Beginner's Guide to GridLAB-D

From GridLAB-D Wiki (UVic shadow)

This guide will provide you an overview of the modules available in GridLAB-D as of Four Corners (Version 2.2). By now you should be familiar with the basic approach to writing models for GridLAB-D, running simulations, and collecting output. If not, please review the Getting Started Using GridLAB-D page first.

Contents

Residential

The residential module provides classes for houses and the appliances one typically finds in them. To load the residential module, use the module directive:

```
module residential;
```

The residential module uses several global variables to give you control over the module's behavior:

```
module residential {
  default_outdoor_temperature 74.0 degF;
  implicit_enduses LIGHTS|PLUGS|OCCUPANCY|DISHWASHER|MICROWAVE|FREEZER|REFRIGERATOR|RANGE|EVCHARGER|WATERHEATER|
}
```

Most of these variables are self-explanatory and details can be found by clicking on the variable name.

House

The house object is the main class of object defined by residential module. These represent typical single-family residential units in North America. They are defined with 110/220 volt distribution panels that allow various end-use loads to be connected (see appliances below).

The principal property of house that determines its behavior is the floor_area. Most other properties are derived by default from the floor area. To change the floor area use

```
object house {
  floor_area 1250 sf;
  // ...
}
```

Other important properties are the thermal integrity and glazing properties, which determine the thermal properties of the house. The change the thermal properties of a house use

```
object house {
  // ...
  thermal_integrity_level ABOVE_NORMAL;
  glazing_treatment LOW_S;
  glazing_layers TWO;
  window_frame INSULATED;
  // ...
}
```

Implicit end-uses

By default, houses have end-uses and appliances implicitly included. This feature is controlled by the implicit_enduses module variable. To disable all implicit definitions of end uses you must do the following

```
module residential {
  implicit_enduses NONE;
}
```

Alternatively, you can include only some enduses:

```
module  residential {
  implicit_enduses LIGHTS|PLUGS|OCCUPANCY;
}
```

Implicit end-uses are driven by load shapes derived from the ELCAP project (http://www.sciencedirect.com /science/article/pii/037877889390026Q) data collected by Pacific Northwest National Laboratory for the Bonneville Power Administration during the late 1980's and early 1990's. These load shapes are defined in schedules that are integrated into the class definition of house.

Explicit end-uses

For certain end-uses it is possible to substitute the implicit end-use with an explicit model. The advantage of doing this lies with the added detail of an explicit model, which includes many the thermal and logic features that are absent from a simply loadshape-based model.

For example, you can omit the WATERHEATER implicit end-use and include the explicit waterheater model:

```
module residential {
  implicit_enduses LIGHTS|PLUGS|REFRIGERATOR|CLOTHESWASHER|DRYER; // omits waterheater
}
object house {
  object waterheater {
  };
}
```

Electric Panels

The house object include an enduse property that is used to collect energy consumption information about the house and the various electrical devices in it. The enduse object is named panel and it contains the following information, among other things

- **complex panel.energy[kVAh]** - the total energy consumed since the last meter reading.
- **complex panel.power[kVA]** - the total power consumption of the load
- **double panel.power_factor** - the power factor of the load
- **complex panel.constant_power[kVA]** - the constant power portion of the total load
- **complex panel.constant_current[kVA]** - the constant current portion of the total load
- **complex panel.constant_admittance[kVA]** - the constant admittance portion of the total load

- **double panel.breaker_amps[A]** - the rated breaker amperage

For example, the following will collect the power consumption for the whole house given the power factor of 0.98:

```
clock {
  timezone EST+5EDT;
  starttime '2000-01-01 00:00:00';
  stoptime '2001-01-01 00:00:00';
}
module residential {
  implicit_enduses LIGHTS|PLUGS|REFRIGERATOR|CLOTHESWASHER|DRYER;
}
module tape;
object house {
  panel.power_factor 0.98;
  object recorder {
    property panel.power;
    file panel_power.csv;
  };
  object waterheater {
  };
}
```

which produces the following output

```
2000-01-01 00:00:00 EST,+5.0488+0.178767j
2000-01-01 00:07:23 EST,+0.5488+0.178767j
2000-01-01 01:00:00 EST,+0.4704+0.153229j
2000-01-01 02:00:00 EST,+0.4256+0.138635j
2000-01-01 04:00:00 EST,+0.4144+0.134987j
2000-01-01 05:00:00 EST,+0.4256+0.138635j
2000-01-01 06:00:00 EST,+0.4816+0.156877j
2000-01-01 07:00:00 EST,+0.5712+0.186063j
2000-01-01 08:00:00 EST,+0.672+0.218898j
2000-01-01 09:00:00 EST,+0.7056+0.229843j
...
```

Climate

An important aspect of GridLAB-D is the ability to include the effect of weather and climate on the performance of systems. To add climate data to a simulation, you must load the climate module and use it to define a climate object:

```
module climate;
object climate {
  tmyfile "CA-Los_angeles.tmy2";
}
```

When the `house.glm` simulation is run using this new weather data, the results reflect the lower and changing outdoor temperatures:

```
2000-01-01 00:00:00 EST,+5.0488+0.178767j
2000-01-01 00:00:01 EST,+11.0972+0.178767j
2000-01-01 00:04:15 EST,+5.0488+0.178767j
2000-01-01 00:07:23 EST,+0.5488+0.178767j
2000-01-01 00:12:28 EST,+6.5972+0.178767j
2000-01-01 00:16:20 EST,+0.5488+0.178767j
2000-01-01 00:26:14 EST,+6.5972+0.178767j
2000-01-01 00:29:54 EST,+0.5488+0.178767j
2000-01-01 00:41:18 EST,+6.5972+0.178767j
```

```
2000-01-01 00:44:51 EST,+0.5488+0.178767j
2000-01-01 00:57:18 EST,+6.5972+0.178767j
2000-01-01 01:00:00 EST,+6.36436+0.153229j
```

Powerflow

To include the electrical system, you need to begin by loading the powerflow module:

```
module powerflow;
```

Like many of the modules, powerflow has a number of module level global variables. The most common are:

```
module powerflow {
  solver_method FBS;
  default_maximum_voltage_error 0.001;
}
```

solver_method allows the user to select the type of solver to use, specifically Newton-Raphson (NR) and Forward Backward Sweep (FBS). FBS is used for radial systems only, and as implemented, is slightly faster on most systems. NR is used for meshed systems and supports a number of additional features, such as reliability and feeder reconfiguration. NR also has a far more robust "error check" functionality, as it is much more susceptible to topological errors. default_maximum_voltage_error determines convergence criteria for the voltage solution in p.u.

To connect a house to the power system, a triplex meter is used:

```
object triplex_meter {
  name Meter1;
  nominal_voltage 120V;
  phases AS;
}
object house {
  parent Meter1;
  // ...
}
```

This provides a connection point between the residential and powerflow modules where the triplex_meter provides a voltage to the house panel and the house provides a description of the current load to the triplex meter. So, every house needs a triplex_meter to connect to the electrical system (although multiple houses can connect to a single triplex_meter). Conceptually, the triplex meter can be thought of as the customer meter, whether AMI, AMR, or electro-mechanical.

Up to this point, this only represents the circuit as seen at a residential house panel, or two 120V circuits and one 240V circuit. Commonly, this is designed as a triplex_line connected to a triplex_node at the secondary side of a center-tap (or split-phase) transformer. The triplex_line setup looks like:

TODO: link these parameters

```
object triplex_line_conductor {
  name "1/0 AA triplex";
  resistance 0.97;
  geometric_mean_radius 0.0111;
}
object triplex_line_configuration {
  name my_line_config;
  conductor_1 "1/0 AA triplex";
  conductor_2 "1/0 AA triplex";
  conductor_N "1/0 AA triplex";
  insulation_thickness 0.08;
  diameter 0.368;
}
object triplex_node {
  name Triplex_Node1;
  nominal_voltage 120V;
  phases AS;
}
object triplex_line {
  from Triplex_Node1;
  to Meter1;
  phases AS;
  length 100 ft;
  configuration my_line_config;
}
object triplex_meter {
  name Meter1;
  nominal_voltage 120V;
  phases AS;
}
object house {
  parent Meter1;
  // ...
}
```

Notice the use of "configuration" or library objects. These are used for all line, transformer, and regulator objects in powerflow, and are very helpful in large files, as a single configuration can be used repeatedly.

This system is still not connected to a three-phase distribution system, as signified by the phases AS. The 'S' indicates that this is a split-phase system (120V/240V), while the 'A' indicates it will be connected to the A-phase of the three-phase system. To connect the house circuit to a 1-, 2-, or 3-phase trunk of a distribution system, a center-tap (or split-phase) transformer is needed. The center-tap transformer and its configuration are defined by:

```
object transformer_configuration {
  name single_phase_transformer;
  connect_type SINGLE_PHASE_CENTER_TAPPED;
  install_type PADMOUNT;
  primary_voltage 7200 V;
  secondary_voltage 120 V;
  power_rating 50.0;
  powerA_rating 50.0;
  resistance 0.011;
  reactance 0.018;
}
object node {
  name Node1;
  nominal_voltage 7200V;
  phases ABCN;
}
object transformer {
  name center_tap_transformer_A;
  phases AS;
  from Node1;
  to Triplex_Node1;
  configuration single_phase_transformer;
}
object triplex_node {
  name Triplex_Node1;
  nominal_voltage 120V;
  phases AS;
}
// ...
```

The ratio of the primary_voltage to secondary_voltage defines the turn ratio of the transformer. The power_rating defines the base power (in kVA) for extracting the per unit reactance and resistance values. The connect_type defines the winding configuration of the transformer. Notice that Node1 contains phases ABCN. This defines all of the phases connected at this node. A two-phase system without a neutral would be defined as AC. In the FBS method, the "from" and "to" nodes (as shown in the transformer) define the topology of the system, with the "from" node being closer to the source, as the system is required to be radial. In the NR method, the order of the nodes is less important because of the meshed system. However, because of this, a swing or slack bus must be defined. This is done by including an additional line:

```
object node {
  bustype SWING;
  // ...
}
```

All other nodes are assumed to be PQ buses, as PV buses are not directly supported (an "effective" PV bus must be created by activating/designing appropriate controls in the generator at that node). The system can built up from here, connecting series of node and link objects until the SWING node is reached, typically at the primary or secondary side of the substation transformer.

This example shows the IEEE 4-node radial test feeder with the load replaced with the previously described split-phase circuit **TODO:** : add this GLM and a diagram. More complex circuits can be designed with additional loads. The Taxonomy of Prototypical Feeders **TODO:** are good examples of standard distribution system models containing static load values. Replacing these static load values with more advanced house (and other) models creates a time-series, powerflow solution which evolves as a function of the interactions between all of the objects. More advanced examples of these highly complex models can be created (or mimicked) using this Matlab(R) script **TODO:** .

TODO: add objects going all the way up to the feeder, including line, configuration, transformers, voltage regulators, fuses, switches, etc.

Market

TODO: implement a simple dynamic-price demand response dispatch

Tape

Important properties of a collector are its group, the property it records, and the file it saves to. A basic example of a collector might look something like this.

```
object collector {
        name transformerLosses;
        group class=transformer;
        property sum(power_losses.real),sum(power_losses.mag);
        file "transformer_losses.csv";
}
```

TODO: implement a histogram

Retrieved from "http://gridlabd.me.uvic.ca/wiki/index.php?title=Beginner%27s_Guide_to_GridLAB-D&oldid=5712"

- This page was last modified on 13 October 2012, at 08:50.