# Getting Started Using GridLAB-D

From GridLAB-D Wiki (UVic shadow)

*Last update: 2015-05-27*
This tutorial is valid for Four Corners (Version 2.2) through Hassayampa (Version 3.0).

---

GridLAB-D is a flexible agent-based simulator that can model the behavior of many objects over time. The simulator looks for changes in objects that affect other objects and keeps track of the evolution of these objects over time. GridLAB-D continues advancing the clock and allows each object to update itself until all the objects report that they are at equilibrium and the clock need not be advanced further. This is very important to understand and is often one of the least understood aspect of GridLAB-D.

GridLAB-D uses modules to define classes of objects. Each class must be defined in a module. Modules can either be static, meaning they are implemented in a dynamic link library (e.g., `.dll` in Windows, `.so` in Linux, `.dylib` on Macs), or they can be dynamic, meaning they are compiled and linked at runtime. Classes define which properties are allowed in objects, and how behaviors are implemented. Objects are instances of classes, so each object can have it's own values for each property while sharing behaviors with other objects of the same class. But during simulations GridLAB-D will keep the objects' properties synchronized with each other as time advances.

## Contents

## GridLAB-D Input Files

GridLAB-D uses two principle kinds of input files. `.glm` files are used primarily to synthesize populations of objects and encode object behavior. `.xml` files are used to represent instances of `.glm` files and exchange data with other software systems. In general `.xml` files will work best with static modules that can be loaded from pre-existing libraries (.so files in Linux), while `.glm` files do not require static modules. `.glm` files can also extend classes of objects implemented by modules, while `.xml` files cannot.

At this point, there is no released tool for editing `.glm` files, although the GldEditor is under development and can be worked on by those with access to source code. Output `.xml` files are viewable using XSL/CSS stylesheets published on the GridLAB-D website.

Please consult the Creating GLM Files page for details on designing GridLAB-D models. For a guide to the modeling using the static module, see the Modeler's Guide.

## Running GridLAB-D

GridLAB-D can be run using the simple command line `gridlabd myfile`. Command line arguments, including options are evaluated and executed in the order in which they appear.

Output is generated to `stdout` and `stderr`. Output redirection is controlled using the `--redirect` command line option.

## GridLAB-D Output Files

GridLAB-D can output result one of two ways. The first and simplest is to use the `-o myfile.xml` command line option to generate an instance of the model at the end of simulation. For more information on `.xml` output, see the XML Data Files page.

The second method of generating output is to use the **tape** module's **recorder** and **collector** objects to generate a time-series of particular values or aggregate values over the entire model. For more information on this see Tape Module Guide page.

# The Primer

If you’re already competent with GridLAB-D and want skip straight to the details of runtime classes or high-performance simulation, then as an advanced user, you should feel free to jump straight the list of modules. However, all beginners and most intermediate users will find there are important concepts and details that you’ll need to get start and quickly become an advanced user.

GridLAB-D is the first (and so far only) environment for simulating the highly integrated modern energy systems that are coming into being all over the world. GridLAB-D is also an open-source system, meaning that the source code, the programming code that makes up GridLAB-D, is freely available to anyone. People all over the world can add to GridLAB-D, fix bugs, make improvements, or suggest optimizations. And they do. GridLAB-D has grown a lot since the prototype implementation (called PDSS, short for Power Distribution System Simulator) created at Pacific Northwest National Laboratory (PNNL) in 2002 [1]. Back then, David Chassin and Ross Guttromson were commissioned under the Laboratory’s Energy Systems Transformation Initiative to look into a) whether such a software system could be built, b) whether it could model how energy systems might evolve over time, and c) how much value would this evolution bring to consumers and utilities. In 2007, after the US Department of Energy's Office of Electricity committed to getting the results of that work more widely available, the open-source model of development and distribution was used to make sure that as many people as possible could both contribute to it and benefit from what it has to offer. Since then, GridLAB-D has grown quickly, mainly because of the hard work and dedication of all the contributors, and of course the early dedication of the GridLAB-D team at PNNL.

Unlike proprietary simulation tools, where the source code is written by a few people and carefully guarded, open-source projects like GridLAB-D exclude no one who is interested in making a contribution if they are competent enough. Many vendors of energy established software tools still scoff at the idea that such a tool can make an impact, but the success of other large-scale open-source projects shows that this approach can work and will work so long as enough support from contributors is available.

Another important advantage of the open-source model is the transparency of the implementation, which is necessary to building confidence in the accuracy of the results that come from using GridLAB-D. While proprietary tools must be carefully validated using test-cases, GridLAB-D has the advantage that validation can begin much sooner by ensuring that the implementation meets industry standards for quality and accuracy.

With Version 1, GridLAB-D revealed the potential for a transformation in how complex energy systems are modeled, and garnered a great deal of interest from potential users around the world. The availability of Version 2 has built on that interest and provides a much more appealing and flexible product with a wider potential range of users. The open-source system works well on both proprietary and open-source operating system and is expected to perform strongly in the utility market. GridLAB-D is set to transform how we model and study modern energy systems.

In this primer we will discuss the following:

- Essential concepts and terminology
- Starting and stopping GridLAB-D
- Creating and running models
- Creating and modifying classes
- Instantiating objects
- Extracting results
- Constructing complex models

## MediaWiki Conventions

Throughout these pages certain conventions are observed to assist readers in gaining access to a maximum amount of information with a minimum of effort.

Blue links

As with all MediaWiki systems, available links are marked in blue. Follow these links to find more information on the topic.

Red links

Unavailable links are marked in red. These indicate that further information is needed, but it has not yet been written. As with many MediaWiki documentation efforts, these pages are always a work in progress. Red links indicate the leading edge of that work.

**Synopsis**

Most pages have a leading synopsis section that briefly describes the usage of the command or concept in question. Often the synopsis will contain blue links to other portions of the page that describe that aspect of the topic in more detail.

`Examples`

Sample code is usually provided with descriptions of features. Often these examples build on previous examples. In such cases, new lines are underlined, ~~unwanted lines are stricken~~, and only the new concepts and terms are marked with blue links.

**See Also**

Most pages have a "See also" section at the bottom that provide you with suggested topics that are related to the page. These are often grouped in topic areas of many related pages.

# Understanding GridLAB-D Basics

GridLAB-D is an agent-based simulation environment specifically designed to model modern energy systems. It's a program capable of tracking the simultaneous states of vast numbers of objects having a wide variety of properties and behaviors, and gathering information about their condition as time progresses in the simulation.

GridLAB-D simulates a system by computing a series of steady states, separated by state transitions. The simulation discovers when those state transitions occur by looking at each object to determine a) what its next proposed steady is, b) whether that state consistent with the proposed steady states of all the other objects in the model, and c) how long that state is expected to last.

The details of how GridLAB-D works are left to a later discussion. But GridLAB-D comes with everything needed to make this happen, and determine what the final result is. All the tools needed for constructing and compiling models, debugging and running them in the simulator, and extracting results are either installed with GridLAB-D or available as free downloads from open-source repositories.

At this point, this is all you need to know to get right into creating a simple model and running a simulation.

## What is a GridLAB-D model?

A GridLAB-D model can be stored in either GLM (GridLAB-D Model) or XML (Extensible Markup Language) files, often with other files around to help define simulation boundary conditions and provide needed supporting data. The names GLM and XML come from the extension used in the file name, .glm and .xml, respectively.

GLM files are not exactly like XML files, though. An XML file is a faithful representation of the exact model being simulated, while a GLM file can provide underlying information such a parametric values and statistical distributions for properties, as well as specifications for synthesizing populations of objects so that large models can be generated from relatively simple descriptions. GLM files also allow you to describe the behavior of objects in the simulation, whereas XML files cannot.

In general we will describe models using the GLM file format and reserve XML for two main functions: 1) viewing output results, and 2) exchanging data with other tools.

Consider a simple example. We need to define a house, each instance of which has a floor_area, drawn from a normal distribution about a mean of 2500 sf with a standard deviation of 250 sf. Compare the GLM object definition

```
module residential;
object house {
	floor_area random.normal(2500,250);
};
```

with the XML code that might be generated after loading it

```
<module>
  <residential/>
<module>
<house>
	<floor_area>2534.2 sf</floor_area>
</house>
```

The GLM file described the property floor_area as having a random value with a normal distribution. If you were to run GridLAB-D multiple times using this object description, then depending on how the random number generator is set up you might see a different value for the floor area each time. But overall the values would have a a mean 2500 sf with a standard deviation of 250 sf. In contrast the XML file simply describes one realization of that object and each time you load the XML you will get the same value for the floor area. The information about randomness is lost in the XML file, but repeatability of the results is gained. Therefore, you should exercise care and use the appropriate type of files depending on the task you need to accomplish. In general, model exchange will require XML files and stochastic (or Monte Carlo) studies will require GLM files.

## Model terminology

In the previous examples, we introduced two very important concepts. The first is notion of an object. Objects describe specific instances of a thing that can respond to and/or act on other objects. A simulation can contain very many objects. It is not uncommon to see a model that contains thousands of objects. Some of the large GridLAB-D models have been built that contain well over 100,000 objects.

However, describing so many objects in detail can be an overwhelmingly difficult challenge. This is where the second notion comes in: a class. Classes are used to group object according to similar properties and behavior. In general we define a class of objects that are similar in structure, but we instantiate objects that exhibit specific properties. For example, according to the Department of Motor Vehicles a person has a name, birth date, height, weight, age, eye and hair color all of which are associated with their driver's license number. This collection of information is the same for all drivers. So a person is a class. John Q Smith, born on May, 1968, who is 6'2", 200 lbs, 32 years old, brown hair and brown eyes and has driver's license number "SMITHJQ325KE" is an instance of that class.

Accordingly, the class definition for the house used above would look something like this

```
// class example
class house {
  double floor_area[sf];
};
```

which tell GridLAB-D that the variable floor_area is a property of all houses and it is stored in units of sf (i.e., square feet). Note that it is not necessary to define the house class this way because modules this for us in a way that ensures everybody uses the same properties and behaviors (see below).

XML files do not really support classes, at least not the way GLM files do. In a GLM files, the class can be defined fully from nothing, but in an XML file, it can only be described in a limited fashion. This is because the definition is not easily transportable from one system to another, but the description is something that other systems often require in order to know how to interpret the model.

For this reason, XML files often use a description file (called an XSD file) to know how to interpret the XML file. The XSD contains all the class information, but XSD files omit many aspects that are specified in GLM files. GLM files provide instructions called directives on what GridLAB-D is supposed to do when it tries to update an object, which is something that is largely irrelevant to (or incompatible with) other tools that might look at a GridLAB-D XML file.

From here on, we will discuss modeling in the context of GLM files, and reserve the special consideration for XML files when they are discussed in later chapters.

## Loading a model

The machine where GridLAB-D is installed is called the host computer. To use GridLAB-D you have several options. First, you can be logged directly on the host computer. Alternatively, you can use a client application on your computer to connect to a server computer that hosts the simulation. Finally, you can run a simple web browser that serves a web-based client application that in turn connects to the simulation either on the web-server or to a third computer that hosts the simulation.

When the GridLAB-D is not directly available from your computer, you will need to connect to a machine where it is available. Often, this is simply a matter of opening a telnet or ssh client to connect to the host computer. Certain client applications may connect using other means, which will certainly be described elsewhere as appropriate. In any case you will need to know the name of the host computer and your login id and password on that computer. These are available from your system administrator, and you should ask him or her for help, if necessary.

Once you've connected to the machine that hosts GridLAB-D, you should check to see what version of GridLAB-D is installed:

```
host% gridlabd --version
GridLAB-D 2.2.0 (Four Corners)
host%
```

The **host%** refers the command shell prompt, which indicates that you are logged in on the host computer where GridLAB-D is installed. It will probably look different on your computer—on Windows machines it's usually **C:\>** and sometimes it's simply **$** on Unix machines.

The bold **gridlabd** text is what you must type in, followed by the Enter or Return key. Sometimes you will have to enter what are called command options to get GridLAB-D to do something different than usual. To get a list of commonly used command options, you can enter the following:

```
host% gridlabd —-help
Syntax: gridlabd [OPTIONS ...] <file> [<file2> ...
(...lots of helpful output...)
host%
```

To get detailed help on a particular topic, you can try the command (NOTE, "--info" is only available in versions 3.0 and newer)

```
host% gridlabd --info house
```

```
host%
```

which will open a web browser window on the page (or list of pages) related to the *topic* indicated. If your topic is multiworded you should use underscores (_) or plus signs (+) instead of spaces:

```
host% gridlabd --info implicit_enduses
host%
```

The command options are used to alter the mode of operation of GridLAB-D. The normal setting for a mode of operation is called the default, and command options are one way to override those defaults. For example, GridLAB-D can be instructed to describing everything it is doing using the verbose mode:

```
host% gridlabd --verbose
  ... load time: 0 sec
  ... starting up batch environment
  ... shutdown complete
  ... elapsed runtime 0 seconds
host%
```

**Tip**

Sometimes the system administrator doesn't properly install GridLAB-D and forgets to add the **gridlabd** command to the system command search path. When this happens and you enter `gridlabd` at the command prompt, you will see some error message to the effect that the command cannot be found or is not recognized. If this occurs, contact the system administrator and ask to have the GridLAB-D installation fixed so that it works from your command line. Alternatively, you will need to provide the full path name to the GridLAB-D executable at the command prompt.

**Note**

Some operating systems are not case sensitive, but GridLAB-D is always case sensitive. Therefore, even though it may be possible to type **GRIDLABD** as well as **gridlabd** at the command line, the command line options may still be case sensitive.

## Creating your first model

The simplest GLM model you can create will contain only one object, will not process any time at all, and will record the state of only one property:

```
// Examples:1a.glm
module residential;
module tape;
object house {
  object recorder {
    property air_temperature;
    file "temperature.csv";
  };
}
```

**Important Note**

Before Hassayampa (Version 3.0) GridLAB-D required a clock block as described in Example 1b below. If you get an error indicated that the local clock is not initialized, you will need to add the following to the beginning of this example:

```
clock {
  starttime '2000-01-01 00:00:00';
  stoptime '2001-01-01 00:00:00';
}
```

However, the result is different because then the simulation will run for the range of time specified instead of simply giving you the state of the system at the default initial time.

This example loads the residential module and the tape module. It then defines a single house with all the default properties. Within that house it defines a single recorder which records the air_temperature property of the house and stores that value in the file *temperature.csv*.

**If you've just installed GridLAB-D**

You may need to set up the system's runtime environment. There are several files that you may have to adjust to accommodate the compilers and runtime libraries that are appropriate to your system. These should have been set automatically, but if you are experiencing trouble with the runtime environment, see the Installation Guide for details on how to make GridLAB-D work with your system.

**If you're a system administrator**

If there is more than one user for GridLAB-D, you will probably have to set up the multiuser runtime support. You can skip this step, but this will require all users to have the same runtime environment, which may be undesirable in certain cases. When there are multiple users, it is quite likely they will need to have the ability to customize their individual GridLAB-D configurations. This is automatically supported by the `gridlabd.conf` runtime file, but several considerations must be observed when operating GridLAB-D this way.

By default, `gridlabd.conf`is located using the `GLPATH` environment variable and the search is always done by first looking in the current folder. If any users have a copy of this file in their search path or in the current directory, then they will not be consulting the host system's standard configuration file. This can result in unpredictable and erratic behavior for different users.

Users may create a file `gridlabd-`*`user`*`.conf`, where *`user`* is the user's login name as specified by the `USER` environment variable. This file will be located and consulted when found. You may opt to create a static one for each user in the GridLAB-D directory that the user cannot edit. Or you can create configuration files for each user in directories they own and can edit. In the latter case you should be certain that their private directories are included in the `GLPATH` environment. On Linux systems, this can be done by adding

```
#setenv GLPATH=${GLPATH}:${HOME}/gridlabd
```

to the `gridlabd.conf` file. In Windows, the correct syntax uses a semicolon instead of a colon:

```
#setenv GLPATH=${GLPATH};${HOME}/gridlabd
```

## Running your model

Using an editor, type this code into the file *house1.glm*. Then run the following command:

```
host% gridlabd house1.glm
host%
```

**Note**

you may some warnings about the clock and voltages, but you can ignore those for now.

Now look at the output file:

```
host% more temperature.csv
1970-01-01 00:02:00 UTC,+69
```

As you can see, the temperature was read once at (actually shortly after) the default initial simulation time.

If you don't get the expected result, make sure your GridLAB-D system is properly installed and make sure you didn't make any typographic errors if you entered the code by hand.

On most systems, you will see extra information about the recording output before the data. Each piece of information is preceded by a '#' character to indicate that it is not part of the recording itself but simply meta-information to describe the recording.

## GLM files

Let's take a quick tour of the **house1.glm** model file to understand how this was result was obtained.

The three commands in this file are called directives. Directives are either simple one-liners like the module directives, or complex multi-line ones like the object directives.

### Loading modules

The first two directives in **house1.glm** instructs GridLAB-D to load modules. Modules define many classes in a single quick and convenient directive. Modules are not like class definitions that you give as a directive because they are not compiled when GridLAB-D is run. Instead, developers package a collection of classes and distribute a dynamic link library (as a `.dll` file in Windows or `.so` file in Linux). The process for building and disseminated a module is described in later chapters.

The directives given in **house1.glm** load the residential and tape modules, which define classes for houses and input/output data processing. In this case, we will be using a house object in the residential module and data recorder object in the tape module.

### Creating new objects

The third and last directive in **house1.glm** is an object definition. Object directives must provide that class type so that GridLAB-D knows what kind of information you'd like to associate with object. An object definition usually includes property values (see below), but sometimes it can also include nested objects.

The third entry includes a nested object. This nested object is the recorder. By nesting the recorder, it is associated with the house. This association is called a parent-child relationship. In this case, the house is the parent object and the recorder is the child object. Parent-child relationships are extremely important in GridLAB-D because they determine a hidden property of objects called the rank. Rank is used to establish things like the order in which time synchronization behaviors are run and whether they can be run simultaneously. This is critical to getting the correct results and enabling fast simulations running on high-performance computers.

### Recording information about an object

The recorder class is one of two classes in the tape module that can gather information about objects as they change over time. Recorders can output data in a variety of formats, and the properties of a recorder object what is gathered, how it is gathered, and where it is delivered to.

The recorder object gathers information from its parent object, in this case the house object. By specifying `property air_temperature;`, the recorder object is instructed to gather only the value of the air_temperature property of the house object. The file property instructs the recorder to deliver the observed values of side to a file called `temperature.csv`. Normally, a recorder object only delivers an observation to the file when the property changes, which is what we want in this case.

The output of a recorder object is normally in the form of comma-separate value file (CSV). This is the format that many programs and applications use to exchange simple tables of column-row oriented data. GridLAB-D will always outputs the date and time of the observation (in simulation time), followed by the observed value(s). Each observation event is placed in a separate row, so that they file is a complete time-series record of the changes in the observed value as the simulation advanced the clock.

## Improving your model

The follow section describes various ways that you can use to extend or improve your models. These include how to use date/time, how to disable and enable objects, referencing other objects, generating populations of objects, embedding context information in objects, using functional properties, performing calculation while loading and running models, and handling units.

### Using date and time

Date and time specifications are used for a lot of things in GridLAB-D. Most importantly they are used to specify the starting and ending time of the simulation:

```
// Examples:1b.glm
clock {
  starttime '2000-01-01 00:00:00 UTC';
  stoptime '2001-01-01 00:00:00 UTC';
}
module residential;
module tape;
object house {
  object recorder {
    property air_temperature;
    file temperature.csv;
  };
}
```

which produces the following output:

```
2000-01-01 00:00:00 UTC,+69
2000-01-01 01:00:00 UTC,+69.9947
2000-01-01 02:00:00 UTC,+70.6269
2000-01-01 03:00:00 UTC,+71.1764
2000-01-01 04:00:00 UTC,+71.6754
2000-01-01 05:00:00 UTC,+72.1189
2000-01-01 06:00:00 UTC,+72.5266
2000-01-01 07:00:00 UTC,+72.927
...
```

Dates and times are usually specified using the ISO (International Standard Organization) standard format, which is "YYYY-MM-DD HH:MM:SS ZZZ". (You can change the format of date/time values using the dateformat global variable.)

If you omit the time zone specification, then the time zone indicated by the TZ environment variable will be used. If you wish to specify the timezone to use in the simulation, use the timezone directive:

```
// Examples:1c.glm
clock {
  timezone EST+5EDT;
  starttime '2000-01-01 00:00:00~~ UTC~~';
  stoptime '2001-01-01 00:00:00~~ UTC~~';
}
module residential;
module tape;
object house {
  object recorder {
    property air_temperature;
    file temperature.csv;
  };
}
```

which results in the following output:

```
2000-01-01 00:00:00 EST,+69
2000-01-01 01:00:00 EST,+69.9947
2000-01-01 02:00:00 EST,+70.6269
2000-01-01 03:00:00 EST,+71.1764
2000-01-01 04:00:00 EST,+71.6754
2000-01-01 05:00:00 EST,+72.1189
2000-01-01 06:00:00 EST,+72.5266
2000-01-01 07:00:00 EST,+72.927
2000-01-01 08:00:00 EST,+73.3376
2000-01-01 09:00:00 EST,+73.7618
```

Time zones are specified in the time zone file `tzinfo.txt` file that is installed with the system in the gridlabd folder. GridLAB-D does not use the operating system's time zone specifications for several reasons. First, some operating systems don't recognize time zone that are historically relevant but no longer used. Second, the simulation often needs to run the simulation in a different time zone than that used by the host computer. Finally, the ability to use alternate time zone rules is essential to understanding the energy use implication of altering the time zone rules, something which policymakers have an interest in and sometimes ask. GridLAB-D can address these questions only if the rules for the simulation are different from the rules on the computer on which the simulation is running.

## Disabling and enabling objects

It is possible to have an object activate at a pre-determined date and time, and have deactivate to exist at some time later. Each object has a pair of built-in properties called in and out that determine when the object enters service and when to go out of service.

In the following example, we modify **house1.glm** to illustrate how the in and out properties function.

```
// Examples:1d.glm
clock {
  timezone EST+5EDT;
  starttime '2000-01-01 00:00:00';
  stoptime '2001-01-01 00:00:00';
}
module residential;
module tape;

```

```
object house {
  object recorder {
    property air_temperature;
    file temperature.csv;
    in '2000-04-01 00:00:00';
    out '2000-04-02 00:00:00';
  };
}
```

Running **house1.glm** give the following result:

```
2000-04-01 00:00:00 EST,+76.0806
2000-04-01 00:05:02 EST,+73.9952
2000-04-01 01:00:00 EST,+75.6225
...
2000-04-01 22:00:00 EST,+75.5875
2000-04-01 23:00:00 EST,+75.8156
2000-04-02 00:00:00 EST,+75.9208
```

The first major difference is that although the simulation started on January 1, 2000 at midnight, the data was collected starting April 1, 2000 as specified by the in property of the recorder. The second difference is that the recorder stopped collecting data after midnight of the April 2, 2000 as specified by the out property of the recorder.

## Referencing other objects

In the above examples, the recorder object is nested within the house object. This is a convenient way to indicate that the recorder depends on the metronome for the property it records. This implied relationship is the parent-child relation.

Another way to specify such a relationship is the give the house a name and reference it from the recorder, as in

```
// Examples:1e.glm
clock {
  timezone EST+5EDT;
  starttime '2000-01-01 00:00:00';
  stoptime '2001-01-01 00:00:00';
}
module residential;
module tape;
object house {
  name MyHouse;
}
object recorder {
  parent MyHouse;
  property air_temperature;
  file temperature.csv;
  in '2000-04-01 00:00:00';
  out '2000-04-02 00:00:00';
}~~;~~
~~}~~
```

In this case, the recorder is explicitly referring to the house as its parent by name. It is using this mechanism that you can link objects together across multiple model files.

The parent relationship has an impact on how behaviors like initialization and time synchronization are performed. So often it’s not appropriate to use it. Some objects reference others objects without using the parent relationship. This is done using an explicitly declared property of object of the object:

```
// Examples:1f.glm  TODO:  add necessary properties so this actually loads ok
```

```
module powerflow;
class link {
  object from;
  object to;
}
object node {
  name Node1;
}
object node {
  name Node2;
}
object link {
  from Node1;
  to Node1;
}
```

## Generating populations

One of the most important capabilities in GridLAB-D is the ability to study large populations of objects as they interact with each other. The challenge for modelers is quickly defining what that population looks like without having to explicitly define each and every object. This is made easy by using a multi-object definition, such as

```
object house:..5 {
  floor_area 2500 sf;
};
```

In this case, GridLAB-D will create 5 identical objects and will all run together.

## Embedding context information

Object populations can create complications for other objects, such as recorders. Because the recorder is nested in the house definition, there are in fact 5 recorders created all of which write to the same file. This will clearly not have the desired outcome. To remedy the problem, context information can be embedded in the name of the output file, such as is shown with the bolded changes:

```
// Examples:1g.glm
clock {
  timezone EST+5EDT;
  starttime '2000-01-01 00:00:00';
  stoptime '2001-01-01 00:00:00';
}
module residential;
module tape;
object house:..5 {
  object recorder {
    property air_temperature;
    file `temperature{id}.csv`;
    in '2000-04-01 00:00:00';
    out '2000-04-02 00:00:00';
  };
}
```

The result is that the a different recorder file is used for each recorder. The recorders' internal id is used for each file name thus allowing a separate file for each. You can enable embedded information from the context of an object by surrounding the value in back-quotes. When a value is defined this way, any part of the value which is in curly-braces is expanded from the context of the object. See Expansion variables for details.

## Functional Values

Sometimes it is necessary to examine the behavior of multiple objects that are not quite the same. When we want to create different objects with different properties we use a functional property, such as

```
object house:..5 {
        floor_area random.uniform(1500,2500);
};
```

Functionals are values that are determined when the object is created. There are a number of supported functionals, most of which are random number generators of different types. In this case, the floor area property will be randomly sampled 5 times from uniform distribution of values between 1500 and 2500. See Functional values for details.

## Performing calculations

Often objects have values that are correlated with each other, rather than independently distributed. For example, by default the air volume of a house ceiling height times the floor area. However, if you want to explicitly calculate a different volume based on floor area you can use an equation, as shown in the following example:

```
object house:..5 {
        floor_area random.uniform(1500,2500);
        air_volume (10*$floor_area);
};
```

Calculated values are always enclosed in parenthese and are an important way to established correlated relationship when large populations of objects are defined. See Property calculations for details.

## Handling units

Every real valued or complex valued property should have units associated with it. All recognized units are defined in the unit file `unitfile.txt`. For example, the house’s floor area property could be defined as

```
class house {
  double floor_area[sf];
}
```

which indicates that the property has a square-foot unit. When a property has a unit associated with it, any use of the property can include a compatible unit, and when it does, the value given will be converted automatically. For example,

```
object house {
  floor_area 125 m^2;
}
```

would convert the value from 125 m^2 to square-feet when loading the model.

Unit conversion is performed automatically, but only when the units are compatible, meaning that they represent the same underlying fundamental quantity, but with different scales. This means that GridLAB-D will automatically convert ft to m, or ft^2 or sf to m^2, but it cannot convert ft to cm^2 because they are fundamentally incompatible.

When a unit is declared in class’s property, but not specified in object’s definition, it is assumed that the value defined uses the declared unit. Therefore `floorarea 2500;` is the same as `floor_area 2500 sf;`. The only difference that if somebody were to come along and change the default unit, then the latter definition would continue to work as expected, while the former would behave differently.

**Note**

Using units is a good way to prevent accidental changes in behavior, to ensure that your models always behave exactly as expected, and to let modelers who use your code later know what you intended.

**Tip**

The unit file does not define every conceivable unit, but the unit parser understands compound units and ISO scales—even though kJ/h is not listed, it can be parsed and converted to any units that is fundamentally compatible with it (e.g., W, kW, Btu/h).

## Providing input

GridLAB-D provides a number of methods for inputting data into the system. **TODO:**

### Players

The first is part of the Tape module, and is used to represent irregular time series data. The Player object provides the ability to update a single object variable at specified times. The values are read from a file formatted like comma-separated value (CSV) files or other sources. The source data must have Timestamp (or time changes) in the first column, and the values to be posted in the second column. The player must be "childed" to the object that you want to play information into.

In this example, the Player is a child of the house object, and will play a stream of values from the CSV file "t_cool" into the property cooling_setpoint of the house object:

```
object house {
   object player {
       property cooling_setpoint;
       file t_cool.csv;
   };
}
```

The actual CSV file is a two-column format with a time value (either an absolute Timestamp or relative time). An absolute timestamp version may look like:

```
2007-01-02 00:00:00, 72.0
2007-01-02 01:00:00, 73.0
2007-01-02 02:00:00, 72.0
2007-01-02 03:00:00, 68.0
2007-01-02 15:00:00, 69.0
2007-01-02 23:00:00, 72.0
```

Note the times can be completely irregular. A second method, using relative time would look like:

```
2007-01-01 23:00:00, 72.0
+1h, 72.0
+1h, 73.0
+1h, 72.0
+1h, 68.0
+12h, 69.0
+8h, 72.0
```

Note that the first time is absolute, but relative timestamps define the same schedule by using differentials in time. This is useful in conjunction with the loop property in Player:

```
object house {
   object player {
       property cooling_setpoint;
       file t_cool_relative.csv;
       loop 31;
   };
}
```

Where the relative times (summing to 24 hours) will be repeated 31 times in a cyclic manner. When the allotted time has expired (i.e., the clock time runs beyond the time designated within the player file), the final value will be used indefinitely.

### Schedules

**TODO:**

### Transforms

**TODO:**

### Loadshapes

**TODO:**

### Links

**TODO:**

## Generating output

GridLAB-D provides a number of ways to generate output from the simulation. The basic concept is that of a data logger. The recorder is hooked up to an object property and makes a copy of the value every so often and writes it to a file for you. What object the recorder looks at and how often it writes is determined by how you set up the recorder. For example

```
object recorder {
  parent MyHouse;
  property air_temperature;
  file "temperature.csv";
};
```

instructs the recorder to observe the `air_temperature` property of the `MyHouse` object and send any changed observations the file `temperature.csv`.

There are additional settings that you can provide to control the behavior of the recorder. To limit the length of the recording, for example:

```
object recorder {
  parent MyHouse;
  property air_temperature;
  file "temperature.csv";
  limit 1000;
};
```

prevent the recorder from making more than 1000 observations.

To change the sampling interval, for example:

```
object recorder {
  parent MyHouse;
  property air_temperature;
  file "temperature.csv";
  limit 1000;
  interval 300;
};
```

additionally instructs the recorder to look at the metronome every 5 minutes.

**Tip**

Setting the interval to 0 causes the recorder to sample the value every internal iteration of the solver, which can help you debug modeling problems when the solvers do not converge.

## Summary

We have seen some of the basic features of GridLAB-D and how you can use it to model a simple system. At this point you are ready to explore the various modules in GridLAB-D and get familiar with the classes and properties they implement as described in the Beginner's Guide to GridLAB-D page.

## References

- [1] RT Guttromson, DP Chassin, SE Widergren, "Residential energy resource models for distribution feeder simulation", IEEE PES GM, 2003, DOI: http://dx.doi.org/10.1109/PES.2003.1267145

Retrieved from "http://gridlabd.me.uvic.ca/wiki/index.php?title=Getting_Started_Using_GridLAB-D&oldid=7636"

- This page was last modified on 27 May 2015, at 09:26.