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, Ubuntu 20.04, Centos 8, 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 a C/C++ compiler, it's then possible to build OpenDSS with Free Pascal (FPC) 3.2 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 implemented native FNCS [2] and HELICS [3] interfaces, and command-line history only for the cross-platform version. In addition, the cross-platform version has the most up-to-date support for Common Information Model (CIM) export [4]. However, both versions maintain the same core modeling and analysis functions.

Install the appropriate version from https://sourceforge.net/projects/electricdss/files/OpenDSSCmd/. This includes a selection of documentation, examples and tech notes on non-graphical features.

To run the program:

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.

2 Streamlined Build Instructions

These instructions were tested on Mac OS X, Ubuntu, Centos, and Windows 10. The streamlined build includes OpenDSSCmd, FNCS and HELICS. On Linux and Mac OS X, the build uses the GNU C/C++ compiler, gcc/g++ version 7 or later. On Windows, the build uses Microsoft Visual Studio Build Tools 2019 or later, such that MSYS2 is no longer necessary. On all platforms, FPC, CMAKE, Git and an SVN client are required. All build steps can be done from a command prompt or terminal window. The Lazarus IDE is optional for those actively developing for OpenDSS, as described later.

These instructions no longer assume that GridLAB-D is built first on the target computer. You can install or build packages that include both OpenDSSCmd and GridLAB-D, with FNCS and HELICS interfaces, from the Transactive Energy Simulation Platform (https://github.com/pnnl/tesp/tree/develop) or from GridAPPS-D (https://github.com/GRIDAPPSD). GridLAB-D itself can be installed alongside OpenDSS from this link: https://sourceforge.net/projects/gridlab-d/, but without FNCS and HELICS interfaces.

2.1 Linux Build Environment

On Ubuntu, use the package manager to install several packages:

```
sudo apt-get -y install git
sudo apt-get -y install git-lfs
sudo apt-get -y install build-essential
sudo apt-get -y install subversion
sudo apt-get -y install cmake
sudo apt-get -y install libzmq5-dev
sudo apt-get -y install libczmq-dev
sudo apt-get -y install fpc
```

On Centos (version 8 tested):

```
sudo dnf clean all
sudo dnf update -y
sudo dnf install zeromq-devel -y
sudo dnf install czmq-devel -y
sudo dnf install fpc -y
```

The CMAKE version must be at least 3.18. If not, update with installer script (tested Ubuntu 18.04):

- sudo apt-get remove cmake
- install the latest version from https://cmake.org/download/ and add to your PATH

Or build the latest CMAKE from source (tested Centos 8):

```
sudo dnf remove cmake
sudo dnf install openssl
sudo dnf install openssl-devel
cd ~/src
wget https://github.com/Kitware/CMake/releases/download/v3.19.2/cmake-3.19.2.tar.gz
tar -zxvf cmake-3.19.2.tar.gz
cd cmake-3.19.2
./bootstrap
make
sudo make install
cmake --version
```

The FPC version must be at least 3.2.0. If not (tested Ubuntu 18.04; Centos 8 already installs 3.2.0):

- sudo apt-get remove fpc
- from https://sourceforge.net/projects/freepascal/files/Linux/, download, unzip and install ***x86_64-linux.tar*** installer

2.2 Mac OS X Build Environment

Use Xcode and HomeBrew to install several packages:

```
xcode-select –-install
/bin/bash -c "$(curl -fsSL
https://raw.githubusercontent.com/Homebrew/install/master/install.sh)"
brew install subversion
brew install gdb
brew install cmake
brew install zmq
brew install czmq
brew install fpc
```

These package versions are currently up to date for OpenDSSCmd. If not, please see http://gridlab-d.shoutwiki.com/wiki/Mac_OSX/Setup or https://wiki.freepascal.org/Installing_Lazarus_on_macOS for more information. See https://sourceforge.net/projects/freepascal/files/Mac%20OS%20X/ for the latest intel-macosx.dmg installer.

2.3 Windows Build Environment

1. Install C/C++ compiler and Cmake from: https://visualstudio.microsoft.com/downloads/ (find *Build Tools for Visual Studio 2019* under *Tools for Visual Studio 2019*)
2. If you don't have an SVN client, install from https://tortoisesvn.net/. On the "Custom Setup" page, select "Command Line Tools" for inclusion.
3. If you don't have a Git client, install from https://www.sourcetreeapp.com/. To use the command line Git from SourceTree on Windows, you may use the "Actions/Open in Terminal" menu command. You may alternatively install the command line Git from https://git-scm.com/downloads
4. Install the latest FPC from https://sourceforge.net/projects/freepascal/files/Win32/. You should install 32-bit and 64-bit compilers from, e.g., ***fpc-3.2.2.win32.and.win64.exe***.

2.4 Supporting Code Repositories

It's assumed that all repositories will be cloned under a common location, such as ***~/src*** or ***c:\src***. From a terminal window in that directory, use these commands to pull the correct branches:

```
git clone -b feature/opendss https://github.com/FNCS/fncs.git
git clone https://github.com/GMLC-TDC/HELICS-src.git
git clone https://github.com/pnnl/linenoise-ng.git
svn checkout https://svn.code.sf.net/p/klusolve/code
```

On Windows only:

```
git clone https://github.com/zeromq/libzmq.git
git clone https://github.com/zeromq/czmq.git
```

2.5 OpenDSS Repository

Because EPRI keeps large build products, Windows-only artifacts and copy-paste code branches in the same repository, two different strategies are provided. Both start in ***~/src*** or ***c:\src***. Strategy 1 is to, "grab everything" for building the executable and everything else in the repository:

```
svn checkout https://svn.code.sf.net/p/electricdss/code/trunk
```

Strategy 2, assumed for the rest of this document, is a "selective retrieval" for building only the executable from the active Version 8 branch. The phrase "Version8" will not appear in your local path names this way. The un-bold steps are optional, but useful for testing and developing installers.

```
mkdir OpenDSS or md OpenDSS
cd OpenDSS
svn checkout --depth immediates https://svn.code.sf.net/p/electricdss/code/trunk/Version8 .
svn update --set-depth infinity Doc
svn update --set-depth infinity Distrib/BitrockInstaller
svn update --set-depth infinity Distrib/Doc
svn update --set-depth infinity Distrib/EPRITestCircuits
svn update --set-depth infinity Distrib/Examples
svn update --set-depth infinity Distrib/IEEETestCases
svn update --set-depth infinity Source/Common
svn update --set-depth infinity Source/CMD
svn update --set-depth infinity Source/Controls
svn update --set-depth infinity Source/Executive
svn update --set-depth infinity Source/epiktimer
svn update --set-depth infinity Source/General
svn update --set-depth infinity Source/GISCommands
svn update --set-depth infinity Source/Meters
svn update --set-depth infinity Source/Shared
svn update --set-depth infinity Source/Parallel_Lib
svn update --set-depth infinity Source/Parser
svn update --set-depth infinity Source/PCElements
```

```
svn update --set-depth infinity Source/PDElements
svn update --set-depth infinity Source/TCP_IP
```

A third strategy uses git for a "full retrieval except for binaries", ignoring part of the revision history that was corrupted on SourceForge. See https://github.com/dss-extensions/electricdss-tst for an example.

2.6 Linux and Mac OS X Builds

Build FNCS (library, broker, player, and tracer only, with CMake):

```
cd ~/src/fncs
mkdir build
cd build
cmake -DCMAKE_BUILD_TYPE=Release ..
make
sudo make install
```

Build HELICS:

```
cd ~/src/HELICS-src
mkdir build
cd build
cmake -DBUILD_SHARED_LIBS=ON -DHELICS_DISABLE_BOOST=ON -DCMAKE_BUILD_TYPE=Release ..
git submodule update --init
make -j4
sudo make install
helics_broker -–version  # should return a version number for "pip3 install helics==version"
```

Build linenoise-ng for OpenDSSCmd:

```
cd ~/src/linenoise-ng
mkdir build
cd build
cmake -DCMAKE_BUILD_TYPE=Release ..
make
sudo make install
```

Build KLUSolve for OpenDSSCmd:

```
cd ~/src/KLUSolve
mkdir build
cd build
cmake -DCMAKE_BUILD_TYPE=Release ..
make
sudo make install
```

Setup Libraries (Linux):

You may need to add /usr/local/lib to the path for shared libraries (*.so). If that's the case, add a file called *opendsscmd.conf* with contents */usr/local*/lib to the */etc/ld.conf.d* directory. Either way:

```
sudo ldconfig
```

(then logout and login)

Build OpenDSSCmd:

```
cd ~/src/OpenDSS/Source/CMD
mkdir units    # only required once
chmod +x *.sh  # only required once
./build.sh
```

Test OpenDSSCmd:

```
cd test
chmod +x *.sh  # only required once
```

See if the command history works. Type "q" or ctrl-C to exit.

```
./opendsscmd
```

Solve a 13-bus circuit, exit, and open a text editor on the voltage results.

```
./opendsscmd IEEE13Nodeckt.dss
```

If FNCS is installed, test that connection. Look for results in *.log and tracer.out.

```
./test_fncs.sh
```

2.7 Windows Builds

Be sure to open the **x64 Native Tools** Command Prompt for VS 2019. (The default is 32-bit, if you open directly from the Microsoft installer). Libraries will be installed into **c:\cmdtools**, but this can be changed in the following scripts.

- Add **c:\cmdtools\bin** to the path for testing purposes

Build libzmq for FNCS and HELICS:

```
cd c:\src\libzmq
md build
cd build
cmake .. -DBUILD_STATIC=OFF -DBUILD_SHARED=ON -DZMQ_BUILD_TESTS=OFF \
         -DENABLE_CPACK=OFF -DWITH_PERF_TOOL=OFF -DCMAKE_INSTALL_PREFIX=C:\cmdtools
cmake --build . --config Release --target install
```

Build czmq for FNCS:

```
cd c:\src\czmq
md build
cd build
cmake .. -DCZMQ_BUILD_SHARED=ON -DCZMQ_BUILD_STATIC=OFF -DCMAKE_PREFIX_PATH=C:\cmdtools \
         -DCMAKE_INSTALL_PREFIX=C:\cmdtools
cmake --build . --config Release –-target install
```

Build FNCS:

```
cd c:\src\FNCS
md build
cd build
cmake .. -DCMAKE_INSTALL_PREFIX=C:\cmdtools
cmake –-build . –-config Release –-target install
```

Build HELICS:

```
cd c:\src\HELICS-src
md build
cd build
cmake .. -DBUILD_SHARED_LIBS=ON -DHELICS_DISABLE_BOOST=ON -DHELICS_ZMQ_SUBPROJECT=OFF \
         -DCMAKE_INSTALL_PREFIX=C:\cmdtools
git submodule update --init
cmake --build . --config Release --target install
helics_broker –-version  # should return a version number for "pip install helics==version"
```

Build linenoise-ng for OpenDSSCmd:

```
cd c:\src\linenoise-ng
md build
cd build
cmake .. -DCMAKE_INSTALL_PREFIX=C:\cmdtools
cmake --build . --config Release –-target install
```

Build KLUSolve for OpenDSSCmd:

```
cd c:\src\KLUSolve
md build
cd build
```

```
cmake .. -DCMAKE_INSTALL_PREFIX=C:\cmdtools
cmake --build . --config Release –-target install
```

Build OpenDSSCmd:

```
cd c:\src\OpenDSS\Source\CMD
md units    :: only required once
build.bat
```

Test OpenDSSCmd:

```
cd test
```

See if the command history works. Type "q" or ctrl-C to exit.

```
opendsscmd
```

Solve a 13-bus circuit, exit, and open a text editor on the voltage results.

```
opendsscmd IEEE13Nodeckt.dss
```

If FNCS is installed, test that connection. Look for results in *.log and tracer.out.

```
test_fncs.bat
```

Copy this executable to the path.

```
copy opendsscmd.exe c:\cmdtools\bin
```

2.8 Installer Builds

The Bitrock Installer has been acquired by VMWare. It runs on Linux, Mac OS X and Windows; it can then build the installers for all three platforms. On a checkout of the full repository, the installer project file will be found at ***~/src/OpenDSS/Distrib/BitrockInstaller/OpenDSSCmd.xml***, with supporting scripts in the same directory. Upload to https://sourceforge.net/projects/electricdss/files/OpenDSSCmd/

- If building in a TESP-like environment, use *$TESP_INSTALL=/usr/local* for FNCS and HELICS
- To sync the Python interface, use *pip install helics==3.0.1*, where 3.0.1 is the result of *helics_broker --version*

3 FNCS and HELICS Examples

3.1 PV and Switching on IEEE 13-Bus Feeder

Input files for this example may be found under ***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.

The bash script that runs this example on Mac or Linux follows, 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 shown below. 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.

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

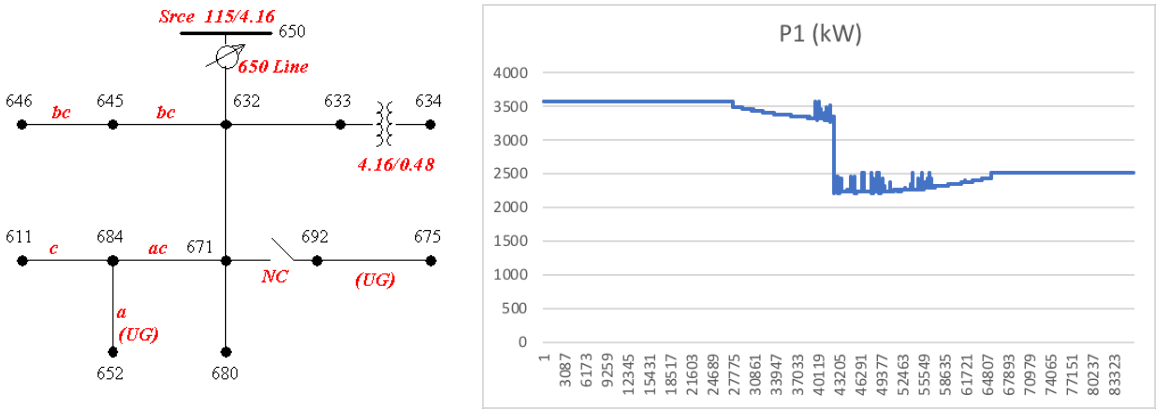

Figure 1: Daily simulation of IEEE 13-bus system at 1-second time steps. This feeder's peak load is about 3600 kW. Here, 285 kW of solar PV generation has been added to bus 634. The normally closed (NC) switch opens at about noon.

The 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.

3.2 Adding Houses to the PV Example

This example depends on a recent installation of GridLAB-D, with the FNCS and/or HELICS interface. These interfaces are not installed with the SourceForge download of GridLAB-D, but they are included with TESP, or they may be built from source. This example adds a center-tap transformer and house load to the 13-bus model, in lines 2-3 of the following player file. The PV does not vary, but the switch still opens midway through the simulation.

gldopendsshelics.playergld

```
#time   topic          value
-1      command "redirect IEEE13Base.dss"
1 command "new XfmrCode.CT5  phases=1 windings=3 kvs=[2.4 0.12 0.12] kVAs=[5.0 5.0 5.0] %imag=0.5
%Rs=[0.6 1.2 1.2] %noloadloss=0.2 Xhl=2.04 Xht=2.04 Xlt=1.36"
2 command "new Transformer.Tpoletop XfmrCode=CT5 buses=[680.1 house.1.0 house.0.2]"
3 command "new Load.F1_house_B0 phases=2 Bus1=house.1.2 kv=0.208 conn=wye model=1 kW=0.1 kvar=0"
4 command "new pvsystem.pv1 bus1=634 phases=3 kV=0.48 irradiance=1 pmpp=285 kVA=300"
5 command "new monitor.pv1v  element=pvsystem.pv1 terminal=1 mode=96"
6 command "new monitor.pv1pq element=pvsystem.pv1 terminal=1 mode=65 PPolar=NO"
7 command "new monitor.fdrpq element=line.650632 terminal=1 mode=65 PPolar=NO"
8 command "new monitor.hsepq element=load.F1_house_B0 terminal=1 mode=65 PPolar=NO"
9 command solve
10 command     "export summary pvsnap_summary.csv"
11 command     "set controlmode=static"
12 command     "set maxcontroliter=1000"
13 command      "helicspub fname=houses.json"
14 command      "solve mode=daily stepsize=1s number=86400"
42401500000000 command "open Line.671692 1"
86400999999990 command "export monitors pv1v"
86400999999992 command "export monitors pv1pq"
86400999999994 command "export monitors fdrpq"
86400999999996 command "export monitors hsepq"
86401000000000 command quit
```

The load is determined at run-time, through the house instantiated in ***HousesHelics.glm***. Only recent builds of GridLAB-D support houses that respond to voltage, without being connected to a GridLAB-D power flow component, as done in this example. The next two listings configure HELICS messages for OpenDSS publishing voltages to the GridLAB-D house panel, and subscribing to power from the GridLAB-D house panel. See Figure 2 for a sample output from ***test_helicsgld.sh***.

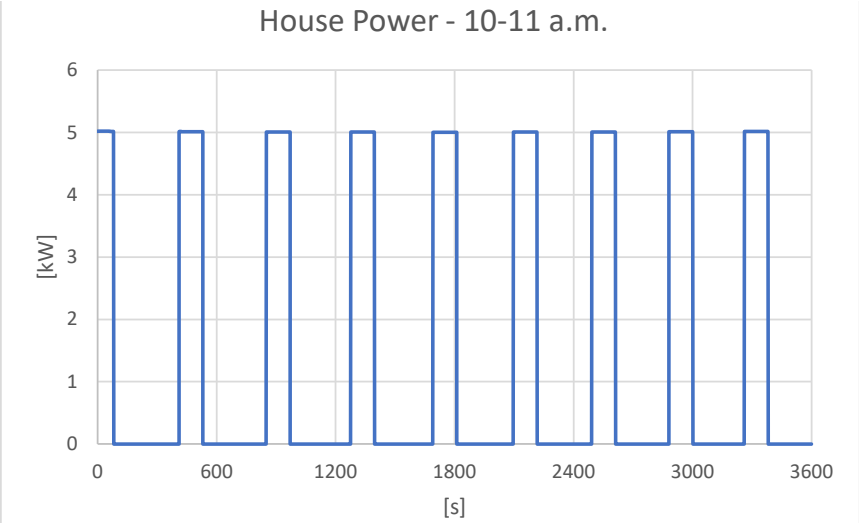

Figure 2: OpenDSS house load power, subscribed from GridLAB-D

gldhouseshelics.json (House topics only)

```
{
  "coreType" : "zmq",
  "name" : "gridlabd",
  "period" : 1,
  "loglevel" : 7,
  "publications" : [
    {
      "global" : false,
      "key" : "F1_house_B0/load",
      "type" : "complex",
      "info" : "{\"object\":\"F1_house_B0\",\"property\":\"power\"}"
    }
  ],
  "subscriptions" : [
    {
      "key" : "opendss/bus/house/voltage/A",
      "type" : "complex",
      "required" : true,
      "info" : "{\"object\":\"F1_house_B0\",\"property\":\"external_v1N\"}"
    },
    {
      "key" : "opendss/bus/house/voltage/B",
      "type" : "complex",
      "required" : true,
      "info" : "{\"object\":\"F1_house_B0\",\"property\":\"external_v2N\"}"
    }
  ]
}
```

gldopendsshouseshelics.json (House and Command topics only)

```
{
  "coreType" : "zmq",
  "name" : "opendss",
  "period" : 0.000000001,
  "loglevel" : 7,
  "publications" : [
    {
      "global" : false,
      "key" : "bus/house/voltage/A",
      "type" : "complex"
    },
    {
      "global" : false,
      "key" : "bus/house/voltage/B",
      "type" : "complex"
    }
  ],
  "subscriptions" : [
    {
      "key" : "player/command",
      "type" : "string",
      "required" : true
    },
    {
      "key" : "gridlabd/F1_house_B0/load",
      "type" : "complex",
      "required" : true
    }
  ]
}
```

3.3 Running on Windows

There is a corresponding batch file for each bash script. For example, to run the PV example with FNCS:

- Invoke ***test_fncspv.bat***
- Use ***list5570.bat*** to list the FNCS federates that are still running
- Use ***kill5570.bat*** to stop the FNCS federates, if necessary. This only stops one federate at a time, so repeat invocations may be needed.
- The FNCS federates may not for several seconds after the simulation ends. Use ***list5570.bat*** and ***kill5570.bat*** to make sure they've all exited, before starting another FNCS co-simulation

For HELICS, use ***list23404.bat*** and ***kill23404.bat*** to list and stop the HELICS federates, as needed.

3.4 Utility-Scale FNCS Example

A larger FNCS example includes weather and a transactive rationing agent, depicted in Figure 3 [5].

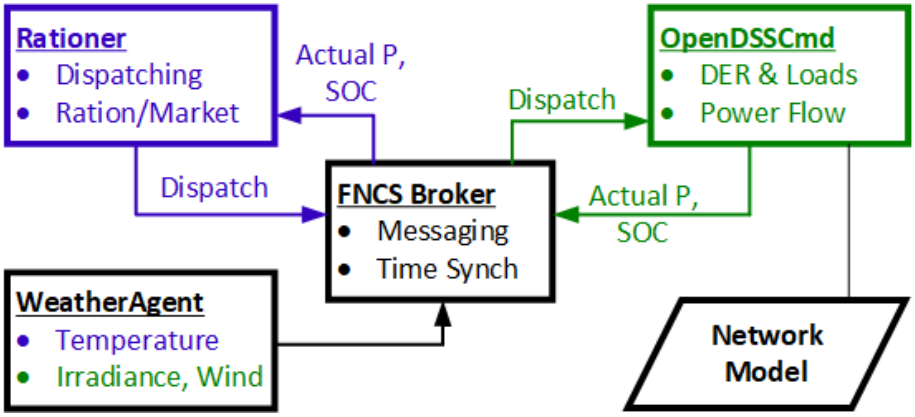

Figure 3: Simulation of transactive rationing with network model and weather [5]

4 Developer Notes for OpenDSSCmd

4.1 Source Code Directories

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/helics.pas – loads and uses the HELICS shared library, i.e., dll, dylib, or so file.
- Source/CMD/opendsscmd.lpr – main source file for the opendsscmd application
- Source/Common/ExportCIMXML.pas – exports the circuit to Common Information Model
- 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 relevant to opendsscmd follows. Please consider any other source directories you may see to be unused or 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/epiktimer – a cross-platform implementation and wrapper of QueryPerformance
- 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/GISCommands – functions that import geographic information system data, which we are not using and depend on Indy support
- 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/Parallel_Lib – supporting multi-core circuit solutions
- 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 and depend on Indy support
- Source/TPerlRegEx – supports the batchedit command
- Source/x64 and Source/x86 – contains build products; PNNL does not use this.

4.2 Notes on FNCS and HELICS Publications

The FNCS and HELICS interfaces are configured as value federates, with publications and subscriptions read from a JSON file (both) or text file (FNCS only).

- Both interfaces should build maps to objects within OpenDSS for each publication, where it makes sense for efficiency. If the network topology changes, the map can be rebuilt automatically because OpenDSS will trigger on topology changes.
- The COM interface implementations in the Source/DLL directory provide some examples of how to map the specific values for publication.
- 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 configured values.
- HELICS endpoint is under development.
- A more efficient binary-level interface to the GridLAB-D houses will be developed, suitable for large numbers of houses.

4.3 Open Issues

1. Some of the newer Version 8 features depend on libraries that aren't available for FPC:
   a. The Indy library for Internet access is not installed with FPC by default. It could be installed separately and become part of the build process.

   b. XsBuiltIns are not available; the date/time function in djson.pas depends on this deprecated feature of Delphi. A new implementation would be needed for FPC.
   c. The memory-mapped files would probably require a new implementation for FPC.
   d. The TlHelp32 library may be supported in a FPC/Lazarus add-on called JEDI Windows API.
2. The regular expressions for the batchedit command, which are implemented in ExecHelper.pas, have become case sensitive. They need to be made case insensitive.
3. Use CMake to build OpenDSSCmd itself. CMake does not come with Pascal support, but a third-party developer has implemented it https://github.com/hedgewars/hw/tree/master/cmake_modules

4.4 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 2.0 and FPC 3.2 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.

4.5 Other OpenDSS Development Branches

The older branch at https://sourceforge.net/p/electricdss/code/HEAD/tree/trunk/Version7/, which had been retained for OpenDSSCmd, is now deprecated. A collection of unofficial OpenDSS extensions, including a C API, Python, Julia and other interfaces is available at:

- https://github.com/dss-extensions/

5 References

[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*, 24-29 July 2011 2011, pp. 1-7, doi: 10.1109/PES.2011.6039829.

[2] S. Ciraci, J. Daily, J. Fuller, A. Fisher, L. Marinovici, and K. Agarwal, "FNCS: A Framework for Power System and Communication Networks Co-Simulation," in *Symposium on Theory of Modeling and Simulation*, San Diego, CA USA, 2014: ACM.

[3] B. Palmintier, D. Krishnamurthy, P. Top, S. Smith, J. Daily, and J. Fuller, "Design of the HELICS high-performance transmission-distribution-communication-market co-simulation framework," in *2017 Workshop on Modeling and Simulation of Cyber-Physical Energy Systems (MSCPES)*, 21-21 April 2017 2017, pp. 1-6, doi: 10.1109/MSCPES.2017.8064542.

[4] R. B. Melton *et al.*, "Leveraging Standards to Create an Open Platform for the Development of Advanced Distribution Applications," *IEEE Access,* vol. 6, pp. 37361-37370, 2018, doi: 10.1109/ACCESS.2018.2851186.

[5] T. E. McDermott, T. Hardy, A. Somani, S. Bender, and M. Moore, "Transactive Energy Rationing in an Islanded Electric Power System," in *IEEE GreenTech*, Denver, 2021, doi: 10.1109/GreenTech48523.2021.00059.