# Power Flow User Guide

From GridLAB-D Wiki

## Powerflow Overview

The `powerflow` module performs distribution level solver methods to primarily obtain the voltage and current values in a system. Details on the different solver methods and how each object are handled are in the Powerflow guide.

## Inherited Classes

Nearly all objects within the `powerflow` module are derived from two primary objects: `node` and `>link`. Therefore, any properties defined for these two objects are also available to any derived object. For example, a `node` has voltage properties, so a `load` automatically has these properties available as well. Any `powerflow` objects that inherit properties from `node` or `link` will be labeled as such. Furthermore, `node` and `link` contain most relevant default quantities. Derived objects often assume zero value or throw an error if an explicit property is not indicated. Any exceptions to this rule will be indicated in the parameter list of the particular object.

## Contents

## Powerflow Objects

Along with all of the properties inherited from either `node` or `link`, all objects within the `powerflow` module inherit two basic properties. These two properties are the phases of the object and the nominal voltage for that area of the system. These are expressed in the `phases` and `nominal_voltage` parameters of `powerflow` objects.

The `phases` property has a variety of valid inputs. These are:

- `A` - Phase A of a three phase connection
- `B` - Phase B of a three phase connection
- `C` - Phase C of a three phase connection
- `D` - Delta connected phases - this implies ABC, but explicitly specifying them is recommended
- `N` - Neutral phase
- `G` - Ground phase
- `S` - Split phase - this represents residential level wires (2 "hot" and 1 neutral wire)

These different phases can be specified in a variety of ways. Below are some identical examples with a simple `node` object (which is covered in more later in this page).

```
object node {
        phases ABC;
        }
```

```
object node {
	phases "ABC";
	}
```

```
object node {
	phases A|B|C;
	}
```

```
object node {
	phases "A|B|C";
	}
```



The other common property is nominal voltage, which is passed into the objects using the `nominal_voltage` parameter. This parameter is used to ensure connected objects are in the proper region (have the same nominal voltage) and also to specify an initial value for the convergence criteria of the different solver methods. Using the same node example, a 7200 Volt nominal voltage would be expressed as:

```
object node {
	nominal_voltage 7200.0;
	}
```

# Node

The node object is equivalent to a bus of the distribution system. It provides a connection point for `link`-based objects and a point of known voltages on the system. Three phase voltage is typically available in either wye-connected or delta-connected three phase. Wye-connected voltages are contained in `voltage_A`, `voltage_B`, and `voltage_C`. Delta-connected voltages are available in `voltage_AB`, `voltage_BC`, and `voltage_CA`.

## Default Node

A minimalist node could be created with

```
object node {
	name NodeOne;
	phases ABC;
	nominal_voltage 7200.0;
	}
```

which is the same as specifying

```
object node {
	name NodeOne;
	phases ABC;
	nominal_voltage 7200.0;
	voltage_A 7200.0+0d;
	voltage_B 7200.0-120.0d;
	voltage_C 7200.0+120.0d;
	bustype PQ;
	}
```

## Node Parameters

As with all powerflow objects, `phases` and `nominal_voltage` are inherently part of `node`.

| Property Name | Type | Unit | Description |
|---|---|---|---|
| voltage_A | complex | Volts | The voltage on phase `A` of a three-phase system. This may be specified in rectangular (7200.0+0.0j) or polar (7200.0+0.0d) formats. |

| | | | |
|---|---|---|---|
| voltage_B | complex | Volts | The voltage on phase `B` of a three-phase system. This may be specified in rectangular (7200.0+0.0j) or polar (7200.0+0.0d) formats. |
| voltage_C | complex | Volts | The voltage on phase `C` of a three-phase system. This may be specified in rectangular (7200.0+0.0j) or polar (7200.0+0.0d) formats. |
| voltage_AB | complex | Volts | The voltage on phase `AB` of a delta-connected three-phase system. This is a derived quantity and can be read, but it is not recommended you set this value. |
| voltage_BC | complex | Volts | The voltage on phase `BC` of a delta-connected three-phase system. This is a derived quantity and can be read, but it is not recommended you set this value. |
| voltage_CA | complex | Volts | The voltage on phase `CA` of a delta-connected three-phase system. This is a derived quantity and can be read, but it is not recommended you set this value. |
| current_A | complex | Amperes | The current load on phase `A` (wye) or phase `AB` (delta) of the node. This value is typically handled through the `load` object, so modification is not recommended here. |
| current_B | complex | Amperes | The current load on phase `B` (wye) or phase `BC` (delta) of the node. This value is typically handled through the `load` object, so modification is not recommended here. |
| current_C | complex | Amperes | The current load on phase `C` (wye) or phase `CA` (delta) of the node. This value is typically handled through the `load` object, so modification is not recommended here. |
| power_A | complex | Volt-Amperes | The power load on phase `A` (wye) or phase `AB` (delta) of the node. This value is typically handled through the `load` object, so modification is not recommended here. |
| power_B | complex | Volt-Amperes | The power load on phase `B` (wye) or phase `BC` (delta) of the node. This value is typically handled through the `load` object, so modification is not recommended here. |
| power_C | complex | Volt-Amperes | The power load on phase `C` (wye) or phase `CA` (delta) of the node. This value is typically handled through the `load` object, so modification is not recommended here. |
| shunt_A | complex | Siemens (mhos) | The shunt admittance load on phase `A` (wye) or phase `AB` (delta) of the node. This value is typically handled through the `load` object, so modification is not recommended here. |
| shunt_B | complex | Siemens (mhos) | The shunt admittance load on phase `B` (wye) or phase `BC` (delta) of the node. This value is typically handled through the `load` object, so modification is not recommended here. |
| shunt_C | complex | Siemens (mhos) | The shunt admittance load on phase `C` (wye) or phase `CA` (delta) of the node. This value is typically handled through the `load` object, so modification is not recommended here. |

| | | | |
|---|---|---|---|
| bustype | enumeration | N/A | The type of bus the node represents. The different bus distinctions are only valid for the Gauss-Seidel and Newton-Raphson solver methods. The Forward-Back Sweep method (Kersting's method) does not presently incorporate anything other than the `PQ` bus. Valid choices are<br>▪ `PQ` for a constant power bus (default)<br>▪ `PV` for a voltage-controlled (magnitude) bus<br>▪ `SWING` for the infinite bus of a system. |
| maximum_voltage_error | double | Volts | The maximum voltage error for convergence checks in the different powerflow solvers. If left blank, it is derived from the `nominal_voltage` parameter. |
| busflags | enumeration | N/A | A flag to indicate if the current bus has a source or not. Mainly used for `PV` implementations. The only valid entries are `HASSOURCE` to indicate it is a supported bus, or an empty value indicating it is not. Unused at this time. |
| reference_bus | object | N/A | A reference node elsewhere in the system that the `node` will use to obtain frequency information if necessary (unimplemented in GridLAB-D at this point). |
| mean_repair_time | double | seconds | Time after a fault clears for the object to be considered back in service. Mainly used for `reliability` module interactions at this time. |

### Node State of Development

Node is considered a highly developed and validated model.

# Link

The link object is a connection between nodes in a distribution system. The `link` object is not directly useful, but is the basis for objects associated with overhead lines, underground lines, triplex lines, transformers, regulators, switches, and fuses.

### Default Link

A `link` only requires three parameters to be specified by default. Most of the actual functionality comes through other objects.

```
object link {
        name Node1toNode2;
        phases ABC;
        from Node1;
        to Node2;
        }
```

### Link Parameters

Again, as with all powerflow objects, `phases` and `nominal_voltage` are inherently part of `link`. `nominal_voltage` does not need to be specified for `link` objects.

| Property Name | Type | Unit | Description |
|---|---|---|---|

| | | | |
|---|---|---|---|
| from | object | N/A | One connecting end of the `link` object. This will be the name or reference to a `node`-based object elsewhere in the powerflow model. |
| to | object | N/A | The other connecting end of the `link` object. This will be the name or reference to a `node`-based object elsewhere in the powerflow model. |
| power_in | complex | Volt-Amperes | The calculated power flowing into the particular `link` object as a sum of all three phases. |
| power_out | complex | Volt-Amperes | The calculated power flowing out of the particular `link` object as a sum of all three phases. |
| power_losses | complex | Volt-Amperes | The calculated power loss for all three phases between the input and output of the `link`.. |
| power_in_A | complex | Volt-Amperes | The calculated power on phase A flowing into the `link`. |
| power_in_B | complex | Volt-Amperes | The calculated power on phase B flowing into the `link`. |
| power_in_C | complex | Volt-Amperes | The calculated power on phase C flowing into the `link`. |
| power_out_A | complex | Volt-Amperes | The calculated power on phase A flowing out of the `link`. |
| power_out_B | complex | Volt-Amperes | The calculated power on phase B flowing out of the `link`. |
| power_out_C | complex | Volt-Amperes | The calculated power on phase C flowing out of the `link`. |
| power_losses_A | complex | Volt-Amperes | The calculated power loss between the input and output of the `link` on phase A. |
| power_losses_B | complex | Volt-Amperes | The calculated power loss between the input and output of the `link` on phase B. |
| power_losses_C | complex | Volt-Amperes | The calculated power loss between the input and output of the `link` on phase C. |
| status | enumeration | N/A | Status of the line in terms of being `OPEN` or `CLOSED`. This property is mainly used for switches and fuses, but may be used to remove lines from service. This functionality is primarily used in the `FBS` solver mode. |
| current_out_A | complex | Amperes | The calculated current flowing out of the link object on phase `A`. Note: This has not been fully tested for every object. |
| current_out_B | complex | Amperes | The calculated current flowing out of the link object on phase `B`. Note: This has not been fully tested for every object. |
| current_out_C | complex | Amperes | The calculated current flowing out of the link object on phase `C`. Note: This has not been fully tested for every object. |
| current_in_A | complex | Amperes | The calculated current flowing into the link object on phase `A`. Note: This has not been fully tested for every object. |
| current_in_B | complex | Amperes | The calculated current flowing into the link object on phase `B`. Note: This has not been fully tested for every object. |
| current_in_C | complex | Amperes | The calculated current flowing into the link object on phase `C`. Note: This has not been fully tested for |

| | | | |
|---|---|---|---|
| | | | every object. |
| flow_direction | set | N/A | This is a flag telling which direction current is flowing, relative to the to and from designations, on a each phase of a link object.<br>0x000 - UNKNOWN - The flow direction is indeterminate.<br>0x001 - AF - Current is flowing from the from node to the to node on phase A.<br>0x002 - AR - Current is flowing from the to node to the from node on phase A (reverse flow).<br>0x003 - AN - No current is flowing on phase A.<br>0x010 - BF - Current is flowing from the from node to the to node on phase B.<br>0x020 - BR - Current is flowing from the to node to the from node on phase B (reverse flow).<br>0x030 - BN - No current is flowing on phase B.<br>0x100 - CF - Current is flowing from the from node to the to node on phase C.<br>0x200 - CR - Current is flowing from the to node to the from node on phase C (reverse flow).<br>0x300 - CN - No current is flowing on phase C. |
| mean_repair_time | double | seconds | Time after a fault has cleared before the object will be restored to service. Utilized by the reliability module. |

### Link State of Development

Link is considered a highly developed and validated model.

## Line

The line object represents power lines in a distribution system. The line object has two implementations: `overhead_line`, and `underground_line`. Each line must be called appropriately. Information about the particular line type will be contained in other objects called `line_configuration`.

`Line`-based objects inherit properties from the `link` object just covered. Two new properties are also added: `configuration` and `length`.

Typical usage of an overhead line would be

```
object overhead_line {
        name Node1toNode2;
        phases ABC;
        from Node1;
        to Node2;
        length 5280;
        configuration Best_overhead_line_cfg;
        }
```

and the typical usage of the underground line would be

```
object underground_line {
        name Node1toNode2;
        phases ABC;
        from Node1;
        to Node2;
        length 5280;
```

```
        configuration An_underground_line_cfg;
        }
```

### Line Parameters

Along with the inherited `link` properties, `line` objects have:

| Property Name | Type | Unit | Description |
|---|---|---|---|
| length | double | feet | Length of the `line` object. |
| configuration | object | N/A | Name or reference to the particular configuration object that describes the properties of the `line` object. |

## Line configuration

Both `underground_line` and `overhead_line` objects take line configuration information to describe the particular line being implemented, or they can be described in their raw z-matrix values. A typical `line_configuration` object would be implemented as

```
object line_configuration {
        name line_config_A;
        conductor_A overhead_line_conductor_100;
        conductor_B overhead_line_conductor_100;
        conductor_C overhead_line_conductor_100;
        conductor_N overhead_line_conductor_101;
        spacing line_spacing_200;
        }
```

or

```
object line_configuration {
       name line_config_B;
       z11 0.45+1.07j;
       z12 0.15+0.50j;
       z13 0.15+0.38j;
       z21 0.15+0.50j;
       z22 0.46+1.04j;
       z23 0.15+0.42j;
       z31 0.15+0.38j;
       z32 0.15+0.42j;
       z33 0.46+1.06j;
       }
```

### Line configuration properties

| Property Name | Type | Unit | Description |
|---|---|---|---|
| conductor_A | object | N/A | Object describing the conductor of phase A in the overhead or underground line object. (`overhead_line_conductor` or `underground_line_conductor`) |
| conductor_B | object | N/A | Object describing the conductor of phase B in the overhead or underground line object. (`overhead_line_conductor` or `underground_line_conductor`) |
| conductor_C | object | N/A | Object describing the conductor of phase C in the overhead or underground line object. (`overhead_line_conductor` or `underground_line_conductor`) |
| conductor_N | object | N/A | Object describing the conductor of phase N in the overhead or underground line object. (`overhead_line_conductor` or `underground_line_conductor`) |
| | | | `line_spacing` object describing how the conductors are |

| | | | |
|---|---|---|---|
| spacing | object | N/A | physically oriented on the pole or in the bundle. |
| z11-z33 | complex | Ohm/mile | describes the z-matrix directly for either underground or overhead lines instead of using the geometric configurations (This will over-write geometric configurations). |

### Line State of Development

Line is considered a highly developed and validated model in terms of powerflow solutions, however, models may be developed to include more advanced features in the future.

## Line spacing

The line spacing object describe how the individual conductors of a distribution line are arranged underground or on the support pole. A typical implementation of a `line_spacing` object is

```
object line_spacing {
        name line_spacing_200;
        distance_AB 2.5;
        distance_BC 4.5;
        distance_AC 7.0;
        distance_AN 5.656854;
        distance_BN 4.272002;
        distance_CN 5.0;
        }
```

### Line Spacing Parameters

| Property Name | Type | Unit | Description |
|---|---|---|---|
| distance_AB | double | feet | Distance between conductors of phase A and phase B. |
| distance_BC | double | feet | Distance between conductors of phase B and phase C. |
| distance_AC | double | feet | Distance between conductors of phase C and phase A. |
| distance_AN | double | feet | Distance between conductors of phase A and the neutral phase. |
| distance_BN | double | feet | Distance between conductors of phase B and the neutral phase. |
| distance_CN | double | feet | Distance between conductors of phase C and the neutral phase. |
| distance_AE | double | feet | Distance between conductor of phase A and the earth (ground). |
| distance_BE | double | feet | Distance between conductor of phase B and the earth (ground). |
| distance_CE | double | feet | Distance between conductor of phase C and the earth (ground). |
| distance_NE | double | feet | Distance between conductor of the neutral phase (phase N) and the earth (ground). |

### Line Spacing State of Development

Line Spacing is considered a highly developed and validated model in terms of powerflow solutions, however, models may be developed to include more advanced features in the future.

## Overhead Line

Overhead lines are one of three specific line types incorporated into the `powerflow` distribution-level module. The `overhead_line` object will take spacing and conductor parameters and translate those values to appropriate impedance matrices based for the specific overhead transmission line configuration. A typical overhead line would be written as

```
object overhead_line{
       phases "ABCN";
       name 701-802;
       from node_701;
       to load_802;
       length 125960;
       configuration line_config_A;
       }
```

`overhead_line` objects are based around the `link` object and inherit all of its properties. `overhead_line` objects primarily translate the `configuration` options specified into a circuit equivalent, so not further properties than those provided by `link` are required.

### Overhead Line State of Development

Overhead Line is considered a highly developed and validated model in terms of powerflow solutions, however, models may be developed to include more advanced features in the future.

## Overhead Line Conductor

For overhead lines, the `line_configuration` object must specify the overhead line conductor types used in the particular setup. A typical `overhead_line_conductor` would be implemented as

```
object overhead_line_conductor {
       name overhead_line_conductor_100;
       geometric_mean_radius .00446;
       resistance 1.12;
       }
```

### Overhead Line Conductor Parameters

| Property Name | Type | Unit | Description |
|---|---|---|---|
| geometric_mean_radius | double | feet | The GMR of the wire. |
| resistance | double | Ohm/mile | The resistance of the particular conductor, incorporating size and material effects. |
| diameter | double | inches | Diameter of the conductor - used for capacitance calculations. |
| rating.summer.continuous | double | Amperes | The continuous rating for the conductor during summer month usage. **This parameter is unused at this point. Future versions of GridLAB-D may implement this functionality** |
| rating.summer.emergency | double | Amperes | The emergency (short time) rating for the conductor during summer month usage. **This parameter is unused at this point. Future versions of GridLAB-D may implement this functionality** |
| rating.winter.continuous | double | Amperes | The continuous rating for the conductor during winter month usage. **This parameter is unused at this point. Future versions of GridLAB-D may implement this functionality** |
| rating.winter.emergency | double | Amperes | The emergency (short time) rating for the conductor during winter month usage. **This parameter is unused at this point. Future versions of GridLAB-D may implement this functionality** |

### Overhead Line Conductor State of Development

Overhead Line Conductor is considered a highly developed and validated model.

# Underground Line

Underground lines represent burial distribution cables in a powerflow system. In terms of GridLAB-D implementation, they are nearly identical to the `overhead_line` objects. A typical `underground_line` object would be written as

```
object underground_line {
        phases "ABC";
        name 703-727;
        from node_703;
        to load_827;
        length 240;
        configuration line_config_7241;
        }
```

As with `overhead_line` objects, `underground_line` objects inherit all of their properties from the `link` object. The `underground_line` object again serves as a method to choose the appropriate translation algorithms to take the physical parameters of the system and create an equivalent model. As such, it has no new properties either.

## Underground Line State of Development

Underground Line is considered a highly developed and validated model in terms of powerflow solutions, however, models may be developed to include more advanced features in the future.

# Underground Line Conductor

Underground lines often contain concentric shielding layers around the central conductor. As a result, they require more parameters than the `overhead_line_conductor` objects to fully describe them. A typical `underground_line_object` is:

```
object underground_line_conductor {
        name ug_conduct_7210;
        outer_diameter 1.980000;
        conductor_gmr 0.036800;
        conductor_diameter 1.150000;
        conductor_resistance 0.105000;
        neutral_gmr 0.003310;
        neutral_resistance 5.903000;
        neutral_diameter 0.102000;
        neutral_strands 20.000000;
        shield_gmr 0.000000;
        shield_resistance 0.000000;
        }
```

## Underground Line Conductor Parameters

| Property Name | Type | Unit | Description |
|---|---|---|---|
| outer_diameter | double | inches | Diameter of the outside of the cable, including jacketing and shielding. |
| conductor_gmr | double | feet | Geometric mean radius of the conductor at the center of the concentric cable. |
| conductor_diameter | double | inches | Diameter of the conductor at the center of the concentric cable. |
| conductor_resistance | double | Ohm/mile | Resistance of the conductor at the center of the concentric cable. |
| neutral_gmr | double | feet | Geometric mean radius of the concentric neutral of the cable. |

| | | | |
|---|---|---|---|
| neutral_diameter | double | inches | Diameter of the concentric neutral of the cable. |
| neutral_resistance | double | Ohm/mile | Resistance of the concentric neutral of the cable. |
| neutral_strands | integer | N/A | Number of strands composing the concentric neutral conductor. |
| insultation_relative_permitivitty | double | N/A (scalar) | Relative permitivitty of the insulation in a concentric neutral cable - relative to air - used for capacitance calculations. |
| shield_gmr | double | feet | Geometric mean radius of the shielding of the cable. |
| shield_resistance | double | Ohm/mile | Resistance of the cable shielding. |
| rating.summer.continuous | double | Amperes | The continuous rating for the conductor during summer month usage. **This parameter is unused at this point. Future versions of GridLAB-D may implement this functionality** |
| rating.summer.emergency | double | Amperes | The emergency (short time) rating for the conductor during summer month usage. **This parameter is unused at this point. Future versions of GridLAB-D may implement this functionality** |
| rating.winter.continuous | double | Amperes | The continuous rating for the conductor during winter month usage. **This parameter is unused at this point. Future versions of GridLAB-D may implement this functionality** |
| rating.winter.emergency | double | Amperes | The emergency (short time) rating for the conductor during winter month usage. **This parameter is unused at this point. Future versions of GridLAB-D may implement this functionality** |

### Underground Line Conductor State of Development

Underground Line Conductor is considered a highly developed and validated model.

## Triplex line

The third type of line available in the `powerflow` module is the triplex lines. Triplex lines represent the distribution wires coming from the transformer into a typical residential home. That is, they are typically composed of one neutral wire and two "hot" wires. Triplex lines require the phase `S` to be specified as part of the `phases` parameter for proper implementation. A typical triplex line would be implemented in a similar fashion to

```
object triplex_line {
       phases AS;
       length 100 ft;
       from node_4a;
       to node_4;
       configuration triplex_config_AB;
       }
```

As with the `underground_line` and `overhead_line` objects, `triplex_line` objects inherit all of their properties from the `link` object. However, `triplex_lines` use a different configuration structure than the `overhead_line` and `underground_line` objects.

### Triplex Line State of Development

Triplex Line is considered a highly developed and validated model in terms of powerflow solutions, however, models may be developed to include more advanced features in the future.

## Triplex Line Configuration

Triplex lines utilize their own configuration description method. Since the phases are no longer described as `A`, `B`, or `C`, the configuration is relabeled. A typical `triplex_line_configuration` is given as either a geometric configuration

```
object triplex_line_configuration {
                conductor_1 trip_cond_H;
                conductor_2 trip_cond_H;
                conductor_N trip_cond_N;
                insulation_thickness 0.08 in;
                diameter 0.368 in;
                }
```

or by using an explicit z-matrix

```
object triplex_line_configuration {
               z11 1.52+0.61j;
               z12 +0.55+0.44j;
               z21 -0.55-0.44j;
               z22 -1.52-0.61j;
               }
```

Note: The explicit z-matrix version is an under-determined system. Ground and neutral currents will not be calculated, however, voltage and line currents will be correctly calculated.

### Triplex Line Configuration Parameters

| Property Name | Type | Unit | Description |
|---|---|---|---|
| conductor_1 | object | N/A | `triplex_conductor` object that represents the physical wire of phase 1. |
| conductor_2 | object | N/A | `triplex_conductor` object that represents the physical wire of phase 2. |
| conductor_N | object | N/A | `triplex_conductor` object that represents the physical wire of the neutral phase. |
| insulation_thickness | double | inches | Thickness of the insulation around the phase 1 and phase 2 conductors |
| diameter | double | inches | Diameter of the conductor |
| spacing | object | N/A | `line_spacing` object with information on the physical layout of the conductors. **This parameter is unused at this point. Future versions of GridLAB-D may implement this functionality** |
| z11-z22 | complex | Ohm/mile | Describes the z-matrix explicitly as opposed to using geometric configurations. Using this will over-write the geometric configurations. |

### Triplex Line Configuration State of Development

Triplex Line Configuration is considered a highly developed and validated model.

## Triplex Conductor

As with the `underground_line` and `overhead_line` objects, `triplex_line` objects have their own conductor objects. This object describes the physical characteristics of the actual wire used in the triple line bundle. A typical implementation would be:

```
object triplex_line_conductor {
       name trip_cond_1;
       resistance 0.97;
       geometric_mean_radius 0.0111;
       }
```

### Triplex Conductor Parameters

| Property Name | Type | Unit | Description |
|---|---|---|---|
| resistance | double | Ohm/mile | Resistance of the conductor. |
| geometric_mean_radius | double | feet | GMR of conductor. |

### Triplex Conductor State of Development

Triplex Conductor is considered a highly developed and validated model.

# Transformer

Transformers provide a means to change the voltage from one node to another in the distribution system. Similar to the different `line` objects, a `transformer` object requires a configuration object to specify the details of the implementation. A typical transform implementation is

```
object transformer {
       name xfrmr_709_775;
       phases "ABC";
       from node_709;
       to node_775;
       configuration xfrmr_config_400;
       }
```

### Transformer Thermal/Aging Model

A newly added feature to transformers in 3.0 is a thermal/aging model. New parameters are placed within the transformer and and transformer_configuration in order to use this new feature. An implementation of the thermal model within transformer is

```
object transformer {
       name xfrmr_709_775;
       phases "ABC";
       from node_709;
       to node_775;
       configuration xfrmr_config_400;
       use_thermal_model TRUE;
       climate Seattle;
       aging_granularity 300;
       percent_loss_of_life 20;
       }
```

### Transformer Parameters

Transformers are derived from the `link` class and inherit all of its properties. The only unique property a `transformer` object contains is

| Property Name | Type | Unit | Description |
|---|---|---|---|
| aging_constant | double | Kelvin | Experimental value used in determining the transformer insulation breaking point. The default is 15000 K. |

| | | | |
|---|---|---|---|
| aging_granularity | double | sec | The maximum time step between transformer age and internal temperature updates. The default is 300 seconds. |
| ambient_temperature | double | Celsius | Output of the ambient temperature around the transformer. The default is 22.8 C. |
| climate | object | N/A | `climate` object that determines the outside ambient temperature around the transformer. |
| configuration | object | N/A | `transformer_configuration` object that describes the specific transformer implementation. |
| percent_loss_of_life | double | % | The percent amount of transformer's operational life used. If no initial value is given then the transformer is considered brand new. |
| top_oil_hot_spot_temperature | double | Celsius | The hot spot temperature of the top-oil in the transformer. Default initial value is the ambient temperature. |
| use_thermal_model | boolean | N/A | Flag used to enable use of the thermal/aging model. Default is FALSE. |
| winding_hot_spot_temperature | double | Celsius | The hot spot temperature of the transformer windings. Default initial value is the ambient temperature. |

### Transformer State of Development

Transformer is considered a highly developed and validated model in terms of powerflow solutions, however, models may be developed to include more advanced features in the future. Additionally, future work may include additional transformer configurations.

## Transformer Configuration

The `transformer_configuration` object describes the details of a particular transformer implementation. It includes information like the power rating, connection type, and nominal voltage on each side. A typical delta-delta transformer configuration would be implemented as

```
object transformer_configuration {
        name xfrm_config_400;
        connect_type DELTA_DELTA;
        install_type PADMOUNT;
        power_rating 500;
        primary_voltage 4800;
        secondary_voltage 480;
        resistance 0.09;
        reactance 1.81;
        }
```

### Transformer Thermal/Aging Model

A new feature added to transformers in 3.0 is the thermal/aging model. New parameters are added to `transformer_configuration` for this new feature. This model only works with a SINGLE_PHASE_CENTER_TAPPED `transformer`. A typical implementation is

```
object transformer_configuration {
        name xfrm_config_400;
        connect_type SINGLE_PHASE_CENTER_TAPPED;
        install_type PADMOUNT;
        power_rating 500;
        primary_voltage 4800;
        secondary_voltage 480;
        full_load_loss 0.006;
```

```
    no_load_loss 0.003;
    reactance_resistance_ratio 10;
    core_coil_weight 50;
    tank_fittings_weight 60;
    oil_volume 5;
    rated_winding_hot_spot_rise 80;
    rated_top_oil_rise 30;
    rated_winding_time_constant 0.5;
    installed_insulation_life 175200;
    coolant_type MINERAL_OIL;
    cooling_type OA;
    }
```

## Transformer Configuration Parameters

| Property Name | Type | Unit | Description |
|---|---|---|---|
| connect_type | enumeration | N/A | Describes the electrical connection between the high and low side of the transformer. These may be referenced by keyword or number<br><br>0 - `UNKNOWN` - An unknown transformer that will throw an error when used.<br>1 - `WYE_WYE` - A wye to wye connected transformer.<br>2 - `DELTA_DELTA` - A delta to delta connected transformer.<br>3 - `DELTA_GWYE` - A delta to grounded-wye connected transformer.<br>4 - `SINGLE_PHASE` - A single leg of a wye to wye connected transformer.<br>5 - `SINGLE_PHASE_CENTER_TAPPED` - A single-phase, center-tapped transformer or split-phase transformer. Used to connect three-phase distribution to triplex-distribution. |
| install_type | enumeration | N/A | Describes the type of transformer the object represents. Used for informational purposes only. Valid types may be referenced by keyword or number<br><br>0 - `UNKNOWN` - No information on the transformer physical type.<br>1 - `POLETOP` - A pole-mounted transformer.<br>2 - `PADMOUNT` - A pad, or ground level transformer.<br>3 - `VAULT` - An enclosed transformer "building," either underground or above ground. |
| primary_voltage | double | Volts | Nominal voltage of the primary winding side of the transformer. |
| secondary_voltage | double | Volts | Nominal voltage of the secondary winding side of the transformer. |
| power_rating | double | kilo-Volt Amperes | Nominal power rating of the entire transformer. |

| | | | |
|---|---|---|---|
| powerA_rating | double | kilo-Volt Amperes | Nominal power rating of windings associated with phase `A` if wye-connected or `AB` if delta-connected. |
| powerB_rating | double | kilo-Volt Amperes | Nominal power rating of windings associated with phase `B` if wye-connected or `BC` if delta-connected. |
| powerC_rating | double | kilo-Volt Amperes | Nominal power rating of windings associated with phase `C` if wye-connected or `CA` if delta-connected. |
| resistance | double | per-unit Ohm | De-referenced characteristic resistance of the transformer |
| reactance | double | per-unit Ohm | De-referenced characteristic reactance of the transformer |
| impedance | complex | per-unit Ohm | De-referenced characteristic impedance of the transformer. Note that `resistance` and `reactance` above directly write the real and complex portions of this parameter, so only `resistance` and `reactance` or just `impedance` need to be specified. |
| shunt_impedance | complex | per-unit Ohm | Some transformer models support a shunt impedance value to represent no load losses (only wye-wye and center-tap transformers use this value at this time). |
| impedance1 | complex | per-unit Ohm | De-referenced characteristic impedance of the transformer. Currently only used with center-tap transformers. Defaults to zero; not required for operation. Allows user to reflect impedance values on both the primary and secondary side of transformer (primary is specified by `impedance`, secondary by `impedance1` and `impedance2`). Phase 1 equals phase 1 of split-phasing. |
| impedance2 | complex | per-unit Ohm | De-referenced characteristic impedance of the transformer. Currently only used with center-tap transformers. Defaults to zero; not required for operation. Allows user to reflect impedance values on both the primary and secondary side of transformer (primary is specified by `impedance`, secondary by `impedance1` and `impedance2`). Phase 2 equals phase 2 of split-phasing. |
| full_load_loss | double | per-unit Ohm | This is the losses of the transformer when at rated load. |
| no_load_loss | double | per-unit Ohm | The losses through the transformer when there is no load. |
| reactance_resistance_ratio | double | N/A | The ratio the reactance to the resistance for both shunt and series impedances of the transformer. default is 10. |
| tank_fittings_weight | double | Pounds | The weight of the transformer's tank and fittings assembly. |
| oil_volume | double | Gallons | The amount of oil contained within the transformer. |

| | | | |
|---|---|---|---|
| core_coil_weight | double | Pounds | The weight of the transformer's core and coil assembly. |
| rated_winding_hot_spot_rise | double | Celsius | The winding hot spot temperature rise over ambient at rated transformer load. Default is 80 degrees C. |
| rated_top_oil_rise | double | Celsius | The top oil temperature rise over ambient at rated transformer load. |
| rated_winding_time_constant | double | Hours | The winding's time constant. |
| installed_insulation_life | double | Hours | The transformer's operational time span. |
| coolant_type | enumeration | N/A | The type of coolant used in the transformer. Valid types may be referenced by keyword or number<br>0 - `UNKNOWN` - An unknown and unknown coolant type that will throw an error when used.<br>1 - `MINERAL_OIL` - A transformer immersed in mineral oil.<br>2 - `DRY` - A transformer with air as its coolant. This type is not handled yet. |
| cooling_type | enumeration | N/A | The type of cooling used in the transformer. Valid types may be referenced by keyword or number<br>0 - `UNKNOWN` - An unknown and unknown cooling type that will throw an error when used.<br>1 - `OA` - A liquid immersed self cooled transformer.<br>2 - `FA` - A forced air cooled liquid immersed transformer.<br>3 - `NDFOA` - A transformer with non-direction forced oil and air flow.<br>4 - `NDFOW` - A transformer with non-direction forced oil and water flow.<br>5 - `DFOA` - A transformer with direction forced oil and air flow.<br>6 - `DFOW` - A transformer with direction forced oil and water flow. |

### Transformer Configuration State of Development

Transformer Configuration is considered a highly developed and validated model in terms of powerflow solutions, however, models may be developed to include more advanced features in the future, and additional models may be included as needed.

## Load

Load objects present a method for taking power out of the system in controlled, known amounts. While implemented as a constant load, `player` objects can be used to vary the load with time. `load` objects provide a means to implement constant current, constant power, and constant impedance losses or generation into the system. The convention is a load is a positive quantity, so generation would need to be represented as a negative number.

Loads can be a mixture of the constant current, constant impedance, and constant power types. A typical, mixed load would be implemented as

```
object load {
        phases "ABCD";
        name 841;
        constant_current_C -0.586139+9.765222j;
        constant_impedance_B 221.915014+104.430595j;
        constant_power_A 42000.000000+21000.000000j;
        nominal_voltage 4800;
        }
```

## Load Parameters

`load` objects are derived from the `node` objects, so all of the same properties apply.

| Property Name | Type | Unit | Description |
|---|---|---|---|
| load_class | enumeration | N/A | Describes the type of load the object represents. Used for informational purposes only. Valid types may be referenced by keyword or number<br>0 - `U` - Unknown load type<br>1 - `R` - Residential load<br>2 - `C` - Commercial load<br>3 - `I` - Industrial load<br>4 - `A` - Agricultural load |
| measured_voltage_A | complex | Volts | A point to measure the voltage on phase `A` of the load. Note that this value will be from the previous powerflow iteration, so may not be up to date. For best results, read `voltage_A` directly. |
| measured_voltage_B | complex | Volts | A point to measure the voltage on phase `B` of the load. Note that this value will be from the previous powerflow iteration, so may not be up to date. For best results, read `voltage_B` directly. |
| measured_voltage_C | complex | Volts | A point to measure the voltage on phase `C` of the load. Note that this value will be from the previous powerflow iteration, so may not be up to date. For best results, read `voltage_C` directly. |
| measured_voltage_AB | complex | Volts | A point to measure the voltage on delta-phase `AB` of the load. Note that this value will be from the previous powerflow iteration, so may not be up to date. For best results, read `voltage_AB` directly. |
| measured_voltage_BC | complex | Volts | A point to measure the voltage on delta-phase `BC` of the load. Note that this value will be from the previous powerflow iteration, so may not be up to date. For best results, read `voltage_BC` directly. |
| measured_voltage_CA | complex | Volts | A point to measure the voltage on delta-phase `CA` of the load. Note that this value will be from the previous powerflow iteration, so may not be up to date. For best results, read `voltage_CA` directly. |

*The following terms define the load in classical format as constant power, current, and impedance loads on each phase.*

| | | | |
|---|---|---|---|
| constant_power_A | complex | Volt-Amperes | The constant power quantity of the load attached to phase `A` in a wye connection and phase `AB` in a delta connection. |
| constant_power_B | complex | Volt-Amperes | The constant power quantity of the load attached to phase `B` in a wye connection and phase `BC` in a delta connection. |
| constant_power_C | complex | Volt-Amperes | The constant power quantity of the load attached to phase `C` in a wye connection and phase `CA` in a delta connection. |
| constant_current_A | complex | Amperes | The constant current quantity of the load attached to phase `A` in a wye connection and phase `AB` in a delta connection. |
| constant_current_B | complex | Amperes | The constant current quantity of the load attached to phase `B` in a wye connection and phase `BC` in a delta connection. |
| constant_current_C | complex | Amperes | The constant current quantity of the load attached to phase `C` in a wye connection and phase `CA` in a delta connection. |
| constant_impedance_A | complex | Ohms | The constant impedance quantity of the load attached to phase `A` in a wye connection and phase `AB` in a delta connection. |
| constant_impedance_B | complex | Ohms | The constant impedance quantity of the load attached to phase `B` in a wye connection and phase `BC` in a delta connection. |
| constant_impedance_C | complex | Ohms | The constant impedance quantity of the load attached to phase `C` in a wye connection and phase `CA` in a delta connection. |

*The following terms are NOT used in conjunction with the previous set.*

*These terms are used in the manner of a ZIPload - base power (in VA) is specified on a by-phase basis, then power factor and ZIP fractions for each are specified. All phase rotations are handled internally.*

| | | | |
|---|---|---|---|
| base_power_A | double | VA | in similar format as ZIPload this represents the nominal power on phase A before applying ZIP fractions |
| base_power_B | double | VA | in similar format as ZIPload this represents the nominal power on phase B before applying ZIP fractions |
| base_power_C | double | VA | in similar format as ZIPload this represents the nominal power on phase C before applying ZIP fractions |
| power_pf_A | double | pu | in similar format as ZIPload this is the power factor of the phase A constant power portion of load |
| current_pf_A | double | pu | in similar format as ZIPload this is the power factor of the phase A constant current portion of load |
| impedance_pf_A | double | pu | in similar format as ZIPload this is the power factor of the phase A constant impedance portion of load |
| power_pf_B | double | pu | in similar format as ZIPload this is the power factor of the phase B constant power portion of load |
| | | | in similar format as ZIPload this is the power |

| | | | |
|---|---|---|---|
| current_pf_B | double | pu | factor of the phase B constant current portion of load |
| impedance_pf_B | double | pu | in similar format as ZIPload this is the power factor of the phase B constant impedance portion of load |
| power_pf_C | double | pu | in similar format as ZIPload this is the power factor of the phase C constant power portion of load |
| current_pf_C | double | pu | in similar format as ZIPload this is the power factor of the phase C constant current portion of load |
| impedance_pf_C | double | pu | in similar format as ZIPload this is the power factor of the phase C constant impedance portion of load |
| power_fraction_A | double | pu | this is the constant power fraction of base power on phase A |
| current_fraction_A | double | pu | this is the constant current fraction of base power on phase A |
| impedance_fraction_A | double | pu | this is the constant impedance fraction of base power on phase A |
| power_fraction_B | double | pu | this is the constant power fraction of base power on phase B |
| current_fraction_B | double | pu | this is the constant current fraction of base power on phase B |
| impedance_fraction_B | double | pu | this is the constant impedance fraction of base power on phase B |
| power_fraction_C | double | pu | this is the constant power fraction of base power on phase C |
| current_fraction_C | double | pu | this is the constant current fraction of base power on phase C |
| impedance_fraction_C | double | pu | this is the constant impedance fraction of base power on phase C |

### Load State of Development

Load is considered a well developed and validated model, with a number of features. Additional features may be included as needed.

## Meter

Meters provide a measurement point for power and energy on the system at a specific point. Coupled with a `recorder` or `collector`, the `meter` object provides a method determine how much power and energy have been used by downstream connections, as well as how much current is flowing through the meter object at the present time. A typical implementation would be

```
object meter {
        name Mtr1;
        phases ABC;
        nominal_voltage 4800.0;
        }
```

### Meter Parameters

A `meter` object is a derivation of the `node` object and thus inherits all of its parameters. Most `meter` parameters are meant to be read-only, but can be set if the need arises.

| Property Name | Type | Unit | Description |
|---|---|---|---|
| measured_real_energy | double | Watt-hours | Measurement of the real energy (accumulation of the real power) that has flowed through the `meter` since it was reset. |
| measured_reactive_energy | double | VA-hours | Measurement of the reactive energy (accumulation of the reactive power) that has flowed through the `meter` since it was reset. |
| measured_power | complex | Volt-Amperes | Measurement of the complex power flowing through the meter at that instant in time. |
| measured_power_A | complex | Volt-Amperes | Measurement of the complex power flowing through the meter at that instant in time on phase `A`. |
| measured_power_B | complex | Volt-Amperes | Measurement of the complex power flowing through the meter at that instant in time on phase `B`. |
| measured_power_C | complex | Volt-Amperes | Measurement of the complex power flowing through the meter at that instant in time on phase `C`. |
| measured_demand | double | Watts | Measurement of the peak power demand of downstream objects. |
| measured_real_power | double | Watts | Measurement of the real portion of the power flowing through the meter at that instant in time. |
| measured_reactive_power | double | Volt-Amperes reactive | Measurement of the reactive portion of the power flowing through the meter at that instant in time. |
| measured_voltage_A | complex | Volts | Measurement of the voltage on phase `A` of the meter. May or may not be as up to date as reading `voltage_A` directly. |
| measured_voltage_B | complex | Volts | Measurement of the voltage on phase `B` of the meter. May or may not be as up to date as reading `voltage_B` directly. |
| measured_voltage_C | complex | Volts | Measurement of the voltage on phase `C` of the meter. May or may not be as up to date as reading `voltage_C` directly. |
| measured_current_A | complex | Amperes | Measurement of the current on phase `A` of the meter at that instant in time. |
| measured_current_B | complex | Amperes | Measurement of the current on phase `B` of the meter at that instant in time. |
| measured_current_C | complex | Amperes | Measurement of the current on phase `C` of the meter at that instant in time. |
| bill_day | int32 | N/A | Sets the date of the month at which the final monthly bill is calculated (at midnight). Maximum value is 28. |
| price | double | $/kWh | Determines the instantaneous market price of energy. Where the price comes from depends upon the `bill_mode`. |
| monthly_fee | double | $ | This is a recurrent monthly service charge that is added into the bill on the first day of the billing cycle (no pro-rating). |

| | | | |
|---|---|---|---|
| monthly_bill | double | $ | This is the running monthly bill at the particular meter as a function of price and the amount of energy used in that month. |
| previous_monthly_bill | double | $ | This stores the total bill from the previous month after the bill has been processed on the `bill_day`. |
| monthly_energy | double | kWh | The rolling amount of energy consumed during the current month at that meter. Used to calculate `monthly_bill`. |
| previous_monthly_energy | double | kWh | Stores the previous month's total energy consumption. |
| bill_mode | enumeration | N/A | Describes the method in which the meter receives its price signal.<br>0 - `NONE` - Billing is not used (default).<br>1 - `UNIFORM` - A static price is used through variable `price`, however, this may change over time using a player or schedule.<br>2 - `TIERED` - Tiered pricing plan where the price changes as a function of the amount of energy used in the month. See `tier_price` and `tier_energy`.<br>3 - `HOURLY` - This is used in conjunction with an `auction` or `stubauction` object. Receives its price directly from a market signal, but only updates on an hourly basis. Used in conjunction with `power_market`. |
| power_market | object | N/A | When using `bill_mode HOURLY`, this points the meter to the object where it will receive a price signal. |
| first_tier_price | double | $/kWh | When using `bill_mode TIERED`, this determines the energy price after energy increases above `first_tier_energy`, but below `second_tier_energy`. If `second_tier_energy` is not defined, then this price will be used to infinity. While energy is below `first_tier_energy`, `price` is used to calculate the `monthly_bill`. |
| second_tier_price | double | $/kWh | When using `bill_mode TIERED`, this determines the energy price after energy increases above `second_tier_energy`, but below `third_tier_energy`. If `third_tier_energy` is not defined, then this price will be used to infinity. |
| third_tier_price | double | $/kWh | When using `bill_mode TIERED`, this determines the energy price after energy increases above `third_tier_energy` and is used to infinite energy. |
| first_tier_energy | double | kWh | Determines the point at which the price of energy changes from `price` to `first_tier_price`. |
| second_tier_energy | double | kWh | Determines the point at which the price of energy changes from `first_tier_price` to `second_tier_price`. |

| | | | |
|---|---|---|---|
| third_tier_energy | double | kWh | Determines the point at which the price of energy changes from `second_tier_price` to `third_tier_price`. |

### Meter State of Development

Meter is considered a highly developed and validated model in terms of powerflow solutions, however, models using billing features have not been fully validated.

## Triplex Node

Triplex nodes represent special cases of the `node` object. The `triplex_node` object still serves as connection point between different links of the system and a point of measurable voltage. However, `triplex_nodes` are casted to represent phases `1`, `2`, and `N` rather than `A`, `B`, and `C` like normal `node` objects. Simplified, they operate in the split-phase level of distribution rather than the three-phase level.

Since `load` objects are directly derived from `node` objects, they are only valid for three-phase connections as well. Therefore, the `load` functionality has been built into the `triplex_node` object for split-phase level systems.

It is important to note that triplex-based objects should include the phase `S` somewhere in their designation.

A typical `triplex_node` implementation is

```
object triplex_node {
        name TPL_tAS;
        phases AS;
        voltage_1 120 + 0j;
        voltage_2 120 + 0j;
        voltage_N 0;
        current_1  1.0;
        power_1 1000+2000j;
        shunt_1 5.3333e-004 -2.6667e-004i;
        nominal_voltage 120;
        };
```

### Triplex Node Parameters

`triplex_node` objects are technically derived from `node` objects as well. However due to the triplex nature of their use and the particular implementation, the normal `node` parameters are not available for use. Also note that both a shunt and impedance term are available. Only use one of these for accurate results.

| Property Name | Type | Unit | Description |
|---|---|---|---|
| bustype | enumeration | N/A | The type of bus the node represents. The different bus distinctions are only valid for the Gauss-Seidel and Newton-Raphson solver methods. The Forward-Back Sweep method (Kersting's method) does not presently incorporate anything other than the `PQ` bus. Valid choices are<br>▪ `PQ` for a constant power bus (default)<br>▪ `PV` for a voltage-controlled (magnitude) bus<br>▪ `SWING` for the infinite bus of a system. |
| busflags | enumeration | N/A | A flag to indicate if the current bus has a source or not. Mainly used for `PV` implementations. The only valid entries are |

| | | | |
|---|---|---|---|
| | | | `HASSOURCE` to indicate it is a supported bus, or an empty value indicating it is not. |
| reference_bus | object | N/A | A reference node elsewhere in the system that the `triplex_node` will use to obtain frequency information if necessary (unimplemented in GridLAB-D at this point). |
| maximum_voltage_error | double | Volts | The maximum voltage error for convergence checks in the different powerflow solvers. If left blank, it is derived from the `nominal_voltage` parameter. |
| voltage_1 | complex | Volts | The voltage on phase 1 of a split-phase or triplex system. This may be specified in rectangular (7200.0+0.0j) or polar (7200.0+0.0d) formats. |
| voltage_2 | complex | Volts | The voltage on phase 2 of a split-phase or triplex system. This may be specified in rectangular (7200.0+0.0j) or polar (7200.0+0.0d) formats. |
| voltage_N | complex | Volts | The voltage on the neutral phase of a split-phase or triplex system. This may be specified in rectangular (7200.0+0.0j) or polar (7200.0+0.0d) formats. |
| voltage_12 | complex | Volts | The voltage between phases `1` and `2` of the split-phase or triplex system. This is a derived quantity and can be read, but it is not recommended you set this value. |
| voltage_1N | complex | Volts | The voltage between phases `1` and `N` of the split-phase or triplex system. This is a derived quantity and can be read, but it is not recommended you set this value. |
| voltage_2N | complex | Volts | The voltage between phases `2` and `N` of the split-phase or triplex system. This is a derived quantity and can be read, but it is not recommended you set this value. |
| current_1 | complex | Amperes | Constant current load on phase `1` of the split-phase or triplex system. |
| current_2 | complex | Amperes | Constant current load on phase `2` of the split-phase or triplex system. |
| current_N | complex | Amperes | Constant current load on the neutral phase of the split-phase or triplex system. |
| current_12 | complex | Amperes | Constant current load on across phases `1` and `2` of the split-phase or triplex system. |
| power_1 | complex | Volt-Amperes | Constant power load on phase `1` of the split-phase or triplex system. |
| power_2 | complex | Volt-Amperes | Constant power load on phase `2` of the split-phase or triplex system. |
| power_12 | complex | Volt-Amperes | Constant power load across phases `1` and `2` of the split-phase or triplex system. |
| shunt_1 | complex | Siemens (mhos) | Constant admittance load on phase `1` of the split-phase or triplex system. |
| shunt_2 | complex | Siemens (mhos) | Constant admittance load on phase `2` of the split-phase or triplex system. |
| shunt_12 | complex | Siemens | Constant admittance load across phases `1` and |

| | | | |
|---|---|---|---|
| | | (mhos) | `2` of the split-phase or triplex system. |
| impedance_1 | complex | Ohms | Constant impedance load on phase `1` of the split-phase or triplex system. |
| impedance_2 | complex | Ohms | Constant impedance load on phase `2` of the split-phase or triplex system. |
| impedance_12 | complex | Ohms | Constant impedance load across phases `1` and `2` of the split-phase or triplex system. |

### Triplex Node State of Development

Triplex Node is considered a highly developed and validated model in terms of powerflow solutions, however, models may be developed to include more advanced features in the future.

## Triplex Meter

Triplex meters provide similar functionality for triplex systems that `meter` objects do in three-phase systems. A triplex meter provides a measurement point for power and energy on the system at a specific point. Coupled with a `recorder` or `collector`, the `triplex_meter` object provides a method determine how much power and energy have been used by downstream connections, as well as how much current is flowing through the meter object at the present time. A typical implementation would be

```
object triplex_meter {
       name TrplMtr1;
       phases AS;
       nominal_voltage 120.0;
       }
```

### Triplex Meter Parameters

A `triplex_meter` object is a derivation of the `triplex_node` object and thus inherits all of its parameters. Most `triplex_meter` parameters are meant to be read-only, but can be set if the need arises.

| Property Name | Type | Unit | Description |
|---|---|---|---|
| measured_real_energy | double | Watt-hours | Measurement of the real energy (accumulation of the real power) that has flowed through the `triplex_meter` since it was reset. |
| measured_reactive_energy | double | Volt-Amperes-hours | Measurement of the reactive energy (accumulation of the reactive power) that has flowed through the `triplex_meter` since it was reset. |
| measured_power | complex | Volt-Amperes | Measurement of the complex power flowing through the triplex meter at that instant in time. |
| measured_demand | double | Watts | Measurement of the peak power demand of downstream objects. |
| measured_real_power | double | Watts | Measurement of the real portion of the power flowing through the triplex meter at that instant in time. |
| measured_reactive_power | double | Volt-Amperes reactive | Measurement of the reactive portion of the power flowing through the meter at that instant in time. |
| indiv_measured_power_1 | complex | Volt- | Measures the complex power flowing |

| | | | |
|---|---|---|---|
| | | Amperes | through the meter on phase `1`. |
| indiv_measured_power_2 | complex | Volt-Amperes | Measures the complex power flowing through the meter on phase `2`. |
| indiv_measured_power_N | complex | Volt-Amperes | Measures the complex power flowing through the meter on phase `N`. |
| measured_voltage_1 | complex | Volts | Measurement of the voltage on phase `1` of the split-phase or triplex system. May or may not be as up to date as reading `voltage_1` directly. |
| measured_voltage_2 | complex | Volts | Measurement of the voltage on phase `2` of the split-phase or triplex system. May or may not be as up to date as reading `voltage_2` directly. |
| measured_voltage_N | complex | Volts | Measurement of the voltage on the neutral phase of the split-phase or triplex system.. May or may not be as up to date as reading `voltage_N` directly. |
| measured_current_1 | complex | Amperes | Measurement of the current on phase `1` of the triplex meter at that instant in time. |
| measured_current_2 | complex | Amperes | Measurement of the current on phase `2` of the triplex meter at that instant in time. |
| measured_current_N | complex | Amperes | Measurement of the current on the neutral phase of the triplex meter at that instant in time. |
| bill_day | int32 | N/A | Sets the date of the month at which the final monthly bill is calculated (at midnight). Maximum value is 28. |
| price | double | $/kWh | Determines the instantaneous market price of energy. Where the price comes from depends upon the `bill_mode`. |
| monthly_fee | double | $ | This is a recurrent monthly service charge that is added into the bill on the first day of the billing cycle (no pro-rating). |
| monthly_bill | double | $ | This is the running monthly bill at the particular meter as a function of price and the amount of energy used in that month. |
| previous_monthly_bill | double | $ | This stores the total bill from the previous month after the bill has been processed on the `bill_day`. |
| monthly_energy | double | kWh | The rolling amount of energy consumed during the current month at that meter. Used to calculate `monthly_bill`. |
| previous_monthly_energy | double | kWh | Stores the previous month's total energy consumption. |
| | | | Describes the method in which the meter receives its price signal.<br>0 - `NONE` - Billing is not used (default).<br>1 - `UNIFORM` - A static price is used through variable `price`, however, this may change over time using a player or schedule. |

| | | | |
|---|---|---|---|
| bill_mode | enumeration | N/A | 2 - TIERED - Tiered pricing plan where the price changes as a function of the amount of energy used in the month. See tier_price and tier_energy.<br>3 - HOURLY - This is used in conjunction with an auction or stubauction object. Receives its price directly from a market signal, but only updates on an hourly basis. Used in conjunction with power_market. NOTE: while this says "hourly", it will actually update any time the price changes in the auction.<br>4 - TIERED_RTP - Merges TIERED and HOURLY modes. Applies both a real time price via the auction to energy usage, but then also applies block / tiered rates to the total monthly energy use. |
| power_market | object | N/A | When using bill_mode HOURLY, this points the meter to the object where it will receive a price signal. |
| first_tier_price | double | $/kWh | When using bill_mode TIERED, this determines the energy price after energy increases above first_tier_energy, but below second_tier_energy. If second_tier_energy is not defined, then this price will be used to infinity. While energy is below first_tier_energy, price is used to calculate the monthly_bill. |
| second_tier_price | double | $/kWh | When using bill_mode TIERED, this determines the energy price after energy increases above second_tier_energy, but below third_tier_energy. If third_tier_energy is not defined, then this price will be used to infinity. |
| third_tier_price | double | $/kWh | When using bill_mode TIERED, this determines the energy price after energy increases above third_tier_energy and is used to infinite energy. |
| first_tier_energy | double | kWh | Determines the point at which the price of energy changes from price to first_tier_price. |
| second_tier_energy | double | kWh | Determines the point at which the price of energy changes from first_tier_price to second_tier_price. |
| third_tier_energy | double | kWh | Determines the point at which the price of energy changes from second_tier_price to third_tier_price. |

## Triplex Meter State of Development

Triplex Meter is considered a highly developed and validated model in terms of powerflow solutions, however, models using billing have not been fully validated. Additional features will be added as needed.

# Triplex Load

NEW in version 3.0!

Triplex load is similar to `load` and `ZIPload` in that load can be specified as a base load, then a ZIP fraction applied to that base load. The load can be place on phase 1 (120V), phase 2 (120V) or phase 12 (240V).

## Triplex Load Parameters

A `triplex_load` object is a derivation of the `triplex_node` object and thus inherits all of its parameters.

| Property Name | Type | Unit | Description |
|---|---|---|---|
| base_power_1 | double | VA | in similar format as ZIPload & load this represents the nominal power on phase 1 before applying ZIP fractions |
| base_power_2 | double | VA | in similar format as ZIPload & load this represents the nominal power on phase 2 before applying ZIP fractions |
| base_power_12 | double | VA | in similar format as ZIPload & load this represents the nominal power on phase 12 before applying ZIP fractions |
| power_pf_1 | double | pu | in similar format as ZIPload & load this is the power factor of the phase 1 constant power portion of load |
| current_pf_1 | double | pu | in similar format as ZIPload & load this is the power factor of the phase 1 constant current portion of load |
| impedance_pf_1 | double | pu | in similar format as ZIPload & load this is the power factor of the phase 1 constant impedance portion of load |
| power_pf_2 | double | pu | in similar format as ZIPload & load this is the power factor of the phase 2 constant power portion of load |
| current_pf_2 | double | pu | in similar format as ZIPload & load this is the power factor of the phase 2 constant current portion of load |
| impedance_pf_2 | double | pu | in similar format as ZIPload & load this is the power factor of the phase 2 constant impedance portion of load |
| power_pf_12 | double | pu | in similar format as ZIPload & load this is the power factor of the phase 12 constant power portion of load |
| current_pf_12 | double | pu | in similar format as ZIPload & load this is the power factor of the phase 12 constant current portion of load |
| impedance_pf_12 | double | pu | in similar format as ZIPload & load this is the power factor of the phase 12 constant impedance portion of load |
| power_fraction_1 | double | pu | this is the constant power fraction of base power on phase 1 |
| current_fraction_1 | double | pu | this is the constant current fraction of base power on phase 1 |
| impedance_fraction_1 | double | pu | this is the constant impedance fraction of base power on phase 1 |

| | | | |
|---|---|---|---|
| power_fraction_2 | double | pu | this is the constant power fraction of base power on phase 2 |
| current_fraction_2 | double | pu | this is the constant current fraction of base power on phase 2 |
| impedance_fraction_2 | double | pu | this is the constant impedance fraction of base power on phase 2 |
| power_fraction_12 | double | pu | this is the constant power fraction of base power on phase 12 |
| current_fraction_12 | double | pu | this is the constant current fraction of base power on phase 12 |
| impedance_fraction_12 | double | pu | this is the constant impedance fraction of base power on phase 12 |
| measured_voltage_1 | complex | V | the current measured voltage on phase 1 |
| measured_voltage_2 | complex | V | the current measured voltage on phase 2 |
| measured_voltage_12 | complex | V | the current measured voltage on phase 12 |

### Triplex Load State of Development

Triplex load is considered a relatively new model based on existing models. However, the object has only been minimally validated.

## Regulator

Regulators are essentially tap-changing transformers that attempt to maintain a voltage level at a specified point in the system. Regulators are one of two objects in the `powerflow` module that incorporate a form of automatic control. To take full advantage of this functionality, simulations of greater than one time step (time-varying simulations) are recommended. Similar to `transformer` and `line` objects, regulators require a `regulator_configuration` to determine many of their operating parameters.

A typical implementation would be

```
object regulator {
        name Reg799781;
        phases "ABC";
        from node_799;
        to node_781;
        configuration reg_conf_79978101;
        }
```

### Regulator Parameters

A `regulator` object is a derived class from `link` objects. Therefore, all of the parameters available to the `link` object apply here as well.

| Property Name | Type | Unit | Description |
|---|---|---|---|
| configuration | object | N/A | `regulator_configuration` object that describes the specific regulator implementation. |
| tap_A | int16 | N/A | Position of the tap on phase `A` of a wye-connected or phase `AB` of a delta-connected system. This parameter is most useful to be read in automatic regulator modes, but serves as the input for tap position of the phase under the manual control scheme. |

| | | | |
|---|---|---|---|
| tap_B | int16 | N/A | Position of the tap on phase `B` of a wye-connected or phase `BC` of a delta-connected system. This parameter is most useful to be read in automatic regulator modes, but serves as the input for tap position of the phase under the manual control scheme. |
| tap_C | int16 | N/A | Position of the tap on phase `C` of a wye-connected or phase `CA` of a delta-connected system. This parameter is most useful to be read in automatic regulator modes, but serves as the input for tap position of the phase under the manual control scheme. |
| sense_node | object | N/A | Remote node for the automatic control method to monitor. Only utilized in `REMOTE_NODE` control scheme. This must be a `node`-based object to work properly. |
| tap_A_change_count | int16 | N/A | Holds the number of times the tap position on phase `A` of a wye-connected or phase `AB` of a delta-connected system has changed. |
| tap_B_change_count | int16 | N/A | Holds the number of times the tap position on phase `B` of a wye-connected or phase `BC` of a delta-connected system has changed. |
| tap_C_change_count | int16 | N/A | Holds the number of times the tap position on phase `C` of a wye-connected or phase `CA` of a delta-connected system has changed. |

### Regulator State of Development

Regulator is considered a well developed and validated model in terms of powerflow solutions, however, models may be developed to include more advanced features in the future. Additional configurations, controls, and/or losses may be included as needed.

## Regulator Configuration

The `regulator_configuration` object describes the details of a particular `regulator` object implementation. This includes details such as the control scheme, regulator type, sensing information, and time delays. A typical regulator configuration would look similar to

```
object regulator_configuration {
        name reg_conf_79978101;
        connect_type 2;
        band_center 122.000;
        band_width 2.0;
        time_delay 30.0;
        raise_taps 16;
        lower_taps 16;
        current_transducer_ratio 350;
        power_transducer_ratio 40;
        compensator_r_setting_A 1.5;
        compensator_x_setting_A 3.0;
        compensator_r_setting_B 1.5;
        compensator_x_setting_B 3.0;
        CT_phase "ABC";
        PT_phase "ABC";
        regulation 0.10;
        Control MANUAL;
        control_level INDIVIDUAL;
        Type A;
        tap_pos_A 7;
        tap_pos_B 4;
        }
```

### Regulator Configuration Parameters

| Property Name | Type | Unit | Description |
|---|---|---|---|

| | | | |
|---|---|---|---|
| connect_type | enumeration | N/A | Selection method for the electrical connection type of the regulator implemented. Valid types may be referred to by number or keyword<br>0 - `UNKNOWN` - Unknown regulator implementation that will throw an error if used<br>1 - `WYE_WYE` - Wye connected regulator implementation<br>2 - `OPEN_DELTA_ABBC` - Open delta connected regulator with CA open - **Note: Unimplemented at this time**<br>3 - `OPEN_DELTA_BCAC` - Open delta connected regulator with AB open - **Note: Unimplemented at this time**<br>4 - `OPEN_DELTA_CABA` - Open delta connected regulator with BC open - **Note: Unimplemented at this time**<br>5 - `CLOSED_DELTA` - Closed delta connected regulator implementation - **Note: Unimplemented at this time** |
| band_center | double | Volts | Center point of the voltage level desired. |
| band_width | double | Volts | Allowed range for the voltage to vary before a change is implemented. Centered around `band_center`, so limits are at `band_center - band_width/2` and `band_center + band_width/2`. |
| time_delay | double | seconds | Amount of time from a change request to the physical changing of the tap position on the regulator. Represents mechanical delays in the regulator. |
| dwell_time | double | seconds | Amount of time a change must be consistently requested before enacted upon. Represents a transient filter or additional hysteresis implementation to prevent excessive tap changes due to transient spikes. |
| raise_taps | int16 | N/A | Upper limit of tap positions allowed in the regulator. |
| lower_taps | int16 | N/A | Lower limit of tap positions allowed in the regulator. Note: This value is represented as a magnitude value. The actual lower limit of the tap positions is assumed to be `-lower_taps`. |
| current_transducer_ratio | double | per-unit | Turns ratio for current transducer for the line-drop compensator control method. |
| power_transducer_ratio | double | per-unit | Turns ratio for the power transducer for the line-drop compensator control method. |
| compensator_r_setting_A | double | Volts | Compensator resistive value for phase `A`. |
| compensator_r_setting_B | double | Volts | Compensator resistive value for phase `B`. |
| compensator_r_setting_C | double | Volts | Compensator resistive value for phase `C`. |
| compensator_x_setting_A | double | Volts | Compensator reactive value for phase `A`. |
| compensator_x_setting_B | double | Volts | Compensator reactive value for phase `B`. |
| compensator_x_setting_C | double | Volts | Compensator reactive value for phase `C`. |
| | | | Current transducer connection phase. Valid keywords are |

| | | | |
|---|---|---|---|
| CT_phase | set | N/A | <ul><li>`A` - Phase `A` current transducer</li><li>`B` - Phase `B` current transducer</li><li>`C` - Phase `C` current transducer</li></ul>**Note: This function is not implemented at this time.** |
| PT_phase | set | N/A | Power transducer connection phase. Valid keywords are<ul><li>`A` - Phase `A` power transducer</li><li>`B` - Phase `B` power transducer</li><li>`C` - Phase `C` power transducer</li></ul>**Note: This function is not implemented at this time.** |
| regulation | double | N/A | Indicates range of voltage adjustment possible (i.e., per tap change ratio equals regulation / raise taps, or regulation of 0.1 indicates 10% rise in voltage at maximum tap position) |
| Control | enumeration | N/A | Defines the control scheme the regulator will use to operate. Valid keywords are:<ul><li>`MANUAL` - Manual control mode. User specifies all tap changes.</li><li>`OUTPUT_VOLTAGE` - Output node of the regulator's voltage is examined. Tap changes are performed based on `band_center` and `band_width`.</li><li>`LINE_DROP_COMP` - Line drop compensator control mode. Utilizes compensator information in addition to `band_center` and `band_width` to determine tap changes.</li><li>`REMOTE_NODE` - Voltage of a remote node (specified by `sense_node` in the `regulator` object) in the system is examined. Tap changes are performed based on `band_center` and `band_width`.</li></ul> |
| control_level | enumeration | N/A | Defines how automatic controls influence the tap settings of the regulator. Valid keywords are:<ul><li>`INDIVIDUAL` - Each phase is controlled individually.</li><li>`BANK` - All phases are controlled identically. Using the `PT_phase` property, the regulator determines any control actions and applies it to all phases identically.</li></ul> |
| Type | enumeration | N/A | Type of step-voltage regulator implemented. Valid keywords are:<ul><li>`A` - Type A step-voltage regulator</li><li>`B` - Type B step-voltage regulator</li></ul> |
| tap_pos_A | int16 | N/A | Initial tap position for phase `A`. If left empty, the regulator will take a best guess at the |

| | | | |
|---|---|---|---|
| | | | initial tap position. |
| tap_pos_B | int16 | N/A | Initial tap position for phase `B`. If left empty, the regulator will take a best guess at the initial tap position. |
| tap_pos_C | int16 | N/A | Initial tap position for phase `C`. If left empty, the regulator will take a best guess at the initial tap position. |

### Regulator Configuration State of Development

Regulator Configuration is considered a well developed and validated model in terms of powerflow solutions, however, models may be developed to include more advanced features in the future. Additional configurations, controls, and/or losses may be included as needed.

# Capacitor

Capacitors are used for reactive power compensation and voltage support scenarios. The `capacitor` implements a switchable set of capacitors. `capacitor` objects are one of two objects in the `powerflow` module that incorporate a form of automatic control. To take full advantage of this functionality, simulations of greater than one time step (time-varying simulations) are recommended. Single-phase powerflow connections (phase `S`) are not supported by capacitors at this time. A typical capacitor implementation is

```
object capacitor {
        phases ABCN;
        name CapNode;
        phases_connected ABCD;
        control MANUAL;
        capacitor_A 0.5 MVAr;
        capacitor_B 0.5 MVAr;
        capacitor_C 0.5 MVAr;
        control_level INDIVIDUAL;
        switchA OPEN;
        switchB OPEN;
        switchC CLOSED;
        nominal_voltage 7200;
        }
```

### Capacitor Parameters

`capacitor` objects are derived from `node` objects, so any parameters of the `node` object are available as well.

| Property Name | Type | Unit | Description |
|---|---|---|---|
| pt_phase | set | N/A | Determines the participating phases. These are the phases the various control schemes will monitor to determine their actions. Follows the same conventions as the overall `phases` property in powerflow. |
| phases_connected | set | N/A | Connection of the capacitors. This allows for delta-connected capacitors on a wye-connected system and vice versa. Follows the same conventions as the overall `phases` property. If left empty or undefined, defaults to the `phases` property of the capacitor. |
| switchA | enumeration | N/A | Status of the switch that enables or disables the capacitor attached to phase `A` if wye-connected or `AB` if delta-connected. |
| switchB | enumeration | N/A | Status of the switch that enables or disables the capacitor attached to phase `B` if wye-connected or |

| | | | |
|---|---|---|---|
| | | | `BC` if delta-connected. |
| switchC | enumeration | N/A | Status of the switch that enables or disables the capacitor attached to phase `C` if wye-connected or `CA` if delta-connected. |
| cap_A_switch_count | int16 | N/A | Hold of the number of times the switch has changed (OPEN to CLOSED, CLOSED to OPEN) on phase `A` if wye-connected or `AB` if delta-connected. |
| cap_B_switch_count | int16 | N/A | Hold of the number of times the switch has changed (OPEN to CLOSED, CLOSED to OPEN) on phase `B` if wye-connected or `BC` if delta-connected. |
| cap_C_switch_count | int16 | N/A | Hold of the number of times the switch has changed (OPEN to CLOSED, CLOSED to OPEN) on phase `C` if wye-connected or `CA` if delta-connected. |
| control | enumeration | N/A | Defines the control scheme the capacitor will utilize to perform switching operations. Valid control mode keywords are<br>▪ `MANUAL` - Capacitor switching is controlled manually through `switchA`, `switchB`, and `switchC`.<br>▪ `VAR` - VAR controlled mode. A remote line needs to be specified in `remote_sense` or `remote_sense_B` that will have its reactive power checked against `VAr_set_high` and `VAr_set_low`.<br>▪ `VOLT` - Voltage controlled mode. The capacitor node itself or a node specified by `remote_sense` or `remote_sense_B` has its voltage checked against `voltage_set_high` and `voltage_set_low`.<br>▪ `VARVOLT` - Combination control scheme. Has two modes. If `voltage_set_low` is specified, performs control similar to `VOLT` first, and then `VAR` second. If `voltage_set_low` is unspecified or set to zero, operates in `VAR` mode primarily. However, `voltage_set_high` is monitored and will switch the capacitors off and lock them out if exceeded (voltage safety). |
| voltage_set_high | double | Volts | High setpoint for voltage-based capacitor switching operations. This setpoint will turn the capacitors off. |
| voltage_set_low | double | Volts | Low setpoint for voltage-based capacitor switching operations. This setpoint will turn the capacitors on. |
| VAr_set_high | double | Volt-Amperes reactive | High setpoint for VAr-based capacitor switching operations. This setpoint will turn the capacitors on. |
| VAr_set_low | double | Volt-Amperes reactive | Low setpoint for VAr-based capacitor switching operations. This setpoint will turn the capacitors off. |
| capacitor_A | double | Volt-Amperes reactive | Capacitor size information for capacitor connected to phase `A` in a wye connection or on phase `AB` in a delta connection. |

| | | | |
|---|---|---|---|
| capacitor_B | double | Volt-Amperes reactive | Capacitor size information for capacitor connected to phase `B` in a wye connection or on phase `BC` in a delta connection. |
| capacitor_C | double | Volt-Amperes reactive | Capacitor size information for capacitor connected to phase `C` in a wye connection or on phase `CA` in a delta connection. |
| cap_nominal_voltage | double | Volts | Capacitor rated nominal voltage. Used for situations when `nominal_voltage` doesn't match the rated voltage or if a line-to-line voltage is specified when the capacitors are on a wye-connected system. If left blank, defaults to the `nominal_voltage` specified. |
| time_delay | double | seconds | Time delay before any capacitor switching operation takes place. Represents mechanical switching delays. |
| dwell_time | double | seconds | Time period a switching operation must be consistently requested before any switching operation is attempted. Serves as a transient filter or additional hysteresis to prevent transient events from causing excessive capacitor switching. |
| lockout_time | double | seconds | Time period a capacitor will lock out switching operations after `voltage_set_high` is exceeded in the `VARVOLT` control method. |
| remote_sense | object | N/A | Remote `node` or `link` object for `VOLT`, `VAR`, or `VARVOLT` control schemes. If a `node` object is specified, the remote voltage is read. If `link` object is specified, the reactive power is read. |
| remote_sense_B | object | N/A | Remote `node` or `link` object for `VOLT`, `VAR`, or `VARVOLT` control schemes. If a `node` object is specified, the remote voltage is read. If `link` object is specified, the reactive power is read. Under the `VARVOLT` control scheme, this must be the opposite of the type specified in `remote_sense`. |
| control_level | enumeration | N/A | Specifies how the switching action occurs for all phases of the capacitor. Valid keywords are<br>▪ `BANK` - All capacitors are switched based on the control scheme and `pt_phase` property.<br>▪ `INDIVIDUAL` - Capacitors are switched individually based on the control scheme and `pt_phase` property. |

### Capacitor State of Development

Capacitor is considered a well developed and validated model in terms of powerflow solutions, however, models may be developed to include more advanced features in the future. Additional configurations and controls may be included as needed.

## Fuse

Fuse objects are used to place a current limitation between two nodes. If the current is exceeded, the fuse will open and prevent further current flow. Due to limitations in the Forward-Back Sweep algorithm, fuses only affect the first downstream node. If other loads exist downstream, they will cause an oscillatory voltage swing that has no real representation. `reliability` module functionality only exists in the Newton-Raphson solver at this time. A minimalist `fuse` could be implemented as

```
object fuse {
        phases "ABC";
        name node1-node2;
        from node1;
        to node2;
        }
```

A typical `fuse` object would be implemented, with the same parameters as above, as

```
object fuse {
       phases "ABC";
       name node1-node2;
       from node1;
       to node2;
       current_limit 9999.0 A;
       mean_replacement_time 3600.0;
       repair_dist_type NONE;
       }
```

## Fuse Parameters

`fuse` objects are derived from `link` objects, so all of those properties are available.

| Property Name | Type | Unit | Description |
|---|---|---|---|
| current_limit | double | Amperes | Current rating for the fuse. If exceeded, the particular phase will go to an open circuit condition. |
| mean_replacement_time | double | seconds | Mean time to replace the fuse if blown. This could represent a travel requirement (remote location) or scarcity requirement (shipping time for parts). This value overrides any value specified in `mean_repair_time` for the `link` object itself. |
| phase_A_status | enumeration | N/A | Status of the fuse on phase `A` (only valid if phase `A` is in the `phases` property). Two keywords are valid:<br>▪ GOOD - The fuse on phase `A` is still conducting and has not exceeded its current limit.<br>▪ BLOWN - The fuse on phase `A` has exceeded its current limit and is no longer conducting. |
| phase_B_status | enumeration | N/A | Status of the fuse on phase `B` (only valid if phase `B` is in the `phases` property). Two keywords are valid:<br>▪ GOOD - The fuse on phase `B` is still conducting and has not exceeded its current limit.<br>▪ BLOWN - The fuse on phase `B` has exceeded its current limit and is no longer conducting. |
| | | | Status of the fuse on phase `C` (only valid if phase `C` is in the `phases` property). Two keywords are valid: |

| | | | |
|---|---|---|---|
| phase_C_status | enumeration | N/A | ▪ GOOD - The fuse on phase `C` is still conducting and has not exceeded its current limit.<br>▪ BLOWN - The fuse on phase `C` has exceeded its current limit and is no longer conducting. |
| repair_dist_type | enumeration | N/A | Distribution to be used after a fuse has blow to restore it. Current valid settings are:<br>▪ `NONE` - No distribution is used and the value in `mean_replacement_time` is taken directly.<br>▪ `EXPONENTIAL` - An exponential distribution is used with `mean_replacement_time` taken as one over the lambda value. |

### Fuse State of Development

Fuse is considered a well developed and validated model in terms of powerflow solutions. Reliability functionality has been tested and validated, but is not fully vetted and may change as advanced functionality is included.

## Switch

Switch objects are used to change topology and add or remove elements from a powerflow system. When a switch is opened, no current flow is permitted and the downstream objects will be effectively removed from the system. A typical switch implementation is

```
object switch {
        name switch1;
        phases ABCN;
        from node_250;
        to node_243;
        status CLOSED;
        }
```

### Switch Parameters

Switch objects are derived from `link` objects and share all of those available parameters. `switch` objects have additional parameters of

| Property Name | Type | Unit | Description |
|---|---|---|---|
| phase_A_state | enumeration | N/A | Status of the phase A portion of the switch. Valid states are:<br>▪ `OPEN` - `switch` is open and no current can flow<br>▪ `CLOSED` - `switch` is closed and conducting |
| phase_B_state | enumeration | N/A | Status of the phase B portion of the switch. Valid states are:<br>▪ `OPEN` - `switch` is open and no current can flow<br>▪ `CLOSED` - `switch` is closed and conducting |
| phase_C_state | enumeration | N/A | Status of the phase C portion of the switch. Valid states are:<br>▪ `OPEN` - `switch` is open and no current can flow<br>▪ `CLOSED` - `switch` is closed and conducting |

| | | | |
|---|---|---|---|
| operating_mode | enumeration | N/A | Switching operations governing criterion. Two settings are available:<br>▪ `INDIVIDUAL` - each phase of the `switch` object is controlled individually.<br>▪ `BANKED` - all valid phases of the `switch` object are controlled together. A voting scheme of valid phases is used to determine this. For example, if a `switch` has phases A, B, and C with only phase C closed, phase A and B will override it and open phase A. |

### Switch State of Development

Switch is considered a well developed and validated model in terms of powerflow solutions, however, models incorporating fault analysis, reliability, and other advanced features have not been validated.

## Recloser

`recloser` objects are a special type of `switch` that open at the detection of a fault condition and will close if the fault condition is removed or isolated within a certain period of time. The time is typically determined by the number of closing tries and the time between tries. `recloser` objects work with both the `FBS` and `NR` solver methods, but their reliability functionality only works with the `NR` method. A typical recloser implementation is

```
object recloser {
    name recloser_2;
    phases "ABCN";
    from node_2a;
    to node_2b;
    retry_time 1s;
    max_number_of_tries 3;
}
```

Recloser objects inherit for switch and therefore share all parameters belonging to both `switch` and `link`. `recloser` objects are exercised only by the `reliability` module at this time.

### Recloser Parameters

| Property Name | Type | Unit | Description |
|---|---|---|---|
| retry_time | double | seconds | The time to wait in seconds before trying to close after a fault condition is detected. This parameter is unused at this time and is put in place for future functionality. |
| maximum_number_of_tries | double | unitless | the maximum number of times the `recloser` tries to close and fails before it permanently opens. |
| number_of_tries | double | unitless | number of tries a `recloser` has been actuated in the current fault condition. |

### Recloser State of Development

Recloser is considered a validated model in terms of powerflow solutions and `reliability` integration. However, testing has been limited and feature additions may still be necessary.

## Relay

Relays are used to provide momentary breaks in the system and are implemented as reclosers. A `relay` object only functions in on a basic level and does not provide any `reliability`-module-functionality at this time. It is untested in the `NR` solver and a `recloser` object is suggested instead. A typical relay would be implemented as

```
object relay {
        name recloser_A;
        phases ABCN;
        from node_1;
        to load_5;
        time_to_change 1.0s;
        recloser_delays 5.0s;
        recloser_tries 5;
        }
```

### Relay Parameters

`relay` objects are derived from `link` objects, so all of those parameters should be available.

| Property Name | Type | Unit | Description |
|---|---|---|---|
| time_to_change | double | seconds | Time to physically change out or reset the relay/recloser after it has locked out. **Note: This feature is unimplemented at this time.** |
| recloser_delay | double | seconds | Time from a trip before the `recloser` will attempt to close the circuit again. |
| recloser_tries | int16 | N/A | Number of reclosing attempts before the `recloser` object will lock in the open position. |

### Relay State of Development

Relay is considered a well developed and validated model in terms of `FBS`-based powerflow solutions, however, models incorporating fault analysis, reliability, and other advanced features have not been validated.

## Substation

Substations were used to connect distribution powerflow in the `powerflow` module with PowerWorld through the `network` module. The `substation` object converts the sequence voltage provided by the `network` module to three-phase swing bus voltage for the unbalanced three-phase powerflow solution. The `substation` object also passes the unbalanced three-phase powerflow solution to back to the `network` module as an single power value representing the average load on all three phases of the swing bus. Furthermore, the `substation` object sets which phase is the reference phase for the distribution powerflow. A typical substation implementation is

```
object substation {
        name SubS;
        bustype SWING;
        parent network_node;
        reference_PHASE_A;
        phase ABCN;
        nominal_voltage 7199.558;
}
```

In order to properly connect the substation object to the `network` module, the substation object's parent must be a pw_load object.

### Substation Parameters

The `substation` object is derived from the `node` object in the `powerflow` module. As a result, all parameters of that object are definable as well.

| Property Name | Type | Unit | Description |
|---|---|---|---|
| positive_sequence_voltage | complex | Volts | The positive sequence voltage given from a PowerWorld bus model or a user specified value. |
| reference_phase | enumeration | None | The phase that will be used as the reference angle for the powerflow solution.<br>• PHASE_A(Default)<br>• PHASE_B<br>• PHASE_C |
| transmission_level_constant_power_load | complex | Volt-Amperes | The positive-sequence constant power load to be posted directly to the pw_load object (powerflow solver does not handle this, it is explicitly converted and posted to PowerWorld's solver). |
| transmission_level_constant_impedance_load | complex | Ohms | The positive-sequence constant impedance load to be posted directly to the pw_load object (powerflow solver does not handle this, it is explicitly converted and posted to PowerWorld's solver). |
| transmission_level_constant_current_load | complex | Amperes | the positive-sequence constant current load to be posted directly to the pw_load object (powerflow solver does not handle this, it is explicitly converted and posted to PowerWorld's solver). |
| average_distribution_load | complex | Volt-Amperes | The average of the loads on all three phases at the substation object. |
| distribution_power_A | complex | Volt-Amperes | The measured power of the attached powerflow on phase A. |
| distribution_power_B | complex | Volts-Amperes | The measured power of the attached powerflow on phase B. |
| distribution_power_C | complex | Volts-Amperes | The measured power of the attached powerflow on phase C. |

### Substation State of Development

The only transmission powerflow software that substation currently works with is PowerWorld. Please note that the specific the transmission current and impedance loads are converted to complex power values first and then posted to the proper properties(load_current and load_impedance) in the pw_load object. The average_transmission_power_load value must be added to the average_distribution_load before posting to pw_load(load_power).

Substation's three phase voltages are determined differently dependent upon three scenarios. If there is a pw_load object attached to the substation object, then the three phase voltages are determined by the sequence voltage value read from the pw_load object. The three phase voltages are determined by the positive_sequence_voltage property if there is a player object populating that property in the absence of a pw_load object. In the absence of a player object and a pw_load object, the three phase voltages are determined by user input or the substation object's powerflow parent just like any node object. If there is no pw_load connected to the substation then the substation doesn't post the average_distribution_load, average_transmission_current_load, average_transmission_impedance_load, and average_transmission_power_load properties.

# Parametric Load

Parametric loads provide a `load`-like object that allows the load to vary based on other conditions in the system. This may be things such as weather conditions or even time-of-day scheduling. Further details on parametric loads can be found in the Industrial and agricultural loads page. A typical parametric load would be called as

```
object pqload {
        name pqload1;
        phases ABC;
        Zp 200 ohm;
        Zq_T 250 F;
        Im 300;
        Ia 45;
        Pp 100;
        Pp_T 3;
        Pq_T 1;
        nominal_voltage 2400;
        }
```

### Parametric Load Parameters

The `pqload` object is directly derived from the `load` object and thus derived from the `node` object as well. As such, parameters of those two objects are also available for use, but most `load` parameters will probably be overwritten.

| Property Name | Type | Unit | Description |
|---|---|---|---|
| weather | object | N/A | Link to the `climate` object used in for parameters of the load. |
| T_nominal | double | Fahrenheit | Nominal temperature **Note: Unused at this time** |
| Zp_T | double | Ohm/degree Fahrenheit | Coefficient of how resistive impedance varies for temperature. |
| Zp_H | double | Ohm | Coefficient for how resistive impedance varies with humidity. |
| Zp_S | double | Ohm-Hour/BTU | Coefficient for how resistive impedance varies with solar gains. |
| Zp_W | double | Ohm-Hour/Mile | Coefficient for how resistive impedance varies with wind speed. |
| Zp_R | double | Ohm-Hour/Inch | Coefficient for how resistive impedance varies with rain fall. |

| | | | |
|---|---|---|---|
| Zp | double | Ohm | Baseline, unvarying resistive impedance value. |
| Zq_T | double | Farad/degree Fahrenheit | Coefficient of how reactive impedance varies for temperature. |
| Zq_H | double | Farad | Coefficient for how reactive impedance varies with humidity. |
| Zq_S | double | Farad-Hour/BTU | Coefficient for how reactive impedance varies with solar gains. |
| Zq_W | double | Farad-Hour/Mile | Coefficient for how reactive impedance varies with wind speed. |
| Zq_R | double | Farad-Hour/Inch | Coefficient for how reactive impedance varies with rain fall. |
| Zq | double | Farads | Baseline, unvarying reactive impedance value. |
| Im_T | double | Ampere/degree Fahrenheit | Coefficient of how current magnitude varies for temperature. |
| Im_H | double | Ampere | Coefficient for how current magnitude varies with humidity. |
| Im_S | double | Ampere-Hour/BTU | Coefficient for how current magnitude varies with solar gains. |
| Im_W | double | Ampere-Hour/Mile | Coefficient for how current magnitude varies with wind speed. |
| Im_R | double | Ampere-Hour/Inch | Coefficient for how current magnitude varies with rain fall. |
| Im | double | Ampere | Baseline, unvarying current magnitude value. |
| Ia_T | double | degrees/degree Fahrenheit | Coefficient of how current angle varies for temperature. |
| Ia_H | double | degrees | Coefficient for how current angle varies with humidity. |
| Ia_S | double | degree-Hour/BTU | Coefficient for how current angle varies with solar gains. |
| Ia_W | double | degree-Hour/Mile | Coefficient for how current angle varies with wind speed. |
| Ia_R | double | degree-Hour/Inch | Coefficient for how current angle varies with rain fall. |
| Ia | double | degrees | Baseline, unvarying current angle value. |
| Pp_T | double | Watts/degree Fahrenheit | Coefficient of how resistive power varies for temperature. |
| Pp_H | double | Watts | Coefficient for how resistive power varies with humidity. |
| Pp_S | double | Watt-Hour/BTU | Coefficient for how resistive power varies with solar gains. |
| Pp_W | double | Watt-Hour/Mile | Coefficient for how resistive power varies with wind speed. |
| Pp_R | double | Watt-Hour/Inch | Coefficient for how resistive power varies with rain fall. |
| Pp | double | Watts | Baseline, unvarying resistive power value. |

| | | | |
|---|---|---|---|
| Pq_T | double | Volt-Amperes reactive/degree Fahrenheit | Coefficient of how reactive power varies for temperature. |
| Pq_H | double | Volt-Amperes reactive | Coefficient for how reactive power varies with humidity. |
| Pq_S | double | Volt-Amperes reactive-Hour/BTU | Coefficient for how reactive power varies with solar gains. |
| Pq_W | double | Volt-Amperes reactive-Hour/Mile | Coefficient for how reactive power varies with wind speed. |
| Pq_R | double | Volt-Amperes reactive-Hour/Inch | Coefficient for how reactive power varies with rain fall. |
| Pq | double | Volt-Amperes reactive | Baseline, unvarying reactive power value. |
| input_temp | double | degrees Fahrenheit | Observed temperature. *(read-only)* |
| input_humid | double | Percentage | Observed humidity. *(read-only)* |
| input_solar | double | BTU/hour | Observed solar gains. *(read-only)* |
| input_wind | double | Miles/hour | Observed wind speed. *(read-only)* |
| input_rain | double | inches/hour | Observed rainfall. *(read-only)* |
| output_imped_p | double | Ohms | Observed load resistive impedance value. *(read-only)* |
| output_imped_q | double | Ohms | Observed load reactive impedance value. *(read-only)* |
| output_current_m | double | Amperes | Observed load current magnitude value. *(read-only)* |
| output_current_a | double | degrees | Observed load current angular value. *(read-only)* |
| output_power_p | double | Watts | Observed load resistive power value. *(read-only)* |
| output_power_q | double | Volt-Amperes | Observed load reactive power value. *(read-only)* |
| output_impedance | complex | Ohms | Observed load combined impedance value. *(read-only)* |
| output_current | complex | Amperes | Observed load combined current value. *(read-only)* |
| output_power | complex | Volt-Amperes | Observed load combined power value. *(read-only)* |

### PQ Load State of Development

PQ Load is considered an experimental model and has not been validated at this time.

## Volt-VAr Control

With multiple feeders attached to a common point, it is often useful to coordinate the voltage regulators and capacitors on the system. The `volt_var_control` object coordinates selected `regulator` and `capacitor` objects on the system. Using voltage measurements at `node` object points, the `volt_var_control` tries to maintain a desired voltage. In addition to voltage measurements, the `volt_var_control` utilizes a power measurement at a `link` object to determine how to switch various `capacitor` objects on the system in and out of service. Due to differences in the timing of power calculations in the Forward-Back Sweep and Newton-Raphson powerflow solvers, capacitors may switch at slightly different intervals for the same system. The overall control behaves the same in both solver methods, but this difference in capacitor timing may result in different final operating points. A typical Volt-VAr Controller implementation is

```
object volt_var_control {
        name IVVC37;
        control_method ACTIVE;
        capacitor_delay 10.0;
        regulator_delay 5.0;
        desired_pf 0.98;
        d_max 0.8;
        d_min 0.1;
        substation_link "SubTransNode-799";
        regulator_list "reg799-781,regnode799-U0081";
        capacitor_list "CapNode_A,CapNode_B";
        voltage_measurements "load829,1,load841,1,load825,1,U0029,2,U0041,2,U0025,2";
        minimum_voltages 2500.0;
        maximum_voltages 3000.0;
        desired_voltages 2600.0;
}
```

Many of the parameters on the Volt-VAr Controller can be left unspecified. When unspecified, default values or criteria are enacted. A minimalist implemenation would look similar to

```
object volt_var_control {
        name IVVC37;
        regulator_list "reg799-781,regnode799-U0081";
        capacitor_list "CapNode_A,CapNode_B";
}
```

## Volt-VAr Control Parameters

The `volt_var_control` object only derives properties from general `powerflow` module sets. It may share similar names to `node` and `link` parameters, but it is not a subclass of either of these objects.

| Property Name | Type | Unit | Description |
|---|---|---|---|
| control_method | enumeration | N/A | Defines the control scheme the `volt_var_control` object is currently operating in. There are two modes currently supported:<br>▪ `STANDBY` - The `volt_var_control` is inactive and all `regulator` and `capacitor` control is handled by their local definitions.<br>▪ `ACTIVE` - The `volt_var_control` is actively adjusting the regulators of interest, as well as coordinating capacitor insertions and removals. This is the default state. |
| capacitor_delay | double | seconds | Default delay for any `capacitors` under the `volt_var_control` object's control. If a delay is not explicitly defined in the `capacitor`'s local properties, this value will be used as the time delay for all switching operations. Defaults to 5.0 seconds. |
| regulator_delay | double | seconds | Default delay for any `regulators` under the `volt_var_control` object's control. If a delay is not explicityly defined in the `regulator_configuration` object association with the `regulator`, this value will be used as the time delay in all tap-changing operations. Defaults to 5.0 seconds. |
| | | | Desired power-factor for the system to try and |

| | | | |
|---|---|---|---|
| desired_pf | double | N/A | achieve. Used for setting threshold values for switching `capacitors` in and out of the system. The power factor is determined at the `substation_link` object. Defaults to 0.98. |
| d_max | double | N/A | Scaling constant for switching the `capacitors` on as a ratio of their size to the required reactive power correction. Typically between 0.3 and 0.6. It must not overlap with `d_min`, and a larger spread between the two values helps prevent `capacitor` switching oscillations. Defaults to 0.6. |
| d_min | double | N/A | Scaling constant for switching the `capacitors` off as a ratio of their size and the required reactive power correction. Typically between 0.1 and 0.4. It must not overlap with `d_max`. As with `d_max`, a larger spread between `d_min` and `d_max` will help prevent `capacitor` switching oscillations. Defaults to 0.3. |
| substation_link | object | N/A | Defines the `link` to extract power information from for power factor correction. This object must be of the main-type `link`. Typically, this will be attached to a subtransmission-level `transformer` object to monitor reactive power for the entire network. If unpopulated, this defaults to monitoring the power through the first regulator specified in `regulator_list`. |
| pf_phase | set | N/A | Defines the phases of `substation_link` to monitor and accumulate for the power factor calculations. `pf_phase` can be any combination of `PHASE_A`, `PHASE_B`, or `PHASE_C`. If left blank, `pf_phase` will default to the phases of the `substation_link` object. |
| regulator_list | char1024 | Regulator objects (implied) | List of regulators for the `volt_var_control` object to control. The list is a comma-separated object name for each regulator for which control is desired. No trailing comma is required. One regulator would be specified as `regulator_list reg799781;`, while three would be<br><br>`regulator_list reg799781,reg981,reg01;`. |
| capacitor_list | char1024 | Capacitor objects (implied) | List of capacitors for the `volt_var_control` object to control. This list is also comma-separated, like the `regulator_list`. If no capacitors are specified, no reactive adjustments are performed. All capacitors are operated in banked mode by the `volt_var_control` object. The listed capacitors are sorted by size and distance for switching operations. An example capacitor list is<br><br>`capacitor_list CapA,CapB,CapC`. |
| | | | List of end-of-line measurements for the `volt_var_control` to use in `regulator` control. This list is comma-separated, like the `capacitor_list` and `regulator_list`. If left blank, the measurement point defaults to load |

| | | | |
|---|---|---|---|
| voltage_measurements | char1024 | Objects (implied) | side of each `regulator` in `regulator_list`. The list can take two formats, depending on the number of regulators:<br>▪ One regulator - The list is a comma-separated list of just the object names. No regulator designation is necessary, since only one regulator is present. An example is<br>`measurement_list NodeA,NodeB,NodeC,NodeD;`.<br>▪ One or more regulators - The list is a comma-separated list of object names, and their associated regulator. All measurements must be represented as a pair in the list. The first item is the object name, and the second item is the `regulator` position in the `regulator_list` variable. For example,<br>`measurement_list NodeA,1,NodeC,2,NodeB,1;` will assign `NodeA` and `NodeB` to regulator 1, and `NodeC` to regulator 2. Notice these can occur in any regulator order, as long as the second portion of the data pair is the regulator number. |
| minimum_voltages | char1024 | Volts (implied) | List of minimum voltages for the `volt_var_control` object to maintain. If only 1 value is specified, this value will be used for all `regulators` in `regulator_list`. A value can be specified for each `regulator` as a comma-separated list. For example, to specify two minimum voltage levels for two different regulators, use `minimum_voltages 2600,2500;`. If left blank, the minimimum voltage is set at 0.95 p.u. of the `regulator`'s TO node `nominal_voltage`. |
| maximum_voltages | char1024 | Volts (implied) | List of maximum voltages for the `volt_var_control` object to maintain. If only 1 value is specified, this value will be used for all `regulators` in `regulator_list`. A value can be specified for each `regulator` as a comma-separated list. For example, to specify two maximum voltage levels for two different regulators, use `maximum_voltages 4500,4400;`. If left blank, the maximimum voltage is set at 1.05 p.u. of the `regulator`'s TO node `nominal_voltage`. |
| desired_voltages | char1024 | Volts (implied) | List of desired, or target voltages for the `volt_var_control` object to maintain. If only 1 value is specified, this value will be used for all `regulators` in `regulator_list`. A value can be specified for each `regulator` as a comma-separated list. For example, to specify two desired voltage levels for two different regulators, use `desired_voltages 4500,4400;`. If left blank, the desired voltage is |

| | | | |
|---|---|---|---|
| | | | set at the `regulator`'s TO node `nominal_voltage`. |
| max_vdrop | char1024 | Volts (implied) | List of voltage drop thresholds for high or low loading operation (selection of `high_load_deadband` or `low_load_deadband` for a regulator). If the voltage drop between the regulator and the lowest end-of-line measurement is greater than `max_vdrop`, the corresponding `high_load_deadband` is used. Otherwise, the corresponding `low_load_deadband` is used. `max_vdrop` can be a comma-separated list of values for each regulator of `regulator_list`. if only one value is specified, that value will be used for all regulators. If left blank, `max_vdrop` defaults to 1.5x the corresponding `regulator` object's step up tap voltage value. |
| high_load_deadband | char1024 | Volts (implied) | List of tap-changing bandwidth thresholds for high loading operation (as determined by `max_vdrop`). `high_load_deadband` represents a +/- `high_load_deadband` deadband around the desired voltage before a tap change is requested on the regulator. This can be specified for each regulator as a comma-separated list. If a single value is specified, that value will be used on all `regulators` in the `regulator_list`. If unspecified, `high_load_deadband` defaults to the voltage value associated with a single tap change on the `regulator`. |
| low_load_deadband | char1024 | Volts (implied) | List of tap-changing bandwidth thresholds for low loading operation (as determined by `max_vdrop`). `low_load_deadband` represents a +/- `low_load_deadband` deadband around the desired voltage before a tap change is requested on the regulator. This can be specified for each regulator as a comma-separated list. If a single value is specified, that value will be used on all `regulators` in the `regulator_list`. If unspecified, `low_load_deadband` defaults to the voltage value associated with a two tap change on the `regulator` (2x the default of `high_load_deadband`). |

### VoltVar Control State of Development

VoltVar Control is considered a well developed model, but has not been fully validated at this time. Advanced features and additional controls may be added as needed.

# Volt Dump

This object allows the user to collect all of the voltages in the system into one *.csv file at a given run time. Voltages are placed in the *.csv output file with format

| node_name, | voltA_real, | voltA_imag, | voltB_real, | voltB_imag, | voltC_real, | voltC_imag, |
|---|---|---|---|---|---|---|
| node_1, | 7200, | 0, | -3600, | -6235.4, | -3600, | 6235.4 |
| node_2, | 2400, | 0, | -1200, | -2078.5, | -1200, | 2078.5 |

### Default Volt Dump

The minimal amount of code to specify a `voltdump` object is

```
object voltdump {
      filename output_voltage.csv;
      }
```

which will produce an output file of the given name in the format shown above, and will display the voltage of every node in the glm file.

### Volt Dump Parameters

| Property Name | Type | Unit | Description |
|---|---|---|---|
| filename | char32 | N/A | Tells the object what file to print all information to. While a *.csv is not necessary, it is recommended as the formatted output is in *.csv format. |
| group | char32 | N/A | Using the `group_id` feature, this allows only nodes with the matching `group_id` to be dumped into the output file. |
| runtime | timestamp | N/A | Tells the object at what time to output the voltages of the system. Can be in either seconds from epoch (Unix time) or with a timestamp ('2006-01-01 00:00:00'). If not specified, the default is immediately after the first time step solution. |
| mode | enumeration | N/A | Allows the user to choose between polar and rectangular coordinates when printing output. Valid choices are<br>▪ `rect` rectangular coordinates<br>▪ `polar` polar coordinates (default - in radians) |

### Volt Dump State of Development

Volt Dump is considered a well developed, but unvalidated model. Additional features may be included as needed.

# Current Dump

This object allows the user to collect all of the currents in the system into one *.csv file at a given run time. In all cases, this is the current flowing INTO the link object (as defined by the to/from convention). Currents are placed in the *.csv output file with format:

| link_name, | currA_real, | currA_imag, | currB_real, | currB_imag, | currC_real, | currC_imag, |
|---|---|---|---|---|---|---|
| In rectangular | | | | | | |
| link_1, | 10, | 0, | -5, | -8.66, | -5, | 8.66 |
| Or in polar (radians) | | | | | | |
| link_2, | 20, | 0, | 20, | -2.0944, | 20, | 2.0944 |

### Default Current Dump

The minimal amount of code to specify a `currdump` object is

```
object currdump {
      filename output_current.csv;
      }
```

which will produce an output file of the given name in the format shown above, and will display the current of every link object in the glm file.

### Current Dump Parameters

| Property Name | Type | Unit | Description |
|---|---|---|---|
| filename | char32 | N/A | Tells the object what file to print all information to. While a *.csv is not necessary, it is recommended as the formatted output is in *.csv format. |
| group | char32 | N/A | Using the `group_id` feature, this allows only nodes with the matching `group_id` to be dumped into the output file. |
| runtime | timestamp | N/A | Tells the object at what time to output the currents of the system. Can be in either seconds from epoch (Unix time) or with a timestamp ('2006-01-01 00:00:00'). If not specified, the default is immediately after the first time step solution. |
| mode | enumeration | N/A | Allows the user to choose between polar and rectangular coordinates when printing output. Valid choices are<br>▪ `rect` rectangular coordinates (default)<br>▪ `polar` polar coordinates (in radians) |

### Current Dump State of Development

Current Dump is considered a well developed, but unvalidated model. Additional features may be included as needed.

# Bill Dump

Similar to `voltdump`, `billdump` allows users to generate a single file where all customers' bills are written from `triplex_meter` to a single output file in a similar format

| meter_name, | previous_monthly_bill, | previous_monthly_energy |
|---|---|---|
| triplex_meter_1, | 154.30 ($), | 1205 (kWh) |
| triplex_meter_2, | 105.10 ($), | 821 (kWh) |

### Default Bill Dump

The minimal specifications for billdump are

```
object billdump {
       filename bill_1.csv;
       }
```

where the previous month's energy and bill for all triplex meters within the system will be placed into bill_1.csv. Additional parameters can be added to describe when to run (runtime) and for only meters with a specific groupid:

```
object billdump {
    group "Residential_tm_solar";
    runtime '2012-04-01 01:00:00';
    filename residential_solar_bill.csv;
}
```

### Bill Dump Parameters

| Property Name | Type | Unit | Description |
|---|---|---|---|
| filename | char32 | N/A | Tells the object what file to print all information to. While a *.csv is not necessary, it is recommended as the formatted output is in *.csv format. |
| group | char32 | N/A | Using the `groupid` feature, this allows only triplex meters with the matching `groupid` to be dumped into the output file. If this is not specified, every triplex meter in the system will be recorded. |
| runtime | timestamp | N/A | Tells the object at what time to output the bills of the system. Can be in either seconds from epoch (Unix time) or with a timestamp ('2006-01-01 00:00:00'). If not specified, the default is immediately after the first time step solution. |

### Bill Dump State of Development

Bill Dump is considered a well developed, but unvalidated model. Additional features may be included as needed.

## Fault Check

The `fault_check` object performs "support checks" on objects inside the `powerflow` module. Its primary purpose is to determine if a particular node or link is still in service after a reconfiguration or fault event. `fault_check` is primarily utilized by the `reliability` module calls to powerflow, but is also called as part of the `restoration` object's functionality. The `fault_check` object only works with the `NR` `solver_method` at this time. Other solvers may be incorporated at a later date. A typical `fault_check` object would be implemented as

```
object fault_check {
	name fault_check_obj;
	check_mode ONCHANGE;
	output_filename outage_check.txt;
	eventgen_object Test_Evt_Obj;
	}
```

As with other objects, not all of the parameters need to be specified. A minimal implementation would be similar to

```
object fault_check {
	name fault_check_obj;
	}
```

### Fault Check Parameters

The `fault_check` object only derives the base `powerflow` module properties. However, these `powerflow`-based properties are not needed and are omitted from the list below.

| Property Name | Type | Unit | Description |
|---|---|---|---|
| check_mode | enumeration | N/A | Defines the fault checking scheme utilized. Valid entries are<br>▪ `SINGLE` - Fault checks and voltage support checks are only performed once, before the first solver pass of the `NR` solver.<br>▪ `ONCHANGE` - Fault checks and voltage support checks are performed any time an admittance change is flagged on the `NR` solver. |

| | | | |
|---|---|---|---|
| | | | ▪ `ALL` - Fault checks and voltage support checks are performed on every iteration |
| output_filename | char1024 | N/A | File name for the text file of support check status values. Will output the bus number and phases unsupported at each timestamp the `fault_check` obejct runs. |
| reliability_mode | bool | N/A | Boolean flag to indicate if the `fault_check` object is running in a `reliability`-module-based mode. `Reliability` will toggle this mode and it is only provided for user information, it is not a specifiable property. |
| eventgen_object | object | N/A | Object link to an `eventgen` object in the `reliability` module. This object will be used to implement any "unscheduled" faults, such as switch openings or fuses blowing. Without this object specified, such objects will still open or trip, but may result in an unsolvable system matrix and prematurely terminate the simulation. |

### Fault Check State of Development

The `fault_check` object has been developed in conjuction with the `reliability` module and the `restoration` object. It has been tested and it considered validated, but rigorous testing has not been conducted and additional features may be added at a future date.

## Frequency Generator

In Development.

### Frequency Generator Parameters

In Development.

### Frequency Generator State of Development

In Development.

## Motor

In Development.

### Motor Parameters

In Development.

### Motor State of Development

In Development.

## Restoration

As the `powerflow` module interacts with the `reliability` module, portions of the system may become isolated. The `restoration` object attempts to do feeder reconfiguration to close the isolated sections back into the system. The `restoration` object requires `reliability` or some reliability-like actions to function properly, as well as the `fault_check` object. The `restoration` object only works with the `NR` solver method at this time.

A `restoration` object can be implemented as:

```
object restoration {
        name RestorVal;
        reconfig_attempts 3;
        reconfig_iteration_limit 5;
        populate_tree TRUE;
        }
```

### Restoration Parameters

| Property Name | Type | Unit | Description |
|---|---|---|---|
| reconfig_attempts | double | Tries | Number of reconfiguration attempts a system will perform before giving up and determining the system can not be fully restored at that condition. |
| reconfig_interation_limit | double | count | Number of `powerflow` iterations a particular reconfiguration can try before failing. Used to prevent infinite iterating by the solver. |
| populate_tree | boolean | N/A | Flag to populate the tree structure of the feeder. Used for the algorithm implmeented to increase reconfiguration iterations and attempts. Should be set to `TRUE` whenever `reconfiguration` is used. |

### Restoration State of Development

The `restoration` object is considered highly experimental at this time.

## Series Reactor

The series reactor is a link object designed to model a series reactance on each of the three phases.

```
object series_reactor {
       from node1;
       to node2;
       phases ABC;
       phase_A_impedance 1+1j;
       phase_B_resistance 2;
       phase_C_reactance 3;
}
```

### Series Reactor Parameters

| Property Name | Type | Unit | Description |
|---|---|---|---|
| phase_A_impedance | double | Ohm | Series impedance on phase `A`. |
| phase_A_resistance | double | Ohm | Series resistance on phase `A`. Maps directly into phase_A_impedance, but allows user to specify real portion separately. |
| phase_A_impedance | double | Ohm | Series reactance on phase `A`. Maps directly into phase_A_impedance, but allows user to specify reactive portion separately. |
| phase_B_impedance | double | Ohm | Series impedance on phase `B`. |
| phase_B_resistance | double | Ohm | Series resistance on phase `B`. Maps directly into phase_B_impedance, but allows user to specify real portion separately. |
| phase_B_impedance | double | Ohm | Series reactance on phase `B`. Maps directly into phase_B_impedance, but allows user to specify reactive portion separately. |
| phase_C_impedance | double | Ohm | Series impedance on phase `C`. |

| | | | |
|---|---|---|---|
| phase_C_resistance | double | Ohm | Series resistance on phase `C`. Maps directly into phase_C_impedance, but allows user to specify real portion separately. |
| phase_C_impedance | double | Ohm | Series reactance on phase `C`. Maps directly into phase_C_impedance, but allows user to specify reactive portion separately. |
| rated_current_limit | double | Amps | Rated current limit for the reactor. Not used at this time. |

### Series Reactor State of Development

Series reactor has been tested, but not fully validated.

## Sectionalizer

`sectionalizer` objects provide a means to isolate faulted portions of a system. `sectionalizer` objects work in conjuction with the `reliability` module and the `recloser` objects. `reliability` will automatically open a `sectionalizer` if an upstream `recloser` is present, and has "tries" available. `sectionalizer` objects should work for both solver methods, but the `reliability` functionality only works in the `NR` solver.

A minimal `sectionalizer` implementation is:

```
object sectionalizer {
	name Test_Section;
	phases ABC;
	}
```

with an equivalent representation of:

```
object sectionalizer {
	name Test_Section;
	phases ABC;
	phase_A_state CLOSED;
	phase_B_state CLOSED;
	phase_C_state CLOSED;
	operating_mode BANKED;
	}
```

`sectionalizer` objects behave exactly like `switch` objects, aside from their `reliability` coordination. `sectionalizer` objects inherit all `switch` properties and default to a banked operation mode. No new parameters are introduced in sectionalizers.

### Sectionalizer State of Development

`sectionalizer` objects are based on `switch` objects and share a common state of development. NOrmal operation is tested and verified. `reliability`-based actions are validated, but not fully tested at this time.

## Power Metrics

The `power_metrics` object is used by the `reliability` module to calculate relevant `powerflow` metrics. The `power_metrics` object calculates the IEEE 1366-2003 metrics for evaluating the reliability indices of a power system.

A minimalist `power_metrics` implementation is

```
object power_metrics {
	name PwrMetrics;
	}
```

with an equivalent of

```
object power_metrics {
        name PwrMetrics;
        base_time_value 60.0;
        }
```

`power_metrics` objects are primarily output objects.

### Power Metrics Parameters

`power_metrics` objects do not inherit properties from any module. Individual metrics are described in the `reliability` user's guide and in the IEEE 1366-2003 standard.

| Property Name | Type | Unit | Description |
|---|---|---|---|
| SAIDI | double | Customer interruption duration over total customers served | The simulation-long computed value of the System Average Interruption Duration Index |
| SAIDI_int | double | Customer interruption duration over total customers served | The interval-long computed value of the System Average Interruption Duration Index. The interval is defined by the `base_time_Value` property. |
| SAIFI | double | Customers interrupted over customers served | The simulation-long computed value of the System Average Interruption Frequency Index |
| SAIFI_int | double | Customers interrupted over customers served | The interval-long computed value of the System Average Interruption Frequency Index. The interval is defined by the `base_time_Value` property. |
| ASAI | double | Customer hours availability over customer hours demand | The simulation-long computed value of the Average Service Availability Index. |
| ASAI_int | double | Customer hours availability over customer hours demand | The interval-long computed value of the Average Service Availability Index. The interval is defined by the `base_time_Value` property. |
| CAIDI | double | Customer interruption duration over total customer interrupted | The simulation-long computed value of the Customer Average Interruption Duration Index. |
| | | Customer | |

| | | | |
|---|---|---|---|
| CAIDI_int | double | interruption duration over total customer interrupted | The interval-long computed value of the Customer Average Interruption Duration Index. The interval is defined by the `base_time_Value` property. |
| MAIFI | double | Customer momentary interruptions over total customers served | The simulation-long computed value of the Momentary Average Interruption Frequency Index |
| MAIFI_int | double | Customer momentary interruptions over total customers served | The interval-long computed value of the Momentary Average Interruption Frequency Index. The interval is defined by the `base_time_Value` property. |
| base_time_value | double | seconds | Interval duration for IEEE 1366-2003 statistics to be computed. This information is the basis for any time calculations. For example, the interruption duration for a CAIDI calculation can be interruptions per hour, interruptions per minute, or any other time base. `base_time_value` dictates this base for the calculations. The value defaults to 1 minute. |

### Power Metrics State of Development

The `power_metrics` object is tested and validated with the `reliability` module. However, it has not been fully validated and is considered experimental at this time.

## Emissions

In Development.

### Emissions Parameters

In Development.

### Emissions State of Development

In Development.

# See also

Powerflow

Retrieved from "http://gridlab-d.sourceforge.net/wiki/index.php?title=Power_Flow_User_Guide&oldid=8357"

- This page was last modified on 7 July 2016, at 21:21.