branches in the same repository, two different strategies are provided. (Note: on Windows, this step can be done in either MSYS or the Command Prompt).
   a. ***mkdir OpenDSS*** or ***md OpenDSS***
   b. cd OpenDSS
   c. Strategy 1: grab everything with
      i. svn checkout https://svn.code.sf.net/p/electricdss/code/trunk .
   d. Strategy 2: selective retrieval (could be the basis of a cross-platform installer)

i. **svn checkout --depth immediates https://svn.code.sf.net/p/electricdss/code/trunk/Version7 .**
ii. svn update --set-depth infinity Doc
iii. svn update --set-depth infinity Test
iv. svn update --set-depth infinity Distrib/Doc
v. svn update --set-depth infinity Distrib/EPRITestCircuits
vi. svn update --set-depth infinity Distrib/IEEETestCases
vii. svn update --set-depth infinity Distrib/Examples
viii. **svn update --set-depth infinity Source/Common**
ix. **svn update --set-depth infinity Source/CMD**
x. svn update --set-depth infinity Source/DDLL
xi. **svn update --set-depth infinity Source/Controls**
xii. **svn update --set-depth infinity Source/Executive**
xiii. **svn update --set-depth infinity Source/General**
xiv. **svn update --set-depth infinity Source/generics.collections**
xv. **svn update --set-depth infinity Source/Meters**
xvi. **svn update --set-depth infinity Source/Shared**
xvii. **svn update --set-depth infinity Source/Parser**
xviii. **svn update --set-depth infinity Source/PCElements**
xix. **svn update --set-depth infinity Source/PDElements**

6. Build and test OpenDSS
   a. **cd *./Source/CMD*** in your command prompt
   b. ***mkdir units*** or ***md units***
   c. ***chmod +x *.sh*** (for the first time on Linux or Mac OS X)
   d. ***./build.sh*** or ***build.bat***
   e. ***cd test***
   f. ***chmod +x *.sh*** (for the first time on Linux or Mac OS X)
   g. ***./opendsscmd*** and see if the command history works. Type "q" or ctrl-C to exit.
      i. libgcc_s_seh-1.dll might not be found on Windows. In that case, unzip https://github.com/pnnl/tesp/blob/master/install/Windows/MinGWredist.zip into c:\opendsscmd
   h. ***./opendsscmd IEEE13Nodeckt.dss***. It should solve a 13-bus circuit, exit, and open a text editor on the voltage results. However, this command may fail to find the circuit, especially if you skipped step 6e. In that case, try again. This is Open Issue #3.
   i. If FNCS is installed, test that connection with ***./test_fncs.sh*** or ***test_fncs.bat***. Look for results in ****.log*** and ***tracer.out***.

2.2 Using Lazarus instead of FPC from the Command Line

Instead of the build scripts, you will open the project file ~/src/OpenDSS/Source/CMD/opendsscmd.lpi from the Lazarus IDE. The IDE provides more convenient management of project files, builds and error messages. To install the IDE on Ubuntu, ***sudo apt-get install lazarus*** works. On Windows and Mac, you can download a combined package of Lazarus 1.6 and FPC 3.0 from http://www.lazarus-ide.org/

On the Mac, pay close attention to http://wiki.freepascal.org/Installing_Lazarus_on_MacOS_X for setting up gdb. When you start the Lazarus IDE for the first time; it should find the debugger (gdb) and

possibly two compilers. **Choose the fpc compiler**, not the default ppc386 compiler. Otherwise, you can only make 32-bit executables from Lazarus. If necessary, you can fix this later from the IDE Tools / Options menu. However, the Lazarus IDE on Mac does not fully support Carbon, meaning that it can practically only create 32-bit GUI applications. This is a significant barrier to the possible cross-platform GUI for OpenDSS, at least one based on free development tools.

2.3 Preliminary Steps for MSYS on Windows; not building GridLAB-D

These were tested on Windows 10 and Windows 7:

1. Download and install 64-bit GCC from http://tdm-gcc.tdragon.net
2. Download and install just MSYS 1.0.11 from www.mingw.org/wiki/msys. During the "post install" procedure:
   a. answer [y] that MinGW is installed
   b. answer c:/TDM-GCC-64 for the location of MinGW
3. If you don't have an SVN client, install from https://tortoisesvn.net/
   a. On the "Custom Setup" page, select "Command Line Tools" for inclusion
4. If you don't have a Git client, install from https://www.sourcetreeapp.com/
   a. To use the command line Git from SourceTree on Windows, you may use the "Actions/Open in Terminal" menu command
   b. You may also install the command line Git from https://git-scm.com/downloads
5. If you don't have Cmake, install from https://cmake.org/
   a. Choose to add Cmake to the system path
   b. Change location to c:\cmake\, which is better for MSYS

Now that the GridLAB-D project has adopted MSYS2, the process in section 2.1 will be much easier.

2.4 Other OpenDSS Development Branches

EPRI's parallelization, which is Windows only and requires Delphi.

- https://sourceforge.net/p/electricdss/code/HEAD/tree/trunk/Parallel_Version/

A mirrored-repository Makefile:

- https://github.com/Muxelmann/OpenDSSDirect.make

NREL's Python interface to the direct-call, shared library version:

- https://github.com/NREL/OpenDSSDirect.py/

EPRI's Julia interface to the direct-call, shared library version:

- https://github.com/tshort/OpenDSSDirect.jl

These last two NREL and EPRI interfaces to the shared library are using mirrored repositories. If these interfaces are of interest to PNNL developers, please use one of the following options, both of which reference the main repository:

- From ~/src/OpenDSS/Source/DDLL, use ***./build.sh*** or ***build.bat***
- From the Lazarus IDE, open the project file ~/src/OpenDSS/Source/DDLL/OpenDSSDirect.lpi

3 FNCS Example for OpenDSS

Input files for this example may be found under ***Version7/Source/CMD/Test*** of the repository. With reference to Figure 1, a 285-kW solar generator has been added to bus 634, and then a cloudy day is simulated at 1-second time steps. At mid-day, the normally-closed switch from 671 to 692 opens, which drops part of the load. The total feeder load is plotted to the right of Figure 1, and it shows the effect of both PV generation during daylight hours, and the load shedding at mid-day. Otherwise, the feeder load would have been constant at about 3600 kW.

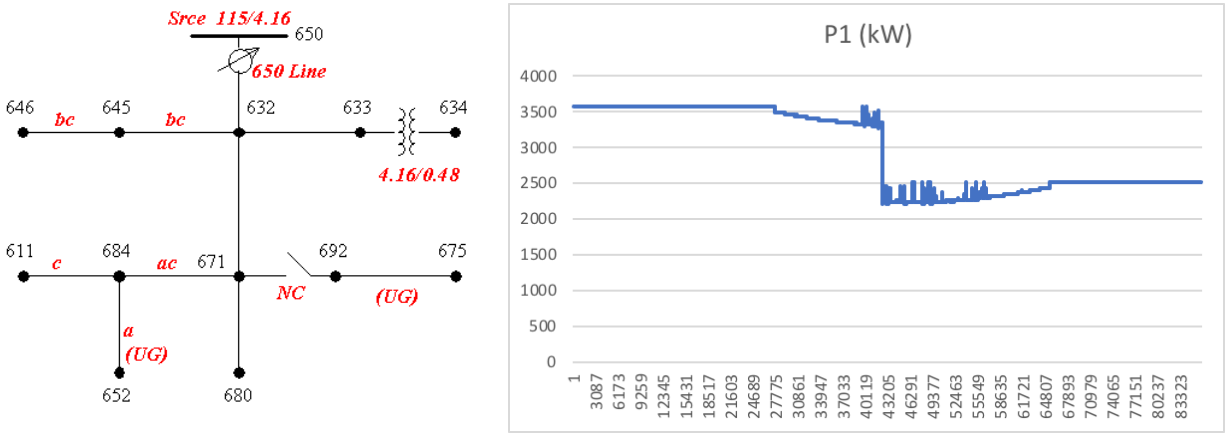

Figure 1: Daily simulation of IEEE 13-bus system at 1-second time steps. This feeder's peak load is about 3600 kW. Here, 285 kW of solar PV generation has been added to bus 634. The normally closed (NC) switch opens at about noon.

The bash script that runs this example on Mac or Linux follows (a similar batch file is under development for Windows), as ***test_fncspv.sh***. The three federates are OpenDSS, a FNCS player to pipe scripted commands into OpenDSS, and a FNCS tracer to log publications from OpenDSS.

```
(exec fncs_broker 3 &> brokerpv.log &)
(exec fncs_player 25h opendss.playerpv &> playerpv.log &)
(export FNCS_CONFIG_FILE=tracer.yaml && exec fncs_tracer 25h tracerpv.out &> tracerpv.log &)
(export FNCS_CONFIG_FILE=opendss.yaml && exec ./opendsscmd -f 25h &> opendsspv.log &)
```

The FNCS configuration file for OpenDSS, ***opendss.yaml***, is show below.

```
name: opendss
time_delta: 1s
broker: tcp://localhost:5570
values:
    command:
        topic: player/command
        default: 0
        list: true
```

For now, OpenDSS only subscribes to scripted text commands, which is enough for the GridAPPS-D and TESP use cases. The only likely changes are highlighted in red, for the FNCS port and the federate that's going to be issuing commands to OpenDSS.

The scripted commands are found in ***opendss.playerpv***, reproduced below. The commands from 0 to 9 ns set up the simulation according to OpenDSS syntax. See the documentation that comes with

OpenDSS for more details on the syntax and features. The command at 10 ns requests the start of an 86400s simulation, which actually begins at 1s or 1e9 ns.

```
#time topic   value
0 command redirect IEEE13Base.dss
1 command new loadshape.pvshape npts=86401 sinterval=1 mult=(file=pvshape.dat) action=normalize
2 command new pvsystem.pv1 bus1=634 phases=3 kV=0.48 irradiance=1 pmpp=285 kVA=300 daily=pvshape
3 command new monitor.pv1v  element=pvsystem.pv1 terminal=1 mode=96
4 command new monitor.pv1pq element=pvsystem.pv1 terminal=1 mode=65 PPolar=NO
5 command new monitor.fdrpq element=line.650632 terminal=1 mode=65 PPolar=NO
6 command solve
7 command export summary pvsnap_summary.csv
8 command set controlmode=static
9 command set maxcontroliter=1000
10 command  solve mode=daily stepsize=1s number=86400
42401500000000 command open Line.671692 1
86401000000000 command export monitors pv1v
86401000000000 command export monitors pv1pq
86401000000000 command export monitors fdrpq
86401000000000 command quit
```

At the end of each time step, which is 1s, a function called ***TSolutionObj.Increment_time*** calls back to another function that issues ***fncs_time_request***. For nearly all of the time steps, the callback function returns immediately and allows OpenDSS to just take the next time step. However, at 42402s, the command "open Line.671692 1" has been received from FNCS. The time request is granted, but before returning control to the time step loop, the callback function executes that command to open a switch. We see the effect of this plotted in Figure 1.

The last four scripted commands will execute just after the time step looping has completed. These export the monitor data to CSV files, and then quit OpenDSS. This happens before the 25 hours specified in ***test_fncspv.sh***, but the broker and each federate complete an orderly shutdown.

3.1 Work to do for FNCS Publications

We need to have a text file of attributes to publish over FNCS, which is how GridLAB-D handles both subscriptions and publications. With OpenDSS, the differences are:

- FNCS.pas should read the publications file, then build maps to objects within OpenDSS for each publication, where it makes sense for efficiency. If the network topology changes, it may be necessary to rebuild this map (OpenDSS will trigger on topology changes).
- There is no conception of "commit" or other stages of a time step. At ***Increment_time*** and a few other points, OpenDSS should publish the following types of value:
    - Bus voltages
    - Branch currents
    - EnergyMeter registers
    - Sensor registers
    - Switch statuses
    - CapControl on/off statuses
    - Tap changer positions
- The COM interface implementations in the Source/DLL directory provide some examples of how to get the specific values to publish over FNCS.
- We need to give some thought on data structures to efficiently publish many values.

4 Source Code Directories for OpenDSS

The main files of interest for work done to date are:

- Source/CMD/CmdForms.pas – non-graphical alternatives to the GUI components that are sprinkled throughout other source files
- Source/CMD/FNCS.pas – loads and uses the FNCS shared library, i.e., dll, dylib or so file.
- Source/CMD/opendsscmd.lpr – main source file for the opendsscmd application
- Source/Common/Solution.pas – the *Increment_time* function is here

OpenDSS is primarily a power flow program. A top-level roadmap to the source code sub-directories follows. Please consider any other source directories you may see to be orphaned, or otherwise deprecated.

- Source/CMD – the Lazarus/FPC version
- Source/Common – circuits, buses and solutions. The CIM export code is here.
- Source/Controls – regulator, capacitor, inverter, switch and other controls. The controls themselves do not carry power
- Source/DDLL – a direct-DLL interface, which we are partly implementing in FNCS and HELICS
- Source/DLL – a Windows COM interface, which we are partly implementing in FNCS and HELICS
- Source/EXE – a Windows standalone application
- Source/Executive – executes the scripted commands, contains the on-line help text
- Source/Forms – GUI components, to be avoided in the Lazarus/FPC version
- Source/General – elements that support PC and PD elements, including line codes, wires, spacings, transformer codes, loadshapes, curves, etc.
- Source/IndMach012a – a dynamic induction machine model that we are not using, but it does provide an example of interfacing a new component to OpenDSS as a DLL
- Source/Meters – sensors, monitors and energy meters (these are not billing meters as in GridLAB-D)
- Source/Parser – parses text input
- Source/PCElements – power conversion (PC) elements like load, generation, PV, storage
- Source/PDElements – power delivery (PD) elements like transformers and lines, also capacitors.
- Source/Plot – plots the circuit layouts and output values; not supported in Lazarus/FPC version
- Source/Shared – supporting complex numbers, hash lists, registry access, etc.
- Source/TCP_IP – counterpart of the GridLAB-D server mode; which we are not using
- Source/TPerlRegEx – supports the batchedit command
- Source/x64 and Source/x86 – contains build products; we don't do this but have been unable to talk EPRI out of archiving build products.