POSSIM: POwer Systems SIMulator Documentation

Version 0.2, 18 June 2013

Contents

1 Introduction

1.1 What is POSSIM?

POSSIM (**PO**wer **S**ystems **SIM**ulator) is a simulation tool aimed at examining the effects of emerging smart grid technologies on the electricity grid at the distribution level. POSSIM provides a time series based simulation platform, takes care of all data input/output and logging, and makes it possible to plug different solutions and power flow tools in and out as required.

POSSIM does *not* conduct any power flow calculations itself, as there are already several established packages that do a very good job of this. Rather, POSSIM provides interfaces to these other packages, allowing smart grid researchers to plug in the power flow package of their choice.

POSSIM was designed to be as flexible and extendable as possible, and allows for a variety of network components and solution algorithms to be implemented and compared, all under the roof of a single, free, common, open-source architecture.

In short, POSSIM *does*:

- input power demand profiles, either for individual components or on an average basis
- allow different models of network components or solution algorithms to be plugged in and out as required
- input data specific to particular network components, such as electric vehicles
- interface with established power flow calculation tools such as MATLAB SimPowerSystems (interfaces to DIgSILENT PowerFactory and OpenDSS are planned)
- log all relevant outputs
- allow researchers with different interests, using different platforms, to share and compare approaches

POSSIM *does not*:

- conduct load flow calculations
- provide a user-interface based interaction, such as drag and drop network building

1.2 Background

POSSIM was originally developed for the electric vehicles research group at the University of Melbourne by Julian de Hoog. In this group, different team members were using different power flow packages to build and examine networks. One goal of the group was to develop and compare smart electric vehicle charging algorithms on each of these various networks. It was decided that a common platform should be developed that would allow different power flow solutions to be plugged in as required, while allowing development of different approaches all within the same codebase. The initial emphasis was on electric vehicles and smart charging, but as the project grew and developed it became clear that further network components (such as distributed generation and storage) would need to be added.

Collaboration, portability, and extendability were goals from the start, and POSSIM has been designed from the bottom up to be as platform independent and flexible as possible. Any further contributions to the code are always welcome.

POSSIM is licensed under the BSD 3-clause license, see Appendix A.

2 Install

To ensure that POSSIM may be built and used across various platforms, some third party libraries and build tools are necessary. Please read the install instructions carefully.

2.1 Boost

POSSIM uses the Boost C++ libraries, primarily to ensure that access to computer clock, file structure, etc works across various platforms. Boost is essential and *must be installed* for POSSIM to function. Boost is available at `http://www.boost.org/`; please ensure that the **filesystem** library is compiled after your boost install.

Full instructions are available in Boost's Getting Started guide; the quick and dirty version is:

- Download the latest version from the boost website, unzip and extract
- In the boost home directory (e.g. C:/boost_1_53_0/) type:

  ```
  $ ./bootstrap.sh --prefix=bin
  ```

  (On Windows, this command would be `bootstrap.bat --prefix=bin`).

  This will prepare your system for compiling all boost libraries (which can take some time); if you only want to install filesystem use instead

  ```
  $ ./bootstrap.sh --prefix=bin --with-libraries=filesystem
  ```

  (again, on Windows use `bootstrap.bat --prefix=bin --with-libraries=filesystem`)

- Compile the libraries:

  ```
  $ ./b2 install
  ```
  (Or on windows, `b2 install`)

2.2 Git

The POSSIM codebase is maintained on GitHub. If you do not know how to use Git, we suggest you briefly familiarise yourself with it via their tutorials.

To fork a copy of POSSIM,

- Create an account with GitHub (`https://github.com/`)
- Find POSSIM in the GitHub interface (simply search for "POSSIM"), and fork the POSSIM repository
- Install Git on your computer (`http://git-scm.com/downloads`)
- Use a terminal and change to the directory where POSSIM is to be installed
- Clone the fork:

  ```
  $ git clone https://github.com/username/POSSIM.git
  ```

2.3 CMake

CMake (`http://www.cmake.org/`) is a cross-platform build tool that generates native makefiles and workspaces to be used in the environment of your choice. POSSIM is supplied with cmake files; using these to build POSSIM is optional, but will be helpful towards getting compiler settings and linker paths correct.

To use cmake to create build files for POSSIM,

- Install the latest version of CMake:
  `http://www.cmake.org/cmake/resources/software.html`
- In POSSIM's home directory, check the CMakeLists.txt file. If you are using MATLAB, the variables should be set as follows:
  - MATLAB_ROOT: Directory where your MATLAB installation sits (e.g. /usr/local/MATLAB/R2012a, or C:/Program Files/MATLAB/R2012a)
  - MATLAB_LIBRARIES: the extension to MATLAB's library files (e.g. /bin/glnx86, or /bin/maci64, or /bin/win32, etc)
  - MATLAB_MEX_LIBRARY ends with .dll on windows, .dylib on mac, .so on linux
  - MATLAB_MX_LIBRARY ends with .dll on windows, .dylib on mac, .so on linux
  - MATLAB_ENG_LIBRARY ends with .dll on windows, .dylib on mac, .so on linux
- In POSSIM's home directory, run:

  ```
  $ cmake .
  ```

  (Note, this can also be done via the CMake user interface that comes with a standard CMake install)
- Ensure that CMake finds boost and MATLAB (if you intend to use MATLAB). If boost was installed in a non-standard location then it may not be found, and you may need to uncomment the lines below in CMakeLists.txt using your system's path to boost:

  ```
  #set(BOOST_ROOT "/path/to/boost/boost_1_53_0"


  #set(BOOST_LIBRARYDIR "/path/to/boost/boost_1_53_0/bin/lib"
  ```

2.4 MATLAB

Currently MATLAB is the only power flow software that POSSIM supports; interfaces to DIgSILENT PowerFactory and OpenDSS are planned. MATLAB allows for network models to be built using the SimPowerSystems toolbox. This toolbox is further used for all power flow calculations conducted as part of POSSIM's simulations.

MATLAB has its own pages dedicated to using MATLAB and C++ together: `http://www.mathworks.com.au/help/matlab/matlab_external/using-matlab-engine.html`.

In short, you need to:

- Ensure MATLAB is on the PATH
- Include MATLAB's header files
- Link to MATLAB's libraries libeng and libmx (and libmex on Mac platforms)

If you use CMake to create your build files, this should be taken care of.

2.5 Installing in Windows XP

Before continuing with the steps in this section, ensure that you have:

- Installed boost (see Section 2.1)

- Compiled boost's filesystem library (Section 2.1)
- Forked/cloned the latest POSSIM codebase from GitHub (Section 2.2)

The following works for getting POSSIM running in Windows XP using Visual Studio 2010:

- It is assumed you have Visual Studio 2010 (or another version) already installed. If not, install this first.
- Open a terminal and navigate to the directory you cloned POSSIM into
- Run cmake to create Visual Studio project files with all relevant compiler and linkage paths included: `$ cmake .` (Remember to include the period). You can also use CMake's user interface option for this if you prefer.
- If there are any errors with running CMake, ensure these are solved before you open the project in Visual Studio. See Section 2.3.
- Next, open Visual Studio, select `File -> Open ... -> Project/Solution` and choose the cmake-generated Visual Studio project file.
- Choose `Build -> Build Solution`, followed by `Debug -> Start Debugging`.
- As part of the build you may see errors like `Cannot find or open the PDB file`; these are related to debugging settings and can be safely ignored (see here).

If you do not have access to a version of Visual Studio, POSSIM may still be run on your Windows machine. However, you will need to ensure that your system has C++ build tools as well as make installed. For example, consider using MinGW for your build tools and make for Windows. Full instructions are not provided as how you set this up could vary considerably from one system to the next.

If you continue to have trouble, check the Install FAQ (Section 2.9).

2.6 Installing in Windows 7

Before continuing with the steps in this section, ensure that you have:

- Installed boost (see Section 2.1)
- Compiled boost's filesystem library (Section 2.1)
- Forked/cloned the latest POSSIM codebase from GitHub (Section 2.2)

The following works for getting POSSIM running in Windows 7 using Visual Studio 2012:

- It is assumed you have Visual Studio 2012 (or another version) already installed. If not, install this first.
- Open a terminal and navigate to the directory you cloned POSSIM into
- Run cmake to create Visual Studio project files with all relevant compiler and linkage paths included: `$ cmake .` (Remember to include the period). You can also use CMake's user interface option for this if you prefer.
- If there are any errors with running CMake, ensure these are solved before you open the project in Visual Studio. See Section 2.3.
- Next, open Visual Studio, select `File -> Open ... -> Project/Solution` and choose the cmake-generated Visual Studio project file.
- Since you are using Windows 7, presumably your version of MATLAB is running in 64-bit. Change the project settings to create a 64-bit target.
- Choose `Build -> Build Solution`, followed by `Debug -> Start Debugging`.
- As part of the build you may see errors like `Cannot find or open the PDB file`; these are related to debugging settings and can be safely ignored (see here).

In Windows 7 you may have some issues involving 32-bit vs 64-bit. If so, ensure that both your boost and matlab versions are 64-bit, and also ensure that you are using a 64-bit build tool (e.g. visual studio) to build a 64-bit target.

If you do not have access to a version of Visual Studio, POSSIM may still be run on your Windows machine. However, you will need to ensure that your system has C++ build tools as well as make installed. For example, consider using MinGW for your build tools and make for Windows. Full instructions are not provided as how you set this up could vary considerably from one system to the next.

If you continue to have trouble, check the Install FAQ (Section 2.9).

2.7 Installing in Linux

Before continuing with the steps in this section, ensure that you have:

- Installed boost (see Section 2.1)
- Compiled boost's filesystem library (Section 2.1)
- Forked/cloned the latest POSSIM codebase from GitHub (Section 2.2)

In linux, CMake is the best option to create your build files. Follow the instructions in Sec. 2.3.

Next, you need to compile using make. Simply run:

```
$ make
```

Following compile, you may receive an error like:

```
make[2]: *** No rule to make target '.../libmx' needed by 'possim'.  Stop.
```

If so, you need to set the runtime library path to include matlab's libraries (as explained on matlab's site). Use the following command:

```
$ export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:matlabroot/bin/glnxa64
```

(of course substituting in the path where your matlab libraries sit)

If you're going to be using POSSIM often, it's probably a good idea to put the above command into your .profile file so that you don't have to add the path each time.

That's it – you can now run POSSIM from a terminal, or load it into your favourite IDE (Eclipse, NetBeans, etc), create a project from makefile, and you're ready to go.

If you continue to have trouble, check the Install FAQ (Section 2.9).

2.8 Installing in Mac

Before continuing with the steps in this section, ensure that you have:

- Installed boost (see Section 2.1)
- Compiled boost's filesystem library (Section 2.1)
- Forked/cloned the latest POSSIM codebase from GitHub (Section 2.2)

In Mac, CMake is the best option to create your build files. Follow the instructions in Sec. 2.3.

Next, you need to compile using make. Simply run:

```
$ make
```

Following compile, you may receive an error like:

```
make[2]: *** No rule to make target '.../libmx' needed by 'possim'.  Stop.
```

If so, you need to set the runtime library path to include matlab's libraries (as explained on matlab's site). Use the following command:

```
$ export DYLD_LIBRARY_PATH=$DYLD_LIBRARY_PATH:matlabroot/bin/maci64
```

(of course substituting in the path where your matlab libraries sit).

If you're going to be using POSSIM often, it's probably a good idea to put the above command into your .profile file so that you don't have to add the path each time.

That's it – you can now run POSSIM from a terminal, or load it into your favourite IDE (Eclipse, NetBeans, etc), create a project from makefile, and you're ready to go.

If you continue to have trouble, check the Install FAQ (Section 2.9).

2.9 Install FAQ

Question: I get the error `Trying to start MATLAB ... matlab: Command not found.`

Answer: Is MATLAB on your PATH? If not make sure you add it. In Windows this can be done via the Control Panel. On Mac or Linux, use:

```
$ export PATH=$PATH:matlabroot/bin/glnxa64
```

(or wherever your matlab binaries are located). You may also want to configure your system to add matlab to the path permanently.

Question: I get the output `- Trying to start MATLAB ... Fail!`

Answer: This means the MATLAB engine isn't opening properly. MATLAB needs a C shell (see here), so make sure you have csh installed. (On linux this is simply done with:
`$ sudo apt-get install csh`).

Question: When starting matlab, I get the message
`.../bin/matlab: 1: .../bin/util/oscheck.sh: /lib/libc.so.6: not found`

Answer: This appears to be a problem with recent installations of matlab and ubuntu (see here); try adding a symbolic link to your c library.

Question: I get the error `Undefined symbols for architecture x86_64:`
`"_mxDestroyArray", referenced from:`
`MatlabInterface:: MatlabInterface() in MatlabInterface.cpp.o.` (Mac)

Answer: Maybe you have an extra libmx.dylib in your path? Try adding the following line to CMakeLists.txt (below the find_library commands):

```
SET(MATLAB_MX_LIBRARY  $MATLAB_LIBRARIES_DIR/libmx.dylib)
```

Then run cmake, make again.

Question: I get the error `dyld: Library not loaded: @rpath/libeng.dylib`. (Mac)

Answer: You need to make sure your runtime library path includes matlab libraries. Try:
`DYLD_LIBRARY_PATH=/Applications/MATLAB_R2012a.app/bin/maci64/:$DYLD_LIBRARY_PATH`
(obviously, replace with your own computer's path to matlab).

Question: I get the error `No rule to make target .../libmx needed by possim.` (Linux)

Answer: You need to make sure your runtime library path includes matlab libraries. Try:

```
$ export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:matlabroot/bin/glnxa64
```

(obviously, replace with your own computer's path to matlab).

Question: I get the message:

```
Undefined symbols for architecture x86_64:
"_mxGetData", referenced from:
MatlabInterface::getVar(...)
```

.

Answer: On some systems, when cmake tries to find matlab's libraries it may find the wrong one (same name, but wrong location, usually `/usr/lib/libmx.dylib`). In this case uncomment the lines in the cmake file and insert your own path:

```
# SET(MATLAB_MEX_LIBRARY "/Application/MATLAB_R2012a.app/bin/maci64/libmex.dylib")
# SET(MATLAB_MX_LIBRARY "/Application/MATLAB_R2012a.app/bin/maci64/libmx.dylib")
# SET(MATLAB_ENG_LIBRARY "/Application/MATLAB_R2012a.app/bin/maci64/libeng.dylib")
```

Question: I get the message:

```
- Trying to start MATLAB ...objc[8975]: Object 0x108044070 of class __NSCFData
autoreleased with no pool in place - just leaking - break on objc_autoreleaseNoPool()
to debug OK.
```

Answer: This is an issue with MATLAB on Mac Lion systems. More info is available here. You can either run the matlab compiler once as superuser (with admin privileges), or simply ignore the warning. (No impact on POSSIM / MATLAB).

3 Getting Started

3.1 Simulator Loop

POSSIM is a single-threaded application. All time-series simulations follow a simple outer loop. As a default, this loop is time-based – however there is no reason it couldn't be extended into e.g. an event-based simulator.

For the application of EV charging, the outer loop is as shown in Fig. 1.

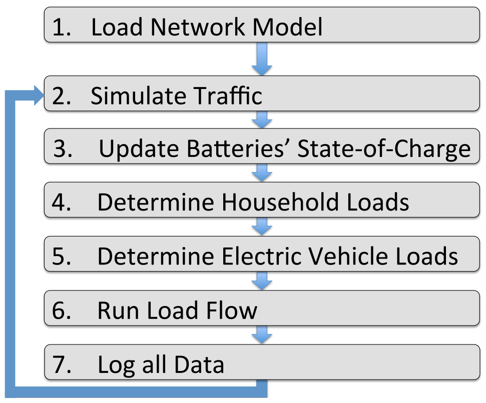
1. Load Network Model
2. Simulate Traffic
3. Update Batteries' State-of-Charge
4. Determine Household Loads
5. Determine Electric Vehicle Loads
6. Run Load Flow
7. Log all Data

Figure 1: Simulator loop

The simulator loop can be found in:
`src/simulator/Simulator.cpp`.

3.2 Configuration

Key global simulation parameters are stored in a single object called `Config`, which maps parameter names to their values. These parameters may be changed via command line options, or in a master xml-based file called `possim_config.xml` in POSSIM's home directory. The Config object may be passed from one object to another, and parameters may be changed on the fly. Call by reference means that any changes to the global configuration proliferate throughout the simulation; this was a conscious design decision.

To access a global configuration parameter, use
`config->getConfigVar("<config_variable_name>")`. For convenience, when the type of the variable is known, type specific functions `config->getBool`, `config->getDouble`, etc may be used.

To set a global configuration parameter, use
`config->setConfigVar("<config_variable_name>", value)`.

For a full list of default parameters and their flags, type `./possim -h` at the command line.

To add your own configuration variable, simply add a line to the `possim_config.xml` file. This variable will now be accessible throughout the code via
`config->getConfigVar("<config_variable_name>")` as above.

3.3 Logging

All essential simulation parameters and outputs are logged at each simulation interval. A separate logging class takes care of all log file creation and updating. For each simulation an individual directory is created that contains all logs. This is located at
`<possim_home>/logs/<date_sim_start>/<time_sim_start>`.

The current version of POSSIM uses MATLAB to generate graphs of particular data of interest; this will likely change in the future.

3.4 DateTime

Since simulations in POSSIM are time-series based, a separate DateTime class was created to keep track of time, compare one moment in time to another, maintain demand value tied to a given time of day, etc. This is an essential component of the simulator, and it is worth familiarising yourself with it briefly before developing or using POSSIM. DateTime objects typically have the format `hh.mm.dd.mm.yyyy`.

4 Network Models and Load Flow

For now it is assumed that MATLAB SimPowerSystems is used to build network models; as extensions to other power flow packages are included in future versions of POSSIM, these will be included here as well.

Actual models of the network may be built with MATLAB's SimPowerSystems package in Simulink, using the convenient drag and drop interface. An example model is presented in Figs. 2, 3, and 4. Generation and transmission is not considered for now, the model starts with the distribution transformer. Phase and neutral voltages and currents are measured individually. Voltages are measured at each individual house. Each segment of line, both feeder and service, is modelled individually. The network uses a matlab-based library, which allows essential network specs to be changed at a single location. However, network components may be altered individually as required, and may be added and removed dynamically (for example in the case of electric vehicles) as the simulation takes place.

All interaction with MATLAB is conducted via the MatlabInterface (in `src/loadflow`). This object starts the matlab engine, and passes any model inputs to the model, while reading any model outputs as required in POSSIM.

To set a value (in this case the impedance of a house) use:

```
loadflow->setDemand(<house_name>, R, L, C)
```

To read an output (in this case the voltage at a house) use:

```
loadflow->getOutputs()
```

(in which case all relevant voltages are stored in individual household objects).

To add for example a vehicle to the matlab model, use:

```
loadflow->addVehicle(<vehicle_name>)
```

(where vehicle_name must match the house it is associated with).

Further interaction with matlab can easily be added to this object.

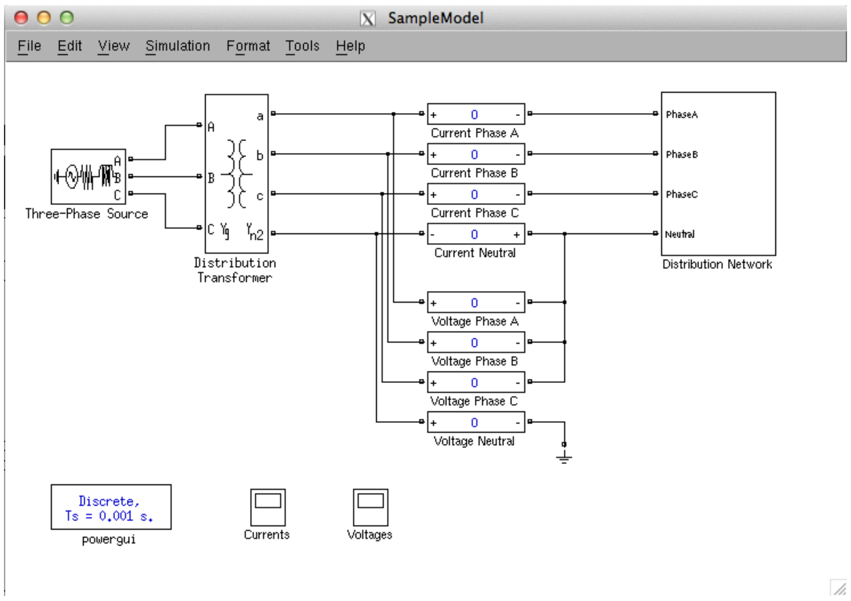

Figure 2: Network model, outer

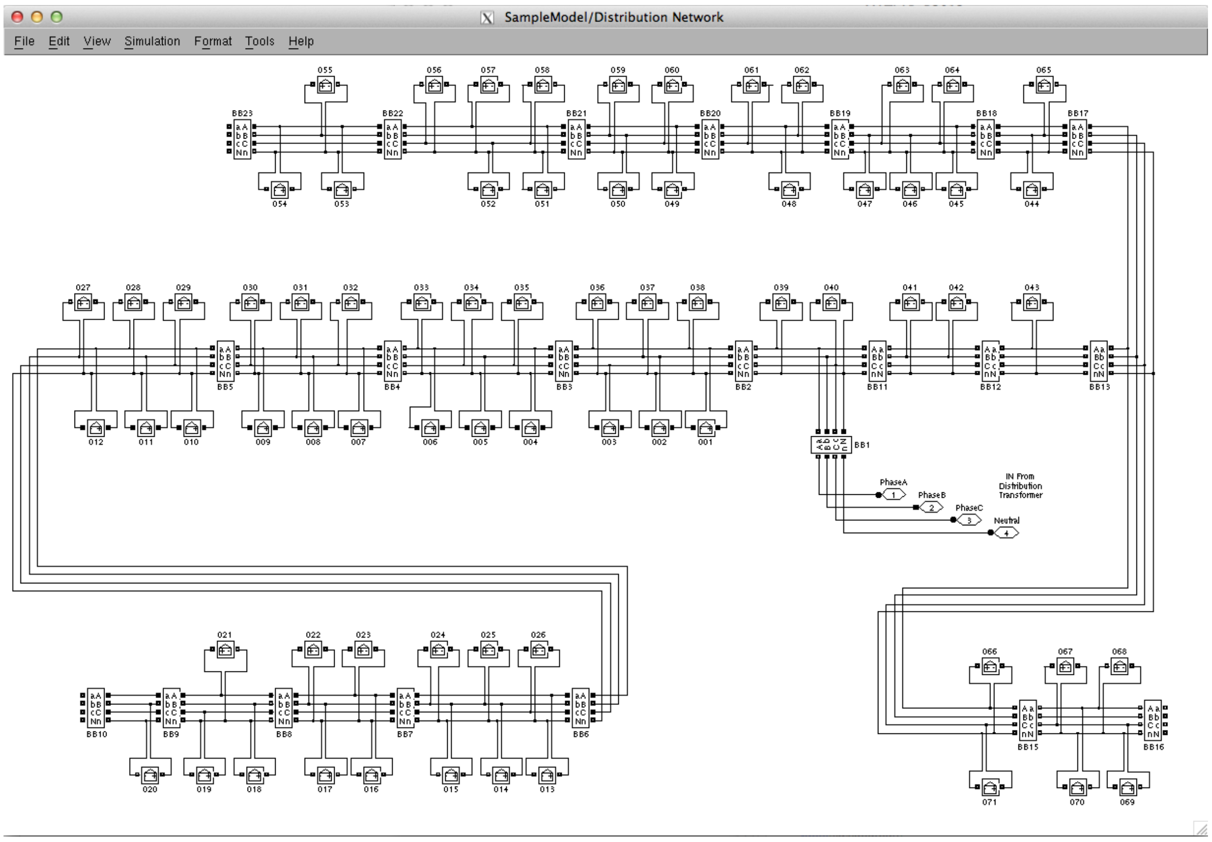
SampleModel/Distribution Network
File Edit View Simulation Format Tools Help
BB1
PhaseA
PhaseB
PhaseC
Neutral
IN From Distribution Transformer

Figure 3: Network model, inner

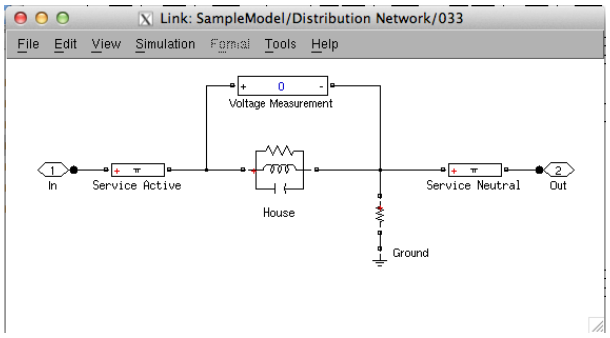
Link: SampleModel/Distribution Network/033
File Edit View Simulation Format Tools Help
Voltage Measurement
In
Service Active
House
Service Neutral
Out
Ground

Figure 4: Network model, individual house

5 Household Demand

The majority of smart grid studies will need to have a sense of daily load fluctuations due to general household demand at the distribution level. How this is done depends on the data available. In some cases precise data for a specific day may be available; in others there may only be average profiles available.

All files specifying household demand data must follow the following format:

```
<datetime1>, <realdemand1>, <reactivedemand1>
<datetime2>, <realdemand2>, <reactivedemand2>
```
etc...

where `datetimeX` has the standard format of the DateTime class.

POSSIM accepts three possible modes for specifying household demand:

1. **Phase-specific**: in this mode, three profiles must be made available, one each for phases A, B, and C. Each house is assigned the demand profile corresponding to the phase it is connected on. This mode is useful for example in situations where phase-specific distribution transformer data has been logged, and average demand profiles are available separately for each phase. To use this mode, set the following config variables:

   ```
   demandmodel          phasespecific
   demanddatadir        path/to/folder/containing/profiles
   ```

   The `demanddatadir` folder must contain three profiles, named `"phaseA", "phaseB",` and `"phaseC"`. File extension doesn't matter.

2. **House-specific**: in this mode, each individual house in the network model is assigned a specific profile. This is possible for example when individual household data has been logged, when the effect of individual household loads wants to be examined, or when individual household profiles have been somehow pre-generated. To use this mode, set the following config variables:

   ```
   demandmodel          housespecific
   demanddatadir        path/to/folder/containing/profiles
   housedemandalloc     path/to/house/allocation/file
   ```

   The `demanddatadir` directory must contain all profiles, and the `housedemandalloc` file must specify which profile is to be assigned to each house in the network. The format of this file is:

   ```
   <housename1>, <profilename1>
   <housename2>, <profilename2>
   ```
   etc...

3. **Random**: in this mode, each house is assigned a random profile. To use this mode, set the following config variables:

   ```
   demandmodel          random
   demanddatadir        path/to/folder/containing/profiles
   ```

   The `demanddatadir` directory must contain all profiles, and each house is assigned one of these randomly. If the directory contains additional files, there may be errors.

6 Vehicle Demand

Coming soon!

7 Electric Vehicle Charging Algorithms

Coming soon!

8 Electric Vehicle Battery Model

Coming soon!

9 Future Plans

POSSIM remains very much an ongoing project, and could be extended in many directions. Among other things, we are keen to:

- improve the documentation
- add a user interface
- add interfaces to DIgSILENT PowerFactory and OpenDSS
- make the install process simpler
- add multi-threading functionality

A License

POSSIM (both the codebase and this documentation) is licensed under the BSD 3-clause license: