GRG-Schema v4.0

The GRG Team:
Russell Bent, Carleton Coffrin, Ferdinando Fioretto,
Terrence W.K. Mak, Patrick Panciatici, Pascal Van Hentenryck [*]

May, 2018

Abstract

This document describes the *Grid Research for Good* (GRG) schema to represent transmission networks. Time series data, stochastic time series data, and contingencies will also be discussed.

1 Basic Concepts

This section defines the basic concepts used to model grid data. We first define data types for representing numerical values, variables, and object statuses. Then we define data types used to represent electrical values.

Throughout the document we adopt the symbol j to denote the complex imaginary unit, which satisfies the equation: $j^2 = -1$. For a complex number $x \in \mathbb{C}$ we write $\bar{x}$ to denote the conjugate of x. Additionally, for a list of elements $A = \langle a_1, \ldots, a_n \rangle$, we use the notation $A[k]$ to indicate the k^{th} element of A, a_k.

For a JSON attribute, the keyword `$ref` describes a *JSON Reference*, i.e., a reference to an object whose fragment part is a URI encoded JSON Pointer. In this document, references are local to the schema, and their values reflect the document hierarchy. To allow ease of format extension in the future and ease of format conversion between GRG and other file formats in the power system community, we set 'additionalProperties' to 'true' in our schema to allow users define additional properties.

1.1 Values

Extended Number

An *extended number* is a numeric data type whose value is either a number or one of the following: *Inf*, denoting ∞, *-Inf* denoting $-\infty$, *NaN*, standing for "not a number" and representing an undefined or unrepresentable value, or *Null*, to denote a value which is missing.

extended_number definition
```
"extended_number": {
    "oneOf": [
        {"type": "number"},
        {"enum": ["Inf", "-Inf", "NaN", "Null"]}
    ]
}
```

The following example illustrates the above concept by where *x* is assigned a numeric value, and *y* to a $-\infty$.

Example: extended_number
```
"x": 10.23,
"y": "-Inf"
```

[*]**Contacts:** Terrence W.K. Mak: wmak@umich.edu, Pascal Van Hentenryck: pvanhent@umich.edu

Extended Positive Number

An *extended positive number* restricts the notion of extended number to non-negative values and the special values *Inf*, *NaN*, and *Null*.

```
extended_positive_number definition
"extended_positive_number": {
    "oneOf": [
        {"type": "number", "minimum": 0},
        {"enum": ["Inf", "NaN", "Null"]}
    ]
}
```

Domain

A *domain* is a collection of values used to describe valid assignments for a variable. We define three types of domains:

- Finite domains describing a collection of strings.
- Finite domains describing a collection of numbers.
- Bound domains describing a range $[lb, ub] \subseteq \mathbb{R}$.

```
GRG schema: domain
"domain": {
    "type"    : "object",
    "required": ["var"],
    "properties": {
        "var": {
            "oneOf": [
            {
                "type"    : "array",
                "items"   : {"type": "string"},
                "minItems": 1
            }, {
                "type"    : "array",
                "items"   : {"type": "number"},
                "minItems": 1
            }, {
                "type"    : "object",
                "required": ["lb", "ub"],
                "properties": {
                    "lb"  : {"$ref": "#/values/extended_number"},
                    "ub"  : {"$ref": "#/values/extended_number"}
                },
                "additionalProperties": true
            }]
        }
    },
    "additionalProperties": true
}
```

Following are examples of finite string domain (x), finite numerical domain (y), and bound domain (z).

```
Example: domain
"x": { "var": ["a", "b", "c"] },
"y": { "var": [1, 2, 3] },
"z": { "var": {"lb": 0, "ub": 10} }
```

Positive Domain

A *positive domain* restricts the notion of domain to non-negative values.

```
GRG schema: positive_domain
"positive_domain": {
    "type"    : "object",
    "required": ["var"],
    "properties": {
        "var": {
            "oneOf": [
            {
                "type"    : "array",
                "items"   : {"type": "string"},
                "minItems": 1
            }, {
                "type"    : "array",
                "items"   : {"type": "number", "minimum": 0},
                "minItems": 1
            }, {
                "type"    : "object",
                "required": ["lb", "ub"],
                "properties": {
                    "lb"  : {"$ref": "#/values/extended_positive_number"},
                    "ub"  : {"$ref": "#/values/extended_positive_number"}
                },
                "additionalProperties": true
            }]
        }
    },
    "additionalProperties": true
}
```

Abstract Value

An *abstract value* is an extended numeric data type which can describe either a numeric value or a variable.

GRG schema: abstract_value

```
"abstract_value": {
    "oneOf": [
        {"$ref": "#/values/extended_number"},
        {"$ref": "#/values/domain"}
    ]
}
```

In the following code, `x` is a variable abstract value, and `y` is a numeric abstract value.

Example: abstract_value

```
"x": { "var": {"lb": 0, "ub": 10} },
"y": 3.56
```

Abstract Positive Value

An *abstract positive value* restricts the notion of *abstract value* to non-negative numeric values and non negative variables.

GRG schema: abstract_positive_value

```
"abstract_value": {
    "oneOf": [
        {"$ref": "#/values/extended_positive_number"},
        {"$ref": "#/values/positive_domain"}
    ]
}
```

Status

A *status* is a special boolean variable whose domain elements are “on” and “off”.

GRG schema: abstract_status

```
"status": {
    "oneOf": [
        {"enum"       : ["off", "on"]},
        {
            "type"    : "object",
            "required": ["var"],

            "properties": {
                "var": {
                    "type"    : "array",
                    "items"   : {"enum": ["on", "off"]},
                    "minItems": 2, "maxItems": 2, "uniqueItems": true
                }
            },
            "additionalProperties": true
        }
    ]
}
```

In the following examples, `x` represents an unassigned status variable, and `y` represents a status element whose value is 'on'.

Example: status

```
"x": { "var": ["on", "off"] }
"y": "on",
```

GRG Pointer

A *GRG pointer* is a string used to identify an object's value in the GRG document. A GRG pointer extends a JSON pointer by adopting the following prefixes:

- `#`, which refers to the document root.
- `@`, which refers to a JSON object in the same scope as the pointer itself.

If the pointer does not start with either `#` or `@`, we assume the pointer refers to a component by its unique ID.

GRG schema: grg_pointer

```
"grg_pointer": {
    "type": "string",
    "pattern": ".*"
}
```

In the following examples, the GRG pointers `p1`, `p2`, and `p3` refer to the same value (`voltage_id_ALH_2`). The GRG pointer `p3` uses the global GRG id of the component object for referencing. In GRG format, we assume all GRG ids (`id`) are global and unique for all the components (i.e. GRG ids are global identifiers of components). The pointer `p4` refers to the first value of the array `var` in the object `status` of `switch_example`.

Example: grg_pointer

```
"network: {
    "components": {
        "switch_example" : {
            "type"    : "switch",
            "subtype" : "breaker",
            "id"      : "sw_722",
            "link_1"  : "voltage_id_ALH_2",
            "link_2"  : "voltage_id_ALH_3",
            "status"  : {"var" : ["off", "on"]},
            "p1"      : "@/link_1"
        }
    }
},

"p2": "#/network/components/switch_example/link_1",
"p3": "sw_722/link_1",
"p4": "sw_722/status/var/0
```

Table

A *table* is an object linking a list of GRG elements $\langle e_1, \ldots, e_k \rangle$ to a set of tuples $\{T_1, \ldots, T_n\}$, where each $T_i = \langle v_1, \ldots, v_k \rangle$ has values v_i (i = 1, …, k). The elements are referred to as table *arguments*, and the set of value tuples as table *values*. In other words, a table expresses the relation between a list of elements and the set of possible values for such elements.

GRG schema: table

```
"table": {
    "type"    : "object",
    "required": ["arguments", "values"],
    "properties": {
        "arguments": {
            "type"    : "array",
            "items"   : {"$ref": "#/values/grg_pointer"},
            "minItems": 1, "uniqueItems": true
        },
        "values": {
            "type" : "array":
            "items": {
                "type" : "array",
                "items": {
                    "oneOf": [
                        {"type": "#/values/abstract_value"},
                        {"type": "string"}
                    ]
                },
                "minItems": 1
            },
            "minItems": 1
        }
    }
}
```

The following example describes three assignments for the elements x, y, and z:

$$\mathrm{x} = 1,\ \mathrm{y} = 12,\ \mathrm{z} = 13$$
$$\mathrm{x} = 2,\ \mathrm{y} = 6,\ \mathrm{z} = 20$$
$$\mathrm{x} = 3,\ \mathrm{y} = 10,\ \mathrm{z} = 31$$

Example: table

```
"table_1" : {
    "arguments":  ["#/x", "#/y", "#/z"],
    "values"   : [[ 1,   12,  13 ],
                  [ 2,   6,   20 ],
                  [ 3,   10,  31 ]]
}
```

1.2 Electrical Values

This section defines the electrical values adopted by the GRG schema.

Impedance

Electrical *impedance* measures the opposition of a circuit to a current when a given voltage is applied. Impedance is represented as a complex quantity Z:

$$Z = R + jX, \tag{1}$$

where R denotes the resistance, and X the reactance.

Table 1 maps the real and imaginary components of impedance to their GRG schema counterparts.

GRG name	Symbol	Unit
`impedance`	Z	Ohm (Ω)
`resistance`	R	Ohm (Ω)
`reactance`	X	Ohm (Ω)

Table 1: Impedance: representation in the GRG `impedance` element and units.

GRG schema: impedance

```
"impedance": {
    "type"                : "object",
    "required"            : ["resistance", "reactance"],
    "additionalProperties": true,
    "properties": {
        "resistance": {"$ref": "#/values/abstract_value"},
        "reactance" : {"$ref": "#/values/abstract_value"}
    }
}
```

Though the schema definition may seem complicated, the following example shows how simple it is to use:

Example: impedance

```
"impedance" : {
    "reactance" : 6.52,
    "resistance" : 2.39
}
```

Admittance

Electrical *admittance* is a measure of how much a circuit allows current to flow. Admittance is represented by a complex quantity Y:

$$Y = G + jB, \tag{2}$$

where G denotes conductance, and B denotes susceptance. These real and imaginary components are mapped to the GRG `admittance` object as shown in Table 2.

GRG name	Symbol	Unit
`admittance`	Y	Siemens (S)
`conductance`	G	Siemens (S)
`susceptance`	B	Siemens (S)

Table 2: Admittance: representation in the GRG `admittance` element and units.

GRG schema: admittance

```
"admittance": {
    "type"                : "object",
    "required"            : ["conductance", "susceptance"],
    "additionalProperties": true,
    "properties": {
        "conductance"     : {"$ref": "#/values/abstract_value"},
        "susceptance"     : {"$ref": "#/values/abstract_value"}
    }
}
```

The following example shows how to define an admittance in GRG format:

Example: admittance

```
"shunt" : {
    "conductance" : 0,
    "susceptance" : 2.3e-05
}
```

Power

Electric power is the rate at which electrical energy is transferred by an electric circuit. We define the complex power S as:

$$S = P + jQ, \tag{3}$$

where P is the active (or real) power, and Q is the reactive power. Table 3 shows how these quantities are encoded in the GRG format.

GRG name	Symbol	Unit
`power`	S	
`active`	P	MegaWatt (*MW*)
`reactive`	Q	MegaVolt-Ampere Reactive (*MVAR*)

Table 3: Power: representation in the GRG format and units.

```
GRG schema: power
"power": {
    "type"                : "object",
    "required"            : ["active", "reactive"],
    "additionalProperties": true,
    "properties": {
        "active"          : {"$ref": "#/values/abstract_value"},
        "reactive"        : {"$ref": "#/values/abstract_value"}
    }
}
```

Active and reactive power are defined as variables with bound domains in the following example:

```
Example: power
"power" : {
    "active"   : { "var" {"lb": 0, "ub": 30.0} },
    "reactive" : { "var" {"lb": -14.0, "ub": 14.0} }
}
```

Voltage

Voltage is the difference in electric potential energy between two points per unit electric charge. We define the voltage phasor V as:

$$V = v \cdot e^{j\theta}, \tag{4}$$

where v is the voltage magnitude, and θ is the voltage phase angle. Table 4 connects the phasor components and units to their GRG representations.

GRG name	Symbol	Unit
`voltage`	V	
`magnitude`	v	kiloVolt (kV)
`angle`	θ	Degrees

Table 4: Voltage: representation in the GRG format and units.

```
GRG schema: voltage
"voltage": {
    "type"                : "object",
    "required"            : ["magnitude", "angle"],
    "additionalProperties": true,
    "properties": {
        "magnitude"       : {"$ref": "#/values/abstract_value"},
        "angle"           : {"$ref": "#/values/abstract_value"}
    }
}
```

In the following example, magnitude and phase angle are defined as variables:

```
Example: voltage
"voltage" : {
    "magnitude" : { "var" {"lb": 210, "ub": 250.0} },
    "angle"     : { "var" {"lb": "-Inf", "ub": "Inf"} }
}
```

1.3 Limits

Current Limits

Transmission lines (see section AC Line) and transformers (see section Transformers) have limited current-carrying capacity, described as ranges of current values $[I^{min}, I^{max})$ that may be sustained for a duration d.

A current limit is represented in GRG format according to Table 5.

GRG name	Symbol	Unit
`min`	I^{min}	Ampere (A)
`max`	I^{max}	Ampere (A)
`duration`	d	Seconds (sec)

Table 5: Current Limits: representation in the GRG format and units.

A current limit object is expressed as an *array* of individual limits. Each individual limit is a JSON object containing at least four fields: `duration`, `min`, `max`, and `report`. The extra `report` field is a boolean field, indicating whether the status of the branch should be signaled to the operator ('on') or not ('off').

```
GRG schema: current_limits
"current_limits": {
    "type": "array",
    "items" : [{
                  "type": "object",
                  "required": ["duration", "min", "max", "report"],
                  "additionalProperties": true,
                  "properties": {
                    "duration": {"$ref": "#/values/basic_values/extended_number"},
                    "min": {"$ref": "#/values/basic_values/extended_number"},
                    "max": {"$ref": "#/values/basic_values/extended_number"},
                    "report": {"enum": ["on","off"]}
                  }
                }],
    "minItems": 1,
    "additionalItems": false
}
```

An example of current limits is provided below. The branch can (1) carry up to 563 A indefinitely, or (2) carry between 563 A and 746 A indefinitely, but with a signal to the system operators, or (3) carry current in excess of 746 A for at most 6300 seconds before tripping the branch.

```
Example: current_limits
"current_limits" : [
  {"duration": "Inf",  "min": 0, "max": 563, "report": "off"},
  {"duration": "Inf", "min": 563, "max": 746, "report": "on"},
  {"duration": 6300,  "min": 746, "max": "Inf", "report": "off"}
]
```

Thermal Limits

Transmission line and transformer limits can also be described in terms of power $[P^{min}, P^{max})$ that may be safely carried for a given duration d. The GRG representation of a thermal limit is provided in Table 6.

Similar to the current limit, a thermal limit object is also expressed as an *array* of individual limits. Each individual limit is again a JSON object containing at least four fields: `duration`, `min`, `max`, and `report`.

GRG name	Symbol	Unit
`min`	S^{min}	MegaWatt (*MW*)
`max`	S^{max}	MegaWatt (*MW*)
`duration`	d	Seconds (*sec*)

Table 6: Thermal Limits: representation in the GRG format and units.

GRG schema: thermal_limits

```
"thermal_limits": {
    "type": "array",
    "items" : [{
                 "type": "object",
                 "required": ["duration", "min", "max", "report"],
                 "additionalProperties": true,
                 "properties": {
                   "duration": {"$ref": "#/values/basic_values/extended_number"},
                   "min": {"$ref": "#/values/basic_values/extended_number"},
                   "max": {"$ref": "#/values/basic_values/extended_number"},
                   "report": {"enum": ["on","off"]}
               }
               }],
    "minItems": 1,
    "additionalItems": false
}
```

2 Network Components

Having described the fundamental parameters of electrical devices like transmission lines, transformers, and generators, we now describe these components themselves.

2.1 AC Line

An AC line connects network devices in two different points of the network. It has two sides: *side 1* is defined as the sending side, and *side 2* is defined as the receiving side. For simplicity, we assume the current is positive when flowing from side 1 to side 2, i.e. ”left to right”. The currents I_1 (at side 1) and I_2 (at side 2) are given by:

$$I_1 = Y_1 \cdot V_1 + \frac{1}{Z}(V_1 - V_2) \tag{5}$$

$$I_2 = -Y_2 \cdot V_2 + \frac{1}{Z}(V_1 - V_2), \tag{6}$$

where $Y_1 = G_1 + jB_1$ and $Y_2 = G_2 + jB_2$ are the admittances at sides 1 and 2, respectively, $Z = R + jX$ is the line impedance, and V_1 and V_2 are the voltages at the connecting points. Figure 1 illustrates an AC line between nodes 1 and 2 (represented by the black points at the ends). The power S_i at side i ($i \in \{1,2\}$) is given by:

$$S_i = \bar{I}_i \cdot V_i \tag{7}$$

Note that many softwares/implementations may assume a different current/power flow directions. Users will need to flip the sign of the current variables or power variables if the direction is changed from ”left to right” to ”right to left”.

Line current/thermal limits are monitored at both ends. For current limits, the absolute value of current magnitude is compared to a sequence of current limits $L_1 = L_{1_1}, \ldots, L_{1_n}$, and $L_2 = L_{2_1}, \ldots, L_{2_n}$, where each L_{i_k} ($i \in \{1,2\}, k \in \{1,\ldots,n\}$) is a current limit object, and for each L_{i_k}, ($k > 1$), $I^{\min}_{i_k} = I^{max}_{i_{k-1}}$. Thermal limits are similar. The absolute value of complex power magnitude is compared to a sequence of thermal limits $L_1 = L_{1_1}, \ldots, L_{1_n}$, and $L_2 = L_{2_1}, \ldots, L_{2_n}$, where each L_{i_k} ($i \in \{1,2\}, k \in \{1,\ldots,n\}$) is a thermal limit object, and for each L_{i_k}, ($k > 1$), $S^{\min}_{i_k} = S^{\max}_{i_{k-1}}$. Finally, the current/thermal limit durations are such that $d_{i_1} = \infty$, denoting that L_{i_1} is a permanent acceptable limit.

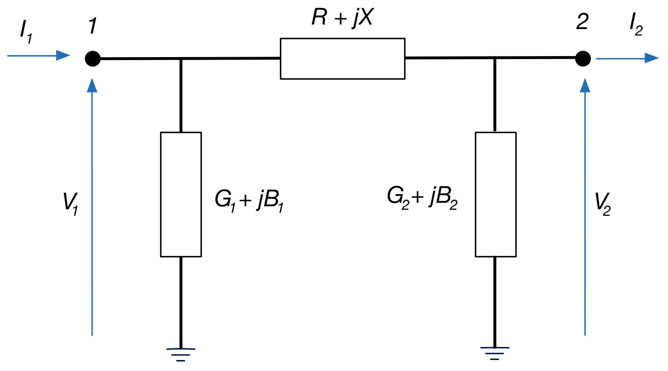

Figure 1: Illustration of an AC line.

GRG schema: ac_line

```
"ac_line": {
    "type"    : "object",
    "required": ["type", "id", "link_1", "link_2",  "shunt_1", "shunt_2", "impedance"],
   "additionalProperties": true,
    "properties": {
        "type"               : {"enum": ["ac_line"]},
        "subtype"                   : {"type": "string"},
        "id"                 : {"type": "string"},
        "description"        : {"type": "string"},
        "link_1"             : {"type": "string"},
        "link_2"             : {"type": "string"},
        "shunt_1"            : {"$ref": "#/electrical_values/admittance"},
        "shunt_2"            : {"$ref": "#/electrical_values/admittance"},
        "impedance"          : {"$ref": "#/electrical_values/impedance"},
        "current_limits_1"   : {"$ref": "#/limits/current_limits"},
        "current_limits_2"   : {"$ref": "#/limits/current_limits"},
        "thermal_limits_1"   : {"$ref": "#/limits/thermal_limits"},
        "thermal_limits_2"   : {"$ref": "#/limits/thermal_limits"}
    }
}
```

For a component object in the GRG schema, we use `type` and `subtype` (optional) to identify the type of an object (for example an *AC line*). `id` is a unique identifier for global referencing, and `description` (optional) is further used to describe the component. Fields `link_1` and `link_2` are global identifiers for identifying the voltage points being connected to the network component. For an AC line, `link_1` and `link_2` will be the voltage points at side 1 and side 2 respectively. `shunt_1` and `shunt_2` define the admittances Y_1 and Y_2 at sides 1 and 2, respectively. `impedance` defines the impedance Z of the line. `current_limits_1` and `current_limits_2` describe collections of current limits associated to sides 1 and 2 of the line. Finally, `thermal_limits_1` and `thermal_limits_2` describe collections of thermal limits associated to sides 1 and 2 of the line. Thermal limits are optional. Starting from GRGv1.6, current limits are also optional. A summary of these AC line components is provided in Table 7.
An example of an AC line in GRG format is provided in Figure 2.

2.2 DC Line

In GRGv4.0, we start to provide support for High-Voltage DC (HVDC) lines. A DC line connects network devices at two different points of the network, with an AC-DC converter on each side. The converter is called rectifier if set to convert power from AC to DC, and will be called inverter if set to convert power from DC to AC. Power or current flowing through the DC line are first converted from AC to DC by a rectifier, before devliering power back to the network in AC by an inverter.

DC line again has two sides. We define *side 1* as the sending side, and *side 2* as the receiving side. For simplicity, we again assume the current is positive when flowing from side 1 to side 2, i.e. ”left to right”. In other words, we implicitly assume side 1 will be connecting to a rectifier, and side 2 will be connecting to an inverter. The DC voltages

GRG name	symbol	unit
link_1	side 1	
link_2	side 2	
impedance→resistance	R	Ohm (Ω)
impedance→reactance	X	Ohm (Ω)
shunt_1→conductance	G_1	Siemens (S)
shunt_1→susceptance	B_1	Siemens (S)
shunt_2→conductance	G_2	Siemens (S)
shunt_2→susceptance	B_2	Siemens (S)
current_limits_1[k]→duration	d_{1_k}	Seconds (sec)
current_limits_1[k]→min	$I_{1_k}^{min}$	Ampere (A)
current_limits_1[k]→max	$I_{1_k}^{max}$	Ampere (A)
current_limits_2[k]→duration	d_{2_k}	Seconds (sec)
current_limits_2[k]→min	$I_{2_k}^{min}$	Ampere (A)
current_limits_2[k]→max	$I_{2_k}^{max}$	Ampere (A)
thermal_limits_1[k]→duration	d_{1_k}	Seconds (sec)
thermal_limits_1[k]→min	$S_{1_k}^{min}$	MegaWatt (MW)
thermal_limits_1[k]→max	$S_{1_k}^{max}$	MegaWatt (MW)
thermal_limits_2[k]→duration	d_{2_k}	Seconds (sec)
thermal_limits_2[k]→min	$S_{2_k}^{min}$	MegaWatt (MW)
thermal_limits_2[k]→max	$S_{2_k}^{max}$	MegaWatt (MW)

Table 7: AC line: representation in the GRG format and units.

Example: AC Line

```
"ac_line_11" : {
    "type"               : "ac_line",
    "subtype"              : "overhead",
    "id"                 : "line_11",
    "link_1"             : "voltage_id_ALH_5",
    "link_2"             : "voltage_id_OQF_5",
    "current_limits_1" : [
      {"duration": "Inf",  "min": 0, "max": 563, "report": "off"},
      {"duration": "Inf", "min": 563, "max": 746, "report": "on"},
      {"duration": 6300,  "min": 746, "max": "Inf", "report": "off"}
    ],
    "current_limits_2" : [
      {"duration": "Inf",  "min": 0, "max": 563, "report": "off"}
    ],
    "impedance"     : {"reactance" : 6.52, "resistance" : 2.39},
    "shunt_1"       : {"conductance" : 0, "susceptance" : 2.3e-05},
    "shunt_2"       : {"conductance" : 0, "susceptance" : 2.3e-05}
}
```

Figure 2: An example of a GRG AC Line.

V_1 (at side 1) and V_2 (at side 2) of a DC line are linked by:

$$V_1 = V_2 + RI \tag{8}$$
(9)

where R is the DC line resistance, and I is the DC current flowing through the DC line. The DC power S_i at side i ($i \in \{1, 2\}$) is given by:

$$S_i = I \cdot V_i \tag{10}$$

Again, many softwares/implementations may assume a different current/power flow directions. Users will need to flip the sign of the current/power variables if the direction is changed from ”left to right” to ”right to left”.

In GRG, we also support modeling of active power loss P_{loss} for the converters by:

$$P_{\text{loss}} = \min(\max(l_{\text{min}}, |c_1 P + c_0|), l_{\text{max}}) \tag{11}$$

where P is the active power flowing through the converter, l_{min} and l_{max} are the minimum and maximum losses of the converter, and c_1/c_0 are the coefficients of the converter loss function. Detailed modeling of the converter control mechanism (e.g. VSC or LCC controls) can also be added to the JSON object as extensions.

Figure 3 illustrates a DC line connecting to two converters.

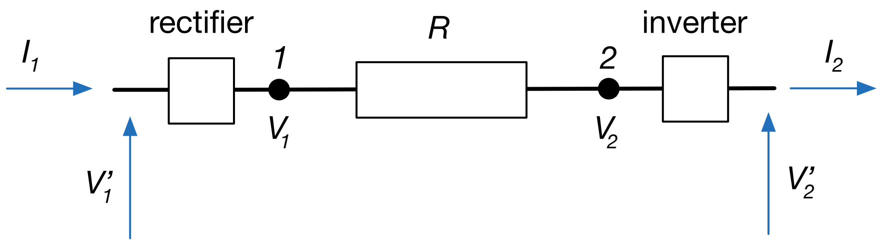

Figure 3: Illustration of a DC line.

Similar to AC lines, line current/thermal limits are monitored on DC lines at both ends. For current limits, the absolute value of current magnitude is compared to a sequence of current limits $L_1 = L_{1_1}, \ldots, L_{1_n}$, and $L_2 = L_{2_1}, \ldots, L_{2_n}$, where each L_{i_k} $(i \in \{1, 2\}, k \in \{1, \ldots, n\})$ is a current limit object, and for each L_{i_k}, $(k > 1)$, $I^{min}_{i_k} = I^{max}_{i_{k-1}}$. Thermal limits are similar. The absolute value of complex power magnitude is compared to a sequence of thermal limits $L_1 = L_{1_1}, \ldots, L_{1_n}$, and $L_2 = L_{2_1}, \ldots, L_{2_n}$, where each L_{i_k} $(i \in \{1, 2\}, k \in \{1, \ldots, n\})$ is a thermal limit object, and for each L_{i_k}, $(k > 1)$, $S^{\text{min}}_{i_k} = S^{max}_{i_{k-1}}$. Finally, the current/thermal limit durations are such that $d_{i_1} = \infty$, denoting that L_{i_1} is a permanent acceptable limit.

GRG schema: dc_line

```
"dc_line":{
    "type": "object",
    "required": ["type", "id", "link_1", "link_2", "resistance",
                  "losses_1", "losses_2" ],
    "additionalProperties": true,
    "properties": {
        "type": { "enum": ["dc_line", "dc_line_vsc", "dc_line_lcc"]},
        "subtype": {"type": "string"},
        "id": { "type": "string"},
        "description": {"type": "string"},
        "link_1": {"type": "string"},
        "link_2": {"type": "string"},
        "resistance": {"$ref": "#/values/basic_values/abstract_value"},
        "losses_1": {
            "type": "object",
            "required": ["c_0", "c_1"],
            "additionalProperties": true,
            "properties": {
                "min" : {"$ref": "#/values/basic_values/abstract_value"},
                "max" : {"$ref": "#/values/basic_values/abstract_value"},
                "c_0" : {"$ref": "#/values/basic_values/abstract_value"},
                "c_1" : {"$ref": "#/values/basic_values/abstract_value"}
            }
        },
        "losses_2": {
            "type": "object",
            "required": ["c_0", "c_1"],
            "additionalProperties": true,
            "properties": {
                "min" : {"$ref": "#/values/basic_values/abstract_value"},
                "max" : {"$ref": "#/values/basic_values/abstract_value"},
                "c_0" : {"$ref": "#/values/basic_values/abstract_value"},
                "c_1" : {"$ref": "#/values/basic_values/abstract_value"}
            }
        },
        "output_1" : {
            "reactive": { "$ref": "#/values/electrical_values/abstract_value" }
        },
        "output_2" : {
            "reactive": { "$ref": "#/values/electrical_values/abstract_value" }
        },
        "current_limits_1": {"$ref": "#/values/limits/current_limits"},
        "current_limits_2": {"$ref": "#/values/limits/current_limits"},
        "thermal_limits_1": {"$ref": "#/values/limits/thermal_limits"},
        "thermal_limits_2": {"$ref": "#/values/limits/thermal_limits"}
    }
}
```

For a component object in the GRG schema, we use `type` and `subtype` (optional) to identify the type of an object (for example a *DC line*). `id` is a unique identifier for global referencing, and `description` (optional) is further used to describe the component. Similar to AC lines, fields `link_1` and `link_2` are again global identifiers for identifying the voltage points being connected to the network component. For a DC line, `link_1` and `link_2` will be the voltage points of its converters at side 1 and side 2 respectively.

`resistance` define the resistance R for the DC line. `losses_1` and `losses_2` give the power losses for the converters at side 1 and 2 respectively. The two JSON objects are required to provide the loss function coefficients `c_0` and `c_1`. `min` and `max` are optional fields denoting the minimum and maximum active power losses. `output_1` and `output_2` are optional fields to denote the feasible range of the reactive power at side 1 and 2 respectively. Feasible active power range are also allowed to be extended and inserted into the `output_1` / `output_2` JSON objects. Similar to AC line, `current_limits_1` and `current_limits_2` describe collections of current limits associated to sides 1 and 2 of the line, and `thermal_limits_1` and `thermal_limits_2` describe collections of thermal limits associated to sides 1 and 2 of the line. Thermal limits and current limits are both optional. A summary of these DC line components is provided in Table 8.
An example of a DC line in GRG format is provided in Figure 4.

```
Example: DC Line
"dc_line_01": {
      "type": "dc_line_vsc",
      "id": "dc_line_1",
      "link_1": "voltage_id_30",
      "link_2": "voltage_id_31",
      "resistance": 0.0192,
      "losses_1": { "min": 0.01, "c_0": 0.02, "c_1": 0.03 },
      "losses_2": { "min": 0.01, "c_0": 0.02, "c_1": 0.03 },
      "output_1": { "reactive": { "var": {"lb": -0.4, "ub": 0.4} } },
      "output_2": { "reactive": { "var": {"lb": -0.4, "ub": 0.4} } },
      "current_limits_1" : [
         {"duration": "Inf",  "min": 0, "max": 600, "report": "off"}
      ],
      "current_limits_2" : [
         {"duration": "Inf",  "min": 0, "max": 600, "report": "off"}
      ]
}
```

Figure 4: An example of a GRG DC Line.

2.3 Transformers

A two winding transformer connects devices located at two different voltage levels of the network. We denote these voltage levels VL_1 and VL_2, and their associated nominal voltage values v_1^{nom} and v_2^{nom}. Most transformers are equipped with taps on their winding to adjust the voltage transformation or the reactive flow through the transformer. For a specific tap k, we denote the voltage magnitude ratio for a transformer to be r_k, phase shift to be δ_k, impedance to be Z_k, and admittance to be Y_k. The current GRG schema supports to two circuit diagram for representing a two

GRG name	symbol	unit
`link_1`	side 1	
`link_2`	side 2	
`resistance`	R	Ohm (Ω)
`losses_1→min`	l_{min}	MegaWatt (MW)
`losses_1→max`	l_{max}	MegaWatt (MW)
`losses_1→c_0`	c_0	MegaWatt (MW)
`losses_1→c_1`	c_1	MegaWatt (MW)
`losses_2→min`	l_{min}	MegaWatt (MW)
`losses_2→max`	l_{max}	MegaWatt (MW)
`losses_2→c_0`	c_0	MegaWatt (MW)
`losses_2→c_1`	c_1	MegaWatt (MW)
`output_1→reactive`	Q	MegaVolt-Ampere Reactive ($MVAR$)
`output_2→reactive`	Q	MegaVolt-Ampere Reactive ($MVAR$)
`current_limits_1[k]→duration`	d_{1_k}	Seconds (sec)
`current_limits_1[k]→min`	$I_{1_k}^{min}$	Ampere (A)
`current_limits_1[k]→max`	$I_{1_k}^{max}$	Ampere (A)
`current_limits_2[k]→duration`	d_{2_k}	Seconds (sec)
`current_limits_2[k]→min`	$I_{2_k}^{min}$	Ampere (A)
`current_limits_2[k]→max`	$I_{2_k}^{max}$	Ampere (A)
`thermal_limits_1[k]→duration`	d_{1_k}	Seconds (sec)
`thermal_limits_1[k]→min`	$S_{1_k}^{min}$	MegaWatt (MW)
`thermal_limits_1[k]→max`	$S_{1_k}^{max}$	MegaWatt (MW)
`thermal_limits_2[k]→duration`	d_{2_k}	Seconds (sec)
`thermal_limits_2[k]→min`	$S_{2_k}^{min}$	MegaWatt (MW)
`thermal_limits_2[k]→max`	$S_{2_k}^{max}$	MegaWatt (MW)

Table 8: DC line: representation in the GRG format and units.

winding transformer: the T model (Figure 5) and the PI model (Figure 6). We describe the T model first, then later the PI model.

Two Winding Transformer: T model

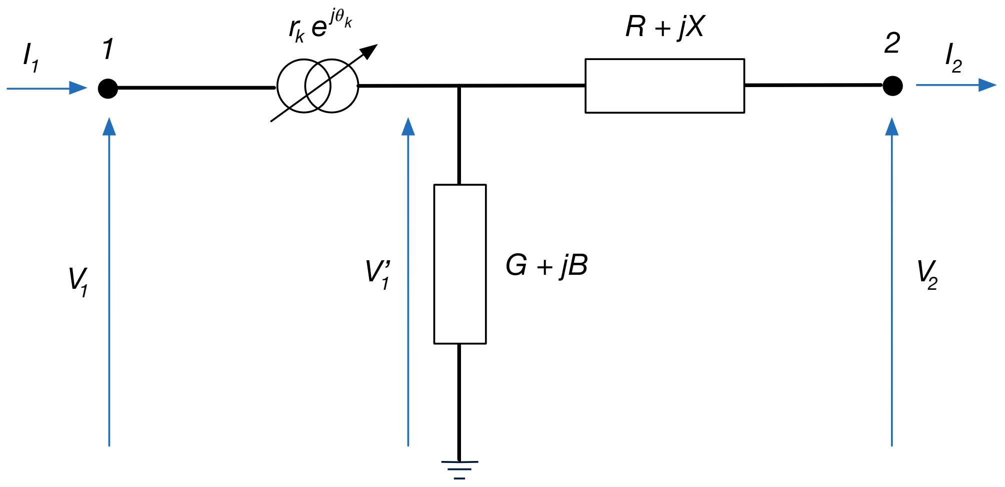

Figure 5: Illustration of the T model for a two winding transformer.

Similar to AC transmission line, without loss of generality, we assume the current is positive when flowing from 1 to 2 (i.e. left to right). The currents I_1 and I_2 at sides 1 and 2:

$$I_1 = \bar{\rho_k} \cdot \left(Y_k \cdot V_1' + \frac{1}{Z_k}(V_1' - V_2)\right) \tag{12}$$

$$I_2 = \frac{1}{Z_k}(V_1' - V_2), \text{ where} \tag{13}$$

$$\rho_k = \left(\left(\frac{V_2^{\text{nom}}}{V_1^{\text{nom}}}\right) \cdot r_k\right) \cdot e^{j\delta_k}, V_1' = V_1\rho_k \tag{14}$$

$Y_k = G_k + jB_k$ is the admittance of the transformer on tap k, $Z_k = R_k + jX_k$ is the impedance of the transformer on tap k, V_1 and V_2 are the voltages at the connecting buses in VL_1 and VL_2, respectively. We use V_1^{nom} and V_2^{nom} to denote the nominal voltage magnitudes at sides 1 and 2 of the transformer, respectively. The power S_i at sides i ($i \in \{1,2\}$) is given by:

$$S_i = \bar{I_i} \cdot V_i \tag{15}$$

As with AC lines, transformers have also both current and thermal limits. The maximum absolute value of the current magnitude is monitored on both sides of the branch. Constraints are described through a sequence of current limits $L_1 = L_{1_1}, \ldots, L_{1_n}$, and $L_2 = L_{2_1}, \ldots, L_{2_n}$, where each L_{i_k} ($i \in \{1,2\}, k \in \{1,\ldots,n\}$) is a current limit object, and for each L_{i_k}, ($k > 1$), $I_{i_k}^{\text{min}} = I_{i_{k-1}}^{max}$. The absolute value of complex power magnitude is similar and also monitored on both sides of the branch. Constraints are described through a sequence of thermal limits $L_1 = L_{1_1}, \ldots, L_{1_n}$, and $L_2 = L_{2_1}, \ldots, L_{2_n}$, where each L_{i_k} ($i \in \{1,2\}, k \in \{1,\ldots,n\}$) is a thermal limit object, and for each L_{i_k}, ($k > 1$), $S_{i_k}^{min} = S_{i_{k-1}}^{\text{max}}$. Finally, the current/thermal limit durations are such that $d_{i_1} = \infty$, denoting that L_{i_1} is a permanent acceptable limit.

Two Winding Transformer: PI model

The PI model (Figure 6) are widely used in the literature, and differs from the T model by moving half of the admittance Y_k to the right hand side. The complex turns ratio $r_k e^{i\delta_k}$ are also used differently. The PI model represents a step-down transformer scaling down the voltage, while the T model represents a step-up transformer scaling up the voltage.

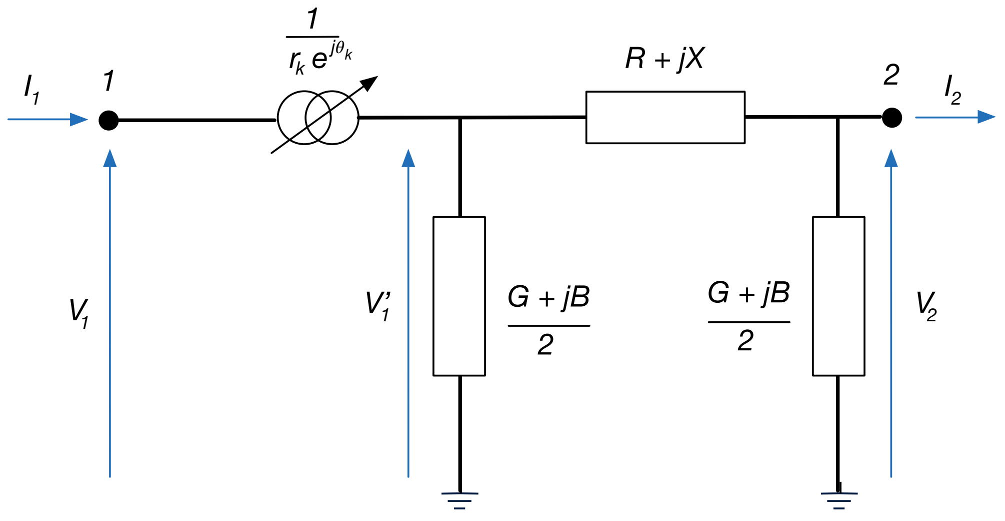

Figure 6: Illustration of the PI model (right) for a two winding transformer.

Without loss of generality, we assume the current is positive when flowing from 1 to 2 (i.e. left to right). The currents I_1 and I_2 at sides 1 and 2 are:

$$I_1 = \bar{\rho_k} \cdot \big(\frac{Y_k}{2} \cdot V_1' + \frac{1}{Z_k}(V_1' - V_2)\big) \tag{16}$$

$$I_2 = \frac{1}{Z_k}(V_1' - V_2) - \frac{Y_k}{2} \cdot V_2, \text{ where} \tag{17}$$

$$\rho_k = \big((\frac{V_2^{\text{nom}}}{V_1^{\text{nom}}}) \cdot \frac{1}{r_k}\big) \cdot e^{-j\delta_k}, V_1' = V_1 \rho_k \tag{18}$$

$Y_k = G_k + jB_k$ is the admittance of the transformer on tap k (half of each side), $Z_k = R_k + jX_k$ is the impedance of the transformer on tap k, V_1 and V_2 are the voltages at the connecting buses in VL$_1$ and VL$_2$, respectively. Similarly, we again use V_1^{nom} and V_2^{nom} to denote the nominal voltage magnitudes at sides 1 and 2 of the transformer, respectively. The power S_i at sides i ($i \in \{1, 2\}$) is again given by:

$$S_i = \bar{I}_i \cdot V_i \tag{19}$$

The thermal and current capacity limits are the same as the T model. Since the T model and the PI model share the same set of components, we use the same JSON format and schema to store and represent both types of transformers.

GRG schema: two_winding_transformer

```
"two_winding_transformer": {
    "type": "object",
    "required": ["type", "id", "link_1", "link_2", "tap_changer"],
    "additionalProperties": true,
    "properties": {
        "type": {
            "enum": ["two_winding_transformer", "T_model_transformer", "PI_model_transformer"]
        },
        "subtype": {"type": "string"},
        "id": {"type": "string"},
        "description": {"type": "string"},
        "link_1": {"type": "string"},
        "link_2": {"type": "string"},
        "current_limits_1": {"$ref": "#/values/limits/current_limits"},
        "current_limits_2": {"$ref": "#/values/limits/current_limits"},
        "thermal_limits_1": {"$ref": "#/values/limits/thermal_limits"},
        "thermal_limits_2": {"$ref": "#/values/limits/thermal_limits"},
        "tap_changer": {
            "type": "object",
            "required": ["position","impedance", "shunt","transform","steps"],
            "additionalProperties": true,
            "properties": {
                "position": {"$ref": "#/values/basic_values/abstract_value" },
                "impedance": {"$ref": "#/values/electrical_values/impedance"},
                "shunt": { "$ref": "#/values/electrical_values/admittance"},
                "transform": {
                    "type": "object",
                    "required": ["tap_ratio", "angle_shift"],
                    "additionalProperties": true,
                    "properties": {
                        "tap_ratio": {"$ref": "#/values/basic_values/abstract_value"},
                        "angle_shift": { "$ref": "#/values/basic_values/abstract_value"}
                    }
                },
                "steps": {
                    "type": "array",
                    "items" : [
                              {
                                "type": "object",
                                "required": ["position", "impedance", "shunt", "transform"],
                                "additionalProperties": true,
                                "properties": {
                                  "position": {"type": "number"},
                                  "impedance": {"$ref": "#/values/electrical_values/impedance"},
                                  "shunt": {"$ref": "#/values/electrical_values/admittance"},
                                        "transform": {
                                          "type": "object",
                                          "required": ["tap_ratio", "angle_shift"],
                                          "additionalProperties": true,
                                          "properties": {
                                             "tap_ratio": { "$ref": "#/values/basic_values/abstract_value"},
                                             "angle_shift": { "$ref": "#/values/basic_values/abstract_value"}
                                           }
                                         }
                                       }
                                    }
                                    ],
                      "minItems": 1,
                      "additionalItems": false
                  }
              }
          }
      }
}
```

In the two winding transformer GRG schema, `type` identify the type of an object. Prior to GRGv4.0, a T model transformer will have `type` equals to string `two_winding_transformer` and a `subtype` equals to string `T_model`. Similarly before GRGv4.0, a PI model transformer will have `type` equals to string `two_winding_transformer` and a `subtype` equals to string `T_model`. Since T model/PI model transformers are specific subclasses of two winding transformers, the label of `two_winding_transformer` are redundant. In GRGv4.0, we unify type and subtype labels. A T model transformer will have `type` set to string `T_model_transformer`, and a PI model transformer will have `type` set to string `PI_model_transformer`. To allow ease of extensions/migrations, the GRGv4.0 schema still accept the old naming conventions, i.e. allowing `type` to be `two_winding_transformer` with a user customized `subtype` field to refer to specific mathematical models.

Field `id` is a unique identifier for global referencing, and `description` (optional) is further used to describe the component. Fields `link_1` and `link_2` are global identifiers for identifying the voltage points being connected to the network component. For a two winding transformer, `link_1` and `link_2` will be the voltage points at side 1 and side 2 respectively. `current_limits_1` and `current_limits_2` describe collections of current limits associated to sides 1 and 2 of the line. `thermal_limits_1` and `thermal_limits_2` describe collections of thermal limits associated to sides 1 and 2 of the line. Both current and thermal limits are optional. Finally, `tap_changer` describes the transformer taps, requiring fields: `position`, `impedance`, `shunt`, `transform` (containing `tap_ratio` and `angle_shift`), and `steps`. The `position` field describes the possible indices (or the pre-set index) of the tap steps for the transformer. The `impedance`, `shunt`, and `transform` (containing `tap_ratio` and `angle_shift`) describes the ranges (or the pre-set values) of the (tap) impedance, shunt, tap ratio, and tap phase shifts. Finally, `steps` is a JSON array object describing *all* the possible assignments for `position`, `impedance`, `shunt`, and `transform` (i.e. `tap_ratio` and `angle_shift`) objects.

A summary of the two winding transformer components in nominal units is provided in Table 9, and an example is provided in Figure 7.

2.4 Three winding Transformer

A three winding transformer connects devices located at three different voltage levels of the network. We denote these voltage levels VL_1, VL_2, and VL_3 and their associated nominal voltage values as V_1^{nom}, V_2^{nom}, and V_3^{nom}, respectively. Figure 8 provides an illustration of a three winding transformer.

Without loss of generality, we assume and define *side 1* to be the high voltage side, *side 2* to be the medium voltage side, and *side 3* to be the low voltage side. To simplify our notations, we assume the current flows from side 1 to side 2 and side 3 (i.e. flows from the left to the right in Figure 8). Voltages V_1, V_2, and V_3 are the voltages at the connecting/terminal buses, and V_1', V_2', and V_3' will be their corresponding voltages behind the three transformer taps. Let V_m to be the voltage in the star middle point.

ρ_{1_k}, ρ_{2_k}, and ρ_{3_k}, corresponding to the k-th transformer tap, will then be the complex voltage multipliers/ratios on

GRG name	symbol	unit
link_1	side 1	
link_2	side 2	
current_limits_1[k]→duration	d_{1_k}	Seconds (*sec*)
current_limits_1[k]→min	$I_{1_k}^{min}$	Ampere (A)
current_limits_1[k]→max	$I_{1_k}^{max}$	Ampere (A)
current_limits_2[k]→duration	d_{2_k}	Seconds (*sec*)
current_limits_2[k]→min	$I_{2_k}^{min}$	Ampere (A)
current_limits_2[k]→max	$I_{2_k}^{max}$	Ampere (A)
thermal_limits_1[k]→duration	d_{1_k}	Seconds (*sec*)
thermal_limits_1[k]→min	$S_{1_k}^{min}$	MegaWatt (MW)
thermal_limits_1[k]→max	$S_{1_k}^{max}$	MegaWatt (MW)
thermal_limits_2[k]→duration	d_{2_k}	Seconds (*sec*)
thermal_limits_2[k]→min	$S_{2_k}^{min}$	MegaWatt (MW)
thermal_limits_2[k]→max	$S_{2_k}^{max}$	MegaWatt (MW)
tap_changer→position	k	
tap_changer→impedance→resistance	R_k	Ohm (Ω)
tap_changer→impedance→reactance	X_k	Ohm (Ω)
tap_changer→shunt→conductance	G_k	Siemens (S)
tap_changer→shunt→susceptance	B_k	Siemens (S)
tap_changer→transform→tap_ratio	$(\frac{V_2^{\text{nom}}}{V_1^{\text{nom}}})r_k$ (T model)	voltage-ratio
	$(\frac{V_1^{\text{nom}}}{V_2^{\text{nom}}})r_k$ (PI model)	voltage-ratio
tap_changer→transform→angle_shift	δ_k	degrees
tap_changer→steps[k]→position	k	
tap_changer→steps[k]→impedance→resistance	R_k	Ohm (Ω)
tap_changer→steps[k]→impedance→reactance	X_k	Ohm (Ω)
tap_changer→steps[k]→shunt→conductance	G_k	Siemens (S)
tap_changer→steps[k]→shunt→susceptance	B_k	Siemens (S)
tap_changer→steps[k]→transform→tap_ratio	$(\frac{V_2^{\text{nom}}}{V_1^{\text{nom}}})r_k$ (T model)	voltage-ratio
	$(\frac{V_1^{\text{nom}}}{V_2^{\text{nom}}})r_k$ (PI model)	voltage-ratio
tap_changer→steps[k]→transform→angle_shift	δ_k	degrees

Table 9: Two winding transformer: representation in the GRG format and units.

sides 1, 2, and 3 of the transformer:

$$V_1' = \rho_{1_k} V_1, \text{ where } \rho_{1_k} = \Big((\frac{V_m^{\text{nom}}}{V_1^{\text{nom}}}) r_{1_k} \Big) \cdot e^{j\delta_{1_k}}, \quad (20)$$

$$V_2' = \rho_{2_k} V_2, \text{ where } \rho_{2_k} = \Big((\frac{V_m^{\text{nom}}}{V_2^{\text{nom}}}) r_{2_k} \Big) \cdot e^{j\delta_{2_k}}, \quad (21)$$

$$V_3' = \rho_{3_k} V_3, \text{ where } \rho_{3_k} = \Big((\frac{V_m^{\text{nom}}}{V_3^{\text{nom}}}) r_{3_k} \Big) \cdot e^{j\delta_{3_k}} \quad (22)$$

where V_1^{nom}, V_2^{nom}, and V_3^{nom} denote the nominal voltage magnitudes at sides 1, 2, and 3 of the transformer. In addition, V_m^{nom} denotes the nominal voltage magnitudes of the voltage in the star middle point, and can be freely set to the nominal voltage magnitude on any side of the transformer. Finally, the current at sides 1, 2, and 3 is given,

Example: Two Winding Transformer

```
"two_winding_transformer_1" : {
    "type" : "T_model_transformer",
    "id"                : "transformer_1",
    "link_1"            : "voltage_id_9",
    "link_2"            : "voltage_id_11",
    "current_limits_1" : [
      {"duration": "Inf", "min": 0, "max": 1029, "report": "off"},
      {"duration": 1200, "min": 1029, "max": 1342, "report": "off"},
      {"duration": 300,  "min": 1342, "max": 1790, "report": "off"},
      {"duration": 60,  "min": 1790, "max": "Inf", "report": "off"}
     ],
    "tap_changer": {
          "position":  { "var": {  "lb": 0, "ub": 0}},
          "impedance": { "resistance": { "var": { "lb": 0.0,"ub": 0.0}},
                         "reactance": {  "var": { "lb": 0.25,"ub": 0.25}}},
          "shunt": {"conductance": {"var": {"lb": 0.0,"ub": 0.0}},
                    "susceptance": {"var": {"lb": 0.0, "ub": 0.0}}},
          "transform": {
              "tap_ratio": {"var": {"lb": 0.0,"ub": 11.0 }},
              "angle_shift": {"var": {"lb": 0.0, "ub": 0.0 }}
          },
          "steps": [
             { "position": 0,
               "impedance":  { "resistance": 0.0,"reactance": 0.2516799999999999 },
               "shunt": { "conductance": 0.0,"susceptance": 0.0},
               "transform": { "tap_ratio":  11.0, "angle_shift": 0.0 }}
          ]
    }
}
```

Figure 7: An example of a GRG Two Winding Transformer.

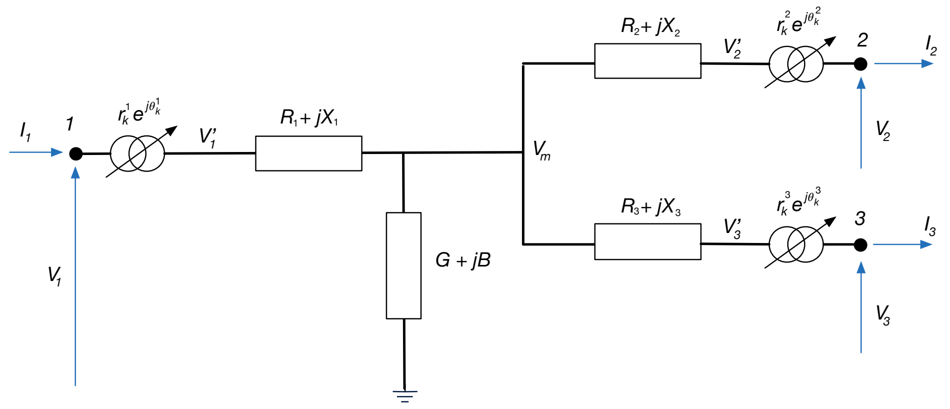

Figure 8: Illustration of a three winding transformer.

respectively by:

$$I_1 = \bar{\rho}_{1_k} \cdot \frac{1}{Z_{1_k}}(V_1 \rho_{1_k} - V_m) \tag{23}$$

$$I_2 = \bar{\rho}_{2_k} \cdot \frac{1}{Z_{2_k}}(V_m - V_2 \rho_{2_k}) \tag{24}$$

$$I_3 = \bar{\rho}_{3_k} \cdot \frac{1}{Z_{3_k}}(V_m - V_3 \rho_{3_k}), \tag{25}$$

where $Y_k = G_k + jB_k$ is the admittance at side 1 of the transformer with tap k; $Z_{1_k} = R_1 + jX_1$, $Z_{2_k} = R_2 + jX_2$, and $Z_{3_k} = R_3 + jX_3$ are the impedance values of sides 1, 2, and 3 of the transformer with tap k. The power S_i at sides i ($i \in \{1, 2, 3\}$) is given by:

$$S_i = \bar{I}_i \cdot V_i \tag{26}$$

GRG schema: three_winding_transformer

```
"three_winding_transformer": {
    "type": "object",
    "required": ["type", "id", "link_1", "link_2", "link_3",
                      "tap_changer_1", "tap_changer_2", "tap_changer_3"],
    "additionalProperties": true,
    "properties": {
        "type": { "enum": [ "three_winding_transformer"] },
        "subtype": { "type": "string" },
        "id": { "type": "string"},
        "description": { "type": "string" },
        "link_1": { "type": "string"},
        "link_2": { "type": "string"},
        "link_3": { "type": "string"},
        "current_limits_1": { "$ref": "#/values/limits/current_limits"},
        "current_limits_2": { "$ref": "#/values/limits/current_limits"},
        "current_limits_3": { "$ref": "#/values/limits/current_limits"},
        "thermal_limits_1": { "$ref": "#/values/limits/thermal_limits"},
        "thermal_limits_2": { "$ref": "#/values/limits/thermal_limits"},
        "thermal_limits_3": { "$ref": "#/values/limits/thermal_limits"},
        "tap_changer_1": {
            "type": "object",
            "required": ["position", "impedance", "shunt", "transform","steps"],
            "additionalProperties": true,
            "properties": {
                "position": { "$ref": "#/values/basic_values/abstract_value"},
                "impedance": { "$ref": "#/values/electrical_values/impedance"},
                "shunt": {"$ref": "#/values/electrical_values/admittance"},
                "transform": {
                    "type": "object",
                    "required": ["tap_ratio", "angle_shift"],
                    "additionalProperties": true,
                    "properties": {
                        "tap_ratio": {"$ref": "#/values/basic_values/abstract_value" },
                        "angle_shift": { "$ref": "#/values/basic_values/abstract_value" }
                    }
                },
                "steps": {
                    "type": "array",
                    "items" : [{
                               "type": "object",
                               "required": ["position", "impedance", "shunt", "transform"],
                               "additionalProperties": true,
                               "properties": {
                               "position": {"type": "number"},
                               "impedance": {"$ref": "#/values/electrical_values/impedance"},
                                "shunt": {"$ref": "#/values/electrical_values/admittance"},
                               "transform": {
                                  "type": "object",
                                  "required": ["tap_ratio", "angle_shift"],
                                  "additionalProperties": true,
                                  "properties": {
                                    "tap_ratio": {"$ref": "#/values/basic_values/abstract_value"},
                                    "angle_shift": {"$ref": "#/values/basic_values/abstract_value"}
                                  }
                               }
                               }
                               }],
                    "minItems": 1,
                    "additionalItems": false
                }
            }
        },
        "tap_changer_2": {
            "type": "object",
            "required": [ "position", "impedance", "transform", "steps"],
            "additionalProperties": true,
            "properties": {
                "position": {"$ref": "#/values/basic_values/abstract_value"},
                "impedance": { "$ref": "#/values/electrical_values/impedance"},
                "transform": {
                    "type": "object",
                    "required": ["tap_ratio", "angle_shift"],
                    "additionalProperties": true,
                    "properties": {
                        "tap_ratio": { "$ref": "#/values/basic_values/abstract_value"},
                        "angle_shift": {"$ref": "#/values/basic_values/abstract_value" }
                    }
                },
                "steps": {
                    "type": "array",
                    "items" : [{
                               "type": "object",
                               "required": ["position", "impedance", "transform"],
                               "additionalProperties": true,
                               "properties": {
                                  "position": {"type": "number"},
                                  "impedance": {"$ref": "#/values/electrical_values/impedance"},
                                  "transform": {
                                    "type": "object",
                                    "required": ["tap_ratio", "angle_shift"],
                                    "additionalProperties": true,
                                    "properties": {
                                      "tap_ratio": {"$ref": "#/values/basic_values/abstract_value"},
                                      "angle_shift": {"$ref": "#/values/basic_values/abstract_value"}
                                   }
                                  }
                               }
                               }],
                    "minItems": 1,
                    "additionalItems": false
                }
            }
        },
        "tap_changer_3": {
            "type": "object",
            "required": [ "position", "impedance", "transform", "steps"],
            "additionalProperties": true,
            "properties": {
                "position": { "$ref": "#/values/basic_values/abstract_value"},
                "impedance": { "$ref": "#/values/electrical_values/impedance"},
                "transform": {
                    "type": "object",
                    "required": ["tap_ratio", "angle_shift"],
                    "additionalProperties": true,
                    "properties": {
                        "tap_ratio": {"$ref": "#/values/basic_values/abstract_value"},
                        "angle_shift": {"$ref": "#/values/basic_values/abstract_value"}
                    }
                },
                "steps": {
                    "type": "array",
                    "items" : [{
                               "type": "object",
                               "required": ["position", "impedance", "transform"],
                               "additionalProperties": true,
                               "properties": {
                                  "position": {"type": "number"},
                                  "impedance": {"$ref": "#/values/electrical_values/impedance"},
                                  "transform": {
                                    "type": "object",
                                    "required": ["tap_ratio", "angle_shift"],
                                    "additionalProperties": true,
                                    "properties": {
                                       "tap_ratio": {"$ref": "#/values/basic_values/abstract_value"},
                                       "angle_shift": {"$ref": "#/values/basic_values/abstract_value"}
                                    }
                                  }
                               }
                               }],
                "minItems": 1,
                "additionalItems": false
            }
        }
    }
}
}
```

In the three winding transformer GRG schema, `type` and `subtype` (optional) identify the type of the object (for example a *three winding transformer*). `id` is a unique identifier for global referencing, and `description` is an optional field to describe the component. Fields `link_1`, `link_2`, and `link_3` are global identifiers for identifying the voltage points being connected to the component. For a three winding transformer, `link_1`, `link_2`, and `link_3` will be the voltage points at side 1, 2 and 3 respectively. `current_limits_1`, `current_limits_2`, and `current_limits_3` describe collections of current limits associated to sides 1, 2, and 3. `thermal_limits_1`, `thermal_limits_2`, and `thermal_limits_3` describe collections of thermal limits associated to sides 1, 2, and 3 accordingly. Again similar to two winding transformers, both type of limits are optional.

Finally, `tap_changer_1`, `tap_changer_2`, and `tap_changer_3` describe the three transformer taps at side 1, 2, and 3 respectively. All of the transformer taps require fields: `position`, `impedance`, `transform` (containing `tap_ratio` and `angle_shift`), and `steps`. `tap_changer_1` further requires the `shunt` field. The `position` fields describe the possible indices (or the pre-set index) for each of the taps. The `impedance`, `shunt`, and `transform` (containing `tap_ratio` and `angle_shift`) fields describe the ranges (or the pre-set values) of the (tap) impedances, shunts, tap ratios, and phase shifts for the corresponding transformer. `steps` is a JSON array object describing *all* the possible assignments for `position`, `impedance`, `shunt` (for `tap_changer_1` only), and `transform` (i.e. `tap_ratio` and `angle_shift`) objects.

A summary of three winding transformer components is provided in Table 10, and Figure 9 provides an example.

2.5 Switch

A switch is an electrical component that can interrupt the current in a circuit. There are two types of switches: *circuit breakers* and *isolators*. Circuit breakers can be switched on or off when they are energized, while isolators (also called disconnectors) can be switched only when not energized. These are series devices with two sides denoted side 1 and side 2. Their operation satisfies the following:

$$s \cdot V_1 = s \cdot V_2, \tag{27}$$

where s is a binary variable denoting the switch status (1 for open, 0 for closed), and V_1 and V_2 are the voltages at sides 1 and 2 of the switch.

Figure 10 illustrates an open isolator (a) and breaker (c), and a closed isolator (b) and breaker (d).

GRG schema: switch

```
"switch": {
    "type"    : "object",
    "required": ["type", "subtype", "id", "status", "link_1", "link_2"],
    "additionalProperties": true,
    "properties": {
        "type"       : {"enum": ["switch"]},
        "subtype"    : {"enum": ["breaker", "isolator"]},
        "description": {"type": "string"},
        "id"         : {"type": "string"},
        "status"     : {"$ref": "#/values/basic_values/status"},
        "link_1"     : {"type": "string"},
        "link_2"     : {"type": "string"}
    }
}
```

GRG name	symbol	unit
link_1	side 1	
link_2	side 2	
link_3	side 3	
current_limits_1[k]→duration	d_{1_k}	Seconds (sec)
current_limits_1[k]→min	$I^{min}_{1_k}$	Ampere (A)
current_limits_1[k]→max	$I^{max}_{1_k}$	Ampere (A)
current_limits_2[k]→duration	d_{2_k}	Seconds (sec)
current_limits_2[k]→min	$I^{min}_{2_k}$	Ampere (A)
current_limits_2[k]→max	$I^{max}_{2_k}$	Ampere (A)
current_limits_3[k]→duration	d_{3_k}	Seconds (sec)
current_limits_3[k]→min	$I^{min}_{3_k}$	Ampere (A)
current_limits_3[k]→max	$I^{max}_{3_k}$	Ampere (A)
thermal_limits_1[k]→duration	d_{1_k}	Seconds (sec)
thermal_limits_1[k]→min	$S^{min}_{1_k}$	MegaWatt (MW)
thermal_limits_1[k]→max	$S^{max}_{1_k}$	MegaWatt (MW)
thermal_limits_2[k]→duration	d_{2_k}	Seconds (sec)
thermal_limits_2[k]→min	$S^{min}_{2_k}$	MegaWatt (MW)
thermal_limits_2[k]→max	$S^{max}_{2_k}$	MegaWatt (MW)
thermal_limits_3[k]→duration	d_{3_k}	Seconds (sec)
thermal_limits_3[k]→min	$S^{min}_{3_k}$	MegaWatt (MW)
thermal_limits_3[k]→max	$S^{max}_{3_k}$	MegaWatt (MW)
tap_changer_1→position	k	
tap_changer_1→impedance→resistance	R_{1_k}	Ohm (Ω)
tap_changer_1→impedance→reactance	X_{1_k}	Ohm (Ω)
tap_changer_1→shunt→conductance	G_{1_k}	Siemens (S)
tap_changer_1→shunt→susceptance	B_{1_k}	Siemens (S)
tap_changer_1→transform→tap_ratio	$(\frac{V_m^{\text{nom}}}{V_1^{\text{nom}}})r_{1_k}$	voltage-ratio
tap_changer_1→transform→angle_shift	δ_{1_k}	degrees
tap_changer_1→steps[k]→position	k	
tap_changer_1→steps[k]→impedance→resistance	R_{1_k}	Ohm (Ω)
tap_changer_1→steps[k]→impedance→reactance	X_{1_k}	Ohm (Ω)
tap_changer_1→steps[k]→shunt→conductance	G_{1_k}	Siemens (S)
tap_changer_1→steps[k]→shunt→susceptance	B_{1_k}	Siemens (S)
tap_changer_1→steps[k]→transform→tap_ratio	$(\frac{V_m^{\text{nom}}}{V_1^{\text{nom}}})r_{1_k}$	voltage-ratio
tap_changer_1→steps[k]→transform→angle_shift	δ_k	degrees
tap_changer_2→position	k	
tap_changer_2→impedance→resistance	R_{2_k}	Ohm (Ω)
tap_changer_2→impedance→reactance	X_{2_k}	Ohm (Ω)
tap_changer_2→transform→tap_ratio	$(\frac{V_m^{\text{nom}}}{V_2^{\text{nom}}})r_{2_k}$	voltage-ratio
tap_changer_2→transform→angle_shift	δ_{2_k}	degrees
tap_changer_2→steps[k]→position	k	
tap_changer_2→steps[k]→impedance→resistance	R_{2_k}	Ohm (Ω)
tap_changer_2→steps[k]→impedance→reactance	X_{2_k}	Ohm (Ω)
tap_changer_2→steps[k]→transform→tap_ratio	$(\frac{V_m^{\text{nom}}}{V_2^{\text{nom}}})r_{2_k}$	voltage-ratio
tap_changer_2→steps[k]→transform→angle_shift	δ_{2_k}	degrees
tap_changer_3→position	k	
tap_changer_3→impedance→resistance	R_{3_k}	Ohm (Ω)
tap_changer_3→impedance→reactance	X_{3_k}	Ohm (Ω)
tap_changer_3→transform→tap_ratio	$(\frac{V_m^{\text{nom}}}{V_3^{\text{nom}}})r_{3_k}$	voltage-ratio
tap_changer_3→transform→angle_shift	δ_{3_k}	degrees
tap_changer_3→steps[k]→position	k	
tap_changer_3→steps[k]→impedance→resistance	R_{3_k}	Ohm (Ω)
tap_changer_3→steps[k]→impedance→reactance	X_{3_k}	Ohm (Ω)
tap_changer_3→steps[k]→transform→tap_ratio	$(\frac{V_m^{\text{nom}}}{V_3^{\text{nom}}})r_{3_k}$	voltage-ratio
tap_changer_3→steps[k]→transform→angle_shift	δ_{3_k}	degrees

Table 10: Three winding transformer: representation in the GRG format and units.

The field `type` identifies the type of the object (`switch`), `subtype` denotes whether the switch is a breaker or an isolator (disconnector), `description` is an optional field for describing the switch, `id` is a unique identifier, `status` denotes whether the switch is open or closed, and `link_1` and `link_2` identify the connecting voltage points at sides 1 and side 2 of the switch.

Table 11 summarizes switch components, and Figure 11 provides an example.

Example: Three Winding Transformer

```
"three_winding_transformer_1" : {
    "type"              : "three_winding_transformer",
    "id"                : "transformer_1",
    "link_1"            : "voltage_id_2",
    "link_2"            : "voltage_id_3",
    "link_3"            : "voltage_id_6",
    "tap_changer_1": {
          "position":  { "var": {  "lb": 0, "ub": 0}},
          "impedance": { "resistance": { "var": { "lb": 1.512,"ub": 1.512}},
                         "reactance": {  "var": { "lb": 60.1,"ub": 60.1}}},
          "shunt": {"conductance": {"var": {"lb": 0.0,"ub": 0.0}},
                    "susceptance": {"var": {"lb": 0.0, "ub": 0.0}}},
          "transform": {
              "tap_ratio": {"var": {"lb": 1.0,"ub": 1.0 }},
              "angle_shift": {"var": {"lb": 0.0, "ub": 0.0 }}
          },
          "steps": [
             { "position": 0,
               "impedance":  { "resistance": 1.512,"reactance": 60.1 },
               "shunt": { "conductance": 0.0,"susceptance": 0.0},
               "transform": { "tap_ratio":  1.0, "angle_shift": 0.0 }}
          ]
    },
     "tap_changer_2": {
          "position":  { "var": {  "lb": 0, "ub": 0}},
          "impedance": { "resistance": { "var": { "lb": 0.0, "ub": 0.0}},
                                 "reactance": {  "var": { "lb": 0.25,"ub": 0.25}}},
          "transform": {
              "tap_ratio": {"var": {"lb": 0.0,"ub": 11.0 }},
              "angle_shift": {"var": {"lb": 0.0, "ub": 0.0 }}
          },
          "steps": [
             { "position": 0,
               "impedance":  { "resistance": 0.0,"reactance": 0.25 },
               "transform": { "tap_ratio":  11.0, "angle_shift": 0.0 }}
          ]
    },
    "tap_changer_3": {
          "position":  { "var": {  "lb": 0, "ub": 0}},
          "impedance": { "resistance": { "var": { "lb": 0.0, "ub": 0.0}},
                                 "reactance": {  "var": { "lb": 0.21,"ub": 0.21}}},
          "transform": {
              "tap_ratio": {"var": {"lb": 0.0,"ub": 9.0 }},
              "angle_shift": {"var": {"lb": 0.0, "ub": 0.0 }}
          },
          "steps": [
             { "position": 0,
               "impedance":  { "resistance": 0.0,"reactance": 0.21 },
               "transform": { "tap_ratio":  9.0, "angle_shift": 0.0 }}
          ]
    }
}
```

Figure 9: An example of a GRG Three Winding Transformer.

GRG name	symbol	unit
status	s	boolean

Table 11: Switch representation in GRG format and units.

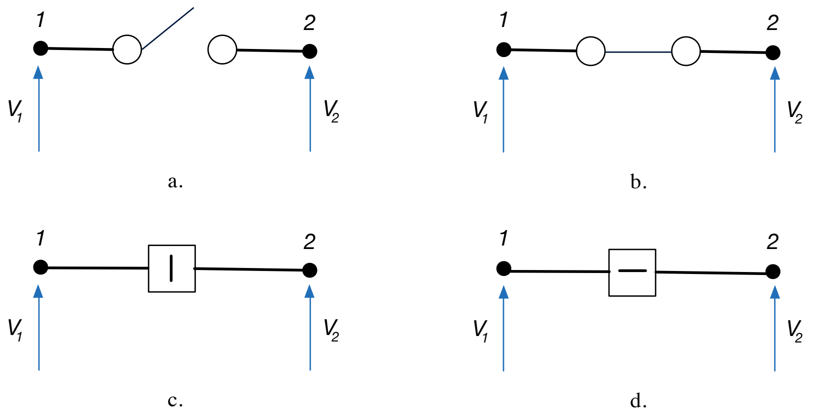

Figure 10: Illustration of an open (a) and close (b) isolator, and an open (c) and close (d) breaker.

Example: Switch
```
"switch_F" : {
    "type"    : "switch",
    "subtype": "breaker",
    "id"      : "sw_1",
    "link_1" : "voltage_id_C_0",
    "link_2" : "voltage_id_C_3",
    "status" : {"var" : ["off", "on"]}
}
```

Figure 11: An example of a GRG switch.

2.6 Bus

A bus is a set of equipment connected together. It could be a configured object or the result of a computation, depending of the context.

GRG schema: bus
```
"bus": {
    "type"     : "object",
    "required": ["type", "id", "link", "voltage"],
    "additionalProperties": true,
    "properties": {
        "type"        : {"enum": ["bus"]},
        "subtype"     : {"type": "string"},
        "description": {"type": "string"},
        "id"          : {"type": "string"},
        "link"        : {"type": "string"}
        "voltage"     : {"$ref": "#/values/electrical_values/voltage"}
        "name"        : {"type": "string"},
    }
}
```

`type` identifies the type of the object. Depending on the network topology adopted, a bus can be represented as a `busbar` (node-breaker topology), `logical bus` (bus-breaker topology), or simply `bus` (bus-branch topology). These information can be stored in the `subtype` optional field. `description` is an optional description field, and `id` is a unique identifier. The field `link` identifies the voltage point at which the bus is connected. Finally, `voltage` refers to the bus voltage magnitude v and phase angle θ, and `name` denotes the bus name.

Table 12 tabulates the bus components, while Figure 12 contains an example of a bus in GRG format.

GRG name	symbol	unit
voltage→magnitude	v	kiloVolt (kV)
voltage→angle→lb	Θ^l	Degrees
voltage→angle→ub	Θ^u	Degrees

Table 12: Bus representation in GRG format and units.

```
Example: Busbar
"bus_Q" : {
    "type" : "bus",
    "id"   : "bus_6",
    "link" : "voltage_id_P_0",
    "voltage" : {
        "angle"     : {"var" : {"lb" : -30.0, "ub" : 30.0}},
        "magnitude" : {"var" : {"lb" : "0", "ub" : "500"}}
    }
}
```

Figure 12: An example of a GRG busbar.

2.7 Shunt

A shunt capacitor or reactor is defined as an admittance:

$$I = -Y \cdot V,$$

where $Y = G + jB$ is the admittance, and V is the voltages at the connecting point. Figure 13 illustrates a shunt.

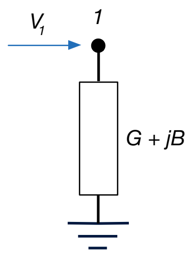

Figure 13: A shunt.

```
GRG schema: shunt
"shunt": {
    "type"    : "object",
    "required": ["type", "id", "link", "shunt"],
    "additionalProperties": true,
    "properties": {
        "type"       : {"enum": ["shunt"]},
        "subtype"   : {"type": "string"},
        "id"         : {"type": "string"},
        "description": {"type": "string"},
        "link"       : {"type": "string"},
        "shunt"      : {"$ref": "#/values/electrical_values/admittance"}
    }
}
```

In the above schema, `type` identifies the type of the object (i.e., a `shunt`), `id` is a unique identifier, and `description` is an optional description field. The field `link` identifies the connecting voltage point. Finally, `shunt` defines the admittance Y.

Table 13 summarizes shunt components, while Figure 14 provides an example.

GRG name	symbol	unit
shunt→conductance	G	Siemens (S)
shunt→susceptance	B	Siemens (S)

Table 13: Shunt representation in the GRG format and units.

```
Example: Shunt
"shunt_IO" : {
    "id"   : "sh_10",
    "type" : "shunt",
    "subtype": "inductor",
    "link" : "voltage_id_HS_9",
    "shunt": {"conductance" : 0.0, "susceptance" : -0.16}
}
```

Figure 14: An example of a GRG shunt.

2.8 Load

A Load consumes active power P and reactive power Q at its connection point, as illustrated in Figure 15.

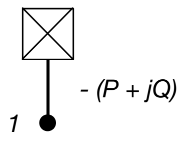

Figure 15: A load.

```
load definition
"load": {
    "type": "object",
    "required": ["type", "id", "link", "demand"],
    "additionalProperties": true,
    "properties": {
        "type"       : {"enum": ["load"]},
        "subtype"    : {"type": "string"},
        "id"         : {"type": "string"},
        "description": {"type": "string"},
        "link"       : {"type": "string"},
        "demand"     : {"$ref": "#/values/electrical_values/power"},
    }
}
```

In the load GRG schema, `type` identifies the type of the object (i.e., a `load`), `subtype` identifies the type of load (for example `withdrawal` to indicate the load will only withdraw power). The attribute `id` is a unique identifier for the load component, and `description` is an optional description field. The field `link` identifies the connecting voltage point in the network. Finally, `demand` defines the power S consumed by the load, and can be set to a variable to be assinged later.

Table 14 summarizes load components, while Figure 16 provides an example.

GRG name	symbol	unit
demand→active	P	MegaWatt (*MW*)
demand→reactive	Q	MegaVolt-Ampere Reactive (*MVAR*)

Table 14: Load representation in GRG format and units.

```
________________________________ Example: Load ________________________________
"load_AER" : {
    "type"   : "load"
    "subtype": "withdrawal",
    "id"     : "ld_6",
    "link"   : "voltage_id_ACM_13",
    "demand" : {
        "active"   : {"var" : {"lb" : "-Inf", "ub" : "Inf"}},
        "reactive" : {"var" : {"lb" : "-Inf", "ub" : "Inf"}}
    }
}
```

Figure 16: An example of a GRG load.

2.9 Generator

A generator produces active power (P) and reactive power (Q), and supports voltage (V). Its output is limited according to a PQ-curve, which specifies minimum and maximum reactive power values for every active power value. More formally, a PQ-curve is a sequence of tuples: $\langle P_k, Q_k^{min}, Q_k^{max} \rangle_{k=1}^{n}$ (with $n > 0$) such that for each $k < n$, $P_k < P_{k+1}$, and $Q_k^{min} \leq Q_k^{max}$.

Figure 17 illustrates a generator (a) and feasible PQ region (shaded gray area) of its PQ-curve.

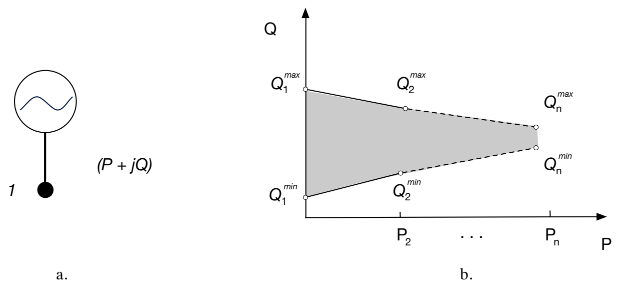

Figure 17: Illustration of a generator (a) and its PQ-curve (b).

```
____________________________ GRG schema: generator ____________________________
"generator": {
    "type"    : "object",
    "required": ["type", "id", "link", "output"],
    "additionalProperties": true,
    "properties": {
        "type"   : {"enum": ["generator"]},
        "subtype": {"enum": ["hydro", "wind", "thermal", "other", "nuclear", "solar"]},
        "id"     : {"type": "string"},
        "description": {"type": "string"},
        "link"   : {"type": "string"},
        "output" : {"$ref": "#/values/electrical_values/power"},
        "PQ_curve"   : {
            "type": "object",
            "required": ["arguments", "values"],
            "additionalProperties": true,
            "properties": {
                "arguments": {
                    "type": "array",
                    "items": [
                        {"enum": ["active"]}, {"enum": ["reactive_lb"]}, {"enum": ["reactive_ub"]}
                    ],
                    "minItems": 3, "maxItems": 3, "additionalItems": false
                },
                "values": {
                    "type": "array",
                    "items": [
                    {
                        "type": "array",
                        items": [
                            {"$ref": "#/values/basic_values/extended_number"},
                            {"$ref": "#/values/basic_values/extended_number"},
                            {"$ref": "#/values/basic_values/extended_number"}
                        ],
                        "minItems": 3, "maxItems": 3, "additionalItems": false
                        }
                    ],
                    "minItemes": 1
                }
            }
        }
    }
}
```

In the generator GRG schema, `type` identifies the type of the object (i.e., a `generator`), `subtype` identifies the type of generator, `id` is a unique identifier, and `description` is an optional description field. `link` identifies the connecting voltage point in the network. The `output` field defines the power S produced by the generator. Finally, the `PQ-curve` is a table of active and reactive bounds, defining the feasible operating region.

Table 15 summarizes generator components, and Figure 18 provides an example.

GRG name	symbol	unit
`output`→`active`	P	MegaWatt (*MW*)
`output`→`reactive`	Q	MegaVolt-Ampere Reactive (*MVAR*)
`PQ_curve`→`values`[k][0]	P_k	MegaWatt (*MW*)
`PQ_curve`→`values`[k][1]	Q_k^{min}	MegaVolt-Ampere Reactive (*MVAR*)
`PQ_curve`→`values`[k][2]	Q_k^{max}	MegaVolt-Ampere Reactive (*MVAR*)

Table 15: Generator: representation in the GRG format and units.

Example: Generator
```
"generator_AEC" : {
    "type"    : "generator",
    "subtype": "solar",
    "id"      : "gen_4",
    "link"    : "voltage_id_ACM_5",
        "PQ_curve" : {
           "arguments": ["active", "reactive_lb", "reactive_ub"],
           "values"   : [[0.0,   0.0, 0.0],
                          20.7, 0.0, 0.0]]
    }
    "output" : {
        "active"   : {"var" : {"lb" : 0.0, "ub" : 20.7}},
        "reactive" : {"var" : {"lb" : 0.0, "ub" : 0.0}}
    }
}
```

Figure 18: An example of a GRG generator.

2.10 Synchronous Condenser

A synchronous condenser is a spinning machine that can compensate lagging current by either generating or absorbing reactive power.

GRG schema: synchronous_condenser
```
"synchronous_condenser": {
    "type"     : "object",
    "required": ["type", "id", "link", "output"],
    "additionalProperties": true,
    "properties": {
         "type"  : {"enum": ["synchronous_condenser"]},
         "subtype":{"type": "string"},
         "id"    : {"type": "string"},
         "link"  : {"type": "string"},
         "output": {
             "reactive": {"$ref": "#/values/electrical_values/abstract_value"}
            }
    }
}
```

In the `synchronous_condenser` GRG schema, `type` and `subtype` identifies the type of the object, the attribute `id` is a unique identifier, and `description` is an optional description field. `link` identifies the connecting voltage point in the network. `output` defines the reactive power (`reactive`) produced or absorbed by the condenser.

Table 16 summarizes synchronous condenser components, and Figure 19 provides an example.

GRG name	symbol	unit
`output`→`reactive`	Q	MegaVolt-Ampere Reactive ($MVAR$)

Table 16: Synchronous condenser representation in GRG format and units.

```
                                  Example: Synchronous Condenser
"sync_cond_3" : {
    "type"   : "synchronous_condenser",
    "id"     : "sync_3",
    "link"   : "voltage_id_11",
    "output": {
        "reactive" : {"var" : {"lb" : 0, "ub" : 14.53}}
    }
}
```

Figure 19: An example of a GRG synchronous condenser.

3 Assignment and Mappings

An *assignment* is a JSON object which contains a list of value assignments for some network component. Assignments may be used to define a particular instance of a network, for example, by assigning values to a set of variables. The assignment schema is as follows.

```
GRG schema: assignment
"assignment": {
    "type" : "object",
    "patternProperties": {
        ".*": {
        "oneOf": [
            {"type": "object"},
            {"$ref": "#/values/basic_values/abstract_value"},
            {"$ref": "#/values/basic_values/status"}]
        }
    }
}
```

The JSON reference refers to any of the components described in section 2, and the `patternProperties` specifies that zero or more component assignments can be specified.

We say that a network component field is *assigned* if its value is specified as an object in an `assignment`. An assignment example is shown in Figure 20.

```
Example: Assignment
"assignment": {
    "switch_773/status": {"on"},
    "bus_6/voltage" : {
            "angle"    : 4.23,
            "magnitude": 63.12
    }
}
```

Figure 20: An example of a GRG assignment.

A *mapping* is a set of assignments. It can be used to to define a particular instantiation of a network, or a desired network state, e.g., target values. Its schema is as follows. In GRGv4.0, we also allow users to define various extra information, e.g. reference bus, in the mapping section. Users are free to add extra information in the *mapping* block, or extra operational information in the *operational constraints* block.

```
GRG schema: mappings
"mappings": {
    "type": "object",
    "patternProperties": {
        ".*": {"$ref": "#/network/network_assignments"}
    }
}
```

The `patternProperties` field specifies that zero or more set of component assignments can be specified. For each of these sets, an arbitrary number of component assignments can be specified. Figure 21 contains an example.

Similar to mappings, an *operation constraints* block is a set of assignments. It is reserved for posting operational constraints to restrict the range of values allowed to be assigned to variables/states in the network. Its schema is as follows.

```
GRG schema: mappings
"operation_constraints": {
        "type": "object",
        "patternProperties": {
            "#/values/basic_values/grg_pointer":   {
                "$ref": "#/network/network_assignments"
             }
         }
}
```

```
                                   Example: Mappings
"mappings": {
    "starting_points": {
        "bus_6/voltage": {
            "magnitude": 132.22,
            "angle": -10.15
        },
        "ld_7/demand": {
            "active": 5.80,
            "reactive": 2.0
        },
        "sync_3/output": {
            "active": 0.0,
            "reactive": 11.87
        },
        "transformer_1/tap_changer/position": 0
    }
}
```

Figure 21: An example of GRG mappings.

The `patternProperties` field again specifies that zero or more set of component assignments can be specified. Figure 22 contains an example on how we describe the angle difference constraints on AC lines and two winding transformers, by explicitly restricting the angle difference value has to be within the range of [-30,30] (i.e. 30 degree angle difference constraints).

```
                              Example: Operation Constraints
"operation_constraints": {
    "transformer_6/angle_difference": {
        "var": {
            "lb": -30,
            "ub": 30
        }
    },
    "line_4/angle_difference": {
        "var": {
            "lb": -30,
            "ub": 30
        }
    }
}
```

Figure 22: An example of GRG operation constraints.

4 Networks

A GRG network is a collection of network components, along with a list of component assignments. When a network component is assigned, its value in the *assignments* block of the GRG document supersedes its value in the *network* block. The GRG schema for a network is shown below.

```
                                   GRG schema: network
"network": {
    "type": "object",
    "required": ["id", "type", "subtype", "per_unit", "components"],
    "properties": {
        "id": {"type": "string"},
        "type": {"enum": ["network"]},
        "subtype": {"enum": ["node_breaker", "bus_breaker", "bus_branch"]},
        "per_unit": {"type": "boolean"},
        "description": {"type": "string"},
        "components": {"$ref": "#/network/network_components"}
        "assignments": {"$ref": "#/network/network_assignments"},
    },
    "additionalProperties": true
}
```

The attribute `type` identifies the type of the object (i.e., a `network`). `subtype` indicates topology type, which defines the level of detail in descriptions of connections between components; `description` is an optional description field, and `per_unit` indicates whether network component values are expressed in per unit or nominal value.

A network is organized as a set of substations connected via transmission lines. Together these comprise the `network_components` block, as shown in the following schema fragment:

```
GRG schema: network components
"network_components": {
    "patternProperties": {
        ".*": {
            "oneOf": [
                {"$ref": "#/network_components/substation"},
                {"$ref": "#/network_components/ac_line"}
            ]
        }
    }
}
```

4.1 Substation

A *substation* is a collection of equipment located at a the same physical site and belonging to one Transmission System Operator (TSO). It is composed of several voltage levels and transformers. Its schema is as follows.

```
GRG schema: substation
"substation": {
    "type"    : "object",
    "required": ["type", "id", "substation_components"],
    "properties": {
        "type"       : {"enum": ["substation"]},
        "subtype":{"type": "string"},
        "description": {"type": "string"},
        "id"         : {"type": "string"},
        "country"    : {"type": "string"},
        "TSO"        : {"type": "string"},
        "substation_components": {
            "patternProperties": {
                ".*": {
                    "oneOf": [
                        {"$ref": "#/network_components/voltage_level"},
                        {"$ref": "#/network_components/two_winding_transformer"}
                        {"$ref": "#/network_components/three_winding_transformer"}
                    ]
                }
            }
        }
    }
}
```

The attribute `type` identifies the type of the object (i.e., a `substation`). The attributes `country` and `TSO` indicate the country in which the substation is located, along with its Transmission System Operator. Substation components are listed in the creatively-named `substation_components` object.

4.2 Voltage Level

A *voltage level* is a collection of equipment located in the same substation at the same nominal voltage value. Its schema is as follows.

GRG schema: voltage_level

```
"voltage_level": {
    "type"    : "object",
    "required": ["type", "id", "voltage", "voltage_points"],
    "properties": {
        "type"        : {"enum": ["voltage_level"]},
        "subtype": {"type": "string"},
        "description": {"type": "string"},
        "id"          : {"type": "string"},
        "voltage_points": {  "type": "array", "items": { "type": "string" }, "minItems": 1 },
        "voltage": {
            "type"        : "object",
            "required"    : ["nominal_value", "upper_limit", "lower_limit"],
            "additionalProperties": true,
            "properties": {
                "nominal_value": {"type": "number"},
                "upper_limit"  : {"type": "number"},
                "lower_limit"  : {"type": "number"}
            }
        },
        "voltage_level_components": {
          "patternProperties": {  ".*": {
             "oneOf": [
                {"$ref": "#/network_components/bus"},
                {"$ref": "#/network_components/shunt"},
                {"$ref": "#/network_components/generator"},
                {"$ref": "#/network_components/synchronous_condenser"}
                {"$ref": "#/network_components/load"},
                {"$ref": "#/network_components/switch"}
             ]
         }}
        }
    }
}
```

The attribute `type` identifies the type of the object (i.e., a `voltage_level`). `voltage` contains the nominal voltage V_{nom} along with voltage limits $[V^L, V^U]$ for all components contained in the voltage level. Since GRGv1.5, we unify all network components to be connected by voltage points, and voltage points are assume to be global and unique. To allow ease of parsing, we require all voltage levels to explicitly list all its voltage point labels in `voltage_points` starting from GRG v1.6. The voltage level components are listed in `voltage_components`.

Table 17 describes the voltage level element.

GRG name	Parameter	unit
`voltage→nominal_value`	V_{nom}	KiloVolt (KV)
`voltage→lower_limit`	V^L	KiloVolt (KV)
`voltage→upper_limit`	V^U	KiloVolt (KV)

Table 17: Voltage Level description.

5 Market

A `market` GRG block describe various costs information to the network component. In previous GRG versions, we use `costs` block to describle operational costs for a component. We allow multiple polynomial cost functions for each component, by defining the costs to be a JSON array of arrays. This can become overly complex to parse for users, especially when most benchmarks only have single polynomial function per component. In GRGv4.0, we use `operational_costs` JSON block, a more simplified structure, to describe operational costs for a component. Since the market block allows additional costs structure, the old costs structure (prior to GRGv4.0) will be automatically allowed, and migration should not be an issue. Other costs structure, e.g. investment costs for expansion planning problems, or startup/shutdown costs for unit committment problems can also be defined and extended in the market GRG block.

```
voltage_level definition
"market": {
    "type": "object",
    "additionalProperties": true,
    "properties": {
        "operational_costs": {
            "patternProperties": {
                ".*": {
                    "type": "object",
                    "required": ["type","input","coefficients"],
                    "additionalProperties": true,
                    "properties": {
                        "type": {
                            "enum": ["polynomial"]
                        },
                        "input": {
                            "$ref": "#/values/basic_values/grg_pointer"
                        },
                        "coefficients": {
                            "type": "array",
                            "items": { "$ref": "#/values/basic_values/abstract_value"},
                            "minItems": 1
                        }
                    }
                }
            }
        }
    }
}
```

Each item in the operational_costs block is named with a GRG pointer in the input field to its associated network element. The coefficient field is a JSON array defining the associated cost coefficients: $a_{n-1}, \ldots, a_0$, generating the polynomial:

$$a_{n-1}\, x_i^{n-1} + \ldots + a_1\, x_i + a_0$$

Since the cost coefficients are dependent on the units of the arguments array, they will need to be re-computed if the units of the arguments array changed (e.g. during per-unit operations).

6 Units

A units object details the units adopted for each physical quantity. These may differ from unit associated to the description of various components in the previous sections. The JSON schema of units is defined below.

```
GRG schema: units
    "units": {
        "type": "object",
        "required": ["voltage", "current", "angle", "reactive_power", "active_power",
                     "impedance","resistance", "reactance", "conductance", "susceptance", "time"],

        "properties": {
            "voltage": {"enum": ["volt", "kilo_volt", "mega_volt", "pu"]},
            "current": {"enum": ["ampere", "kilo_ampere", "mega_ampere", "pu"]},
            "angle"  : {"enum": ["degree", "radian"]},
            "active_power": {"enum": ["watt", "kilo_watt", "mega_watt", "pu"]},
            "reactive_power": {"enum": ["volt_ampere_reactive", "mega_volt_ampre_reactive", "pu"]},
            "impedance": {"enum": ["ohm", "pu"]},
            "resistance": {"enum": ["ohm", "pu"]},
            "reactance": {"enum": ["ohm", "pu"]},
            "conductance": {"enum": ["siemens", "pu"]},
            "susceptance": {"enum": ["siemens", "pu"]},
            "time": {"enum": ["seconds", "minutes", "hours", "days", "months", "years"]}
        }
    }
```

7 Time Series Data

A `network_time_series` object describes time-varying data for network components. The JSON schema of `network_time_series` is defined below.

```
GRG schema: network_time_series
"network_time_series" :{
    "type" : "object",
    "patternProperties" :{
        ".*" :{
            "type" : "object",
            "required" : ["step_duration", "steps", "step_duration_units"],
            "additionalProperties": true,
            "properties" :{
                "step_duration" : {"$ref": "#/values/basic_values/extended_positive_number"},
                "steps": {"$ref": "#/values/basic_values/extended_positive_number"},
                "step_duration_units": {"enum": [ "microseconds",  "milliseconds", "seconds",
                                                   "minutes", "hours", "days",  "months",   "years"]},
                "assignments": {
                    "type": "object",
                    "patternProperties":{
                        ".*":{
                            "type": "array",
                            "items":{
                                "oneOf": [ {"$ref": "#/values/basic_values/extended_number"},
                                                {"enum": ["off", "on"]}]
                            },
                            "minItems" : 1
                        }
                    }
                },
                "external_file" :{
                    "type": "object",
                    "required" : ["file_name", "ids"],
                    "additionalProperties": true,
                    "properties" :{
                        "file_name" : { "type" : "string" },
                        "ids" : {
                            "type": "array",
                            "items": { "type": "string" },
                            "minItems" : 1
                        }
                    }
                }
            }
        }
    },
    "additionalProperties": true
}
```

The `patternProperties` field specifies that zero or more sets of time series data are allowed to be specified. For each of the time series data block, we require fields `steps`, `step_duration`, `step_duration_units`. `steps` is used to specify the number of time series data for each component within the time series data block. `step_duration` and `step_duration_units` are used to specify the time interval and units for each adjacent pair of time series data. Finally, `assignments` block list all the time series data in JSON array format for network components. Figure 23 shows an example of network time series represented in our GRG format. In this example, we specify active and reactive load data for load 1 and active dispatch data for generator 2. The data consists of three data points (`steps`) at a one hour interval (`step_duration` & `step_duration_units`).

In GRGv4.0, we further allow users to store potentially large time series data into a CSV file, instead of storing all data in the `assignments` block. The `external_file` JSON block is used to record the information for the external CSV file. `file_name` records the full path (or the web accessible path) to the CSV data file. `ids` is a JSON array block for recording all the network components with time series data in the external file, with the i$^{\text{th}}$ component in the array corresponds to the data at the i$^{\text{th}}$ row of the CSV data file. In other words, the data at the i$^{\text{th}}$ row and j$^{\text{th}}$ column in the CSV file represents the data at the j$^{\text{th}}$ time step for the i$^{\text{th}}$ network component in the `ids` array.

Example: Network time series
```
    "network_time_series": {
        "one_year_data": {
            "step_duration": 1,
            "steps": 3,
            "step_duration_units": "hours",
            "assignments": {
                "load_1/demand/active": [ 0.94, 0.91, 0.88 ],
                "load_1/demand/reactive": [ -0.15, -0.15, -0.14],
                "gen_2/output/active": [ 0.14, 0.12, 0.101]
            },
            "external_file" :{
                "file_name": "/home/users/data1.csv",
                "ids": ["load_3/demand/active", "load_3/demand/reactive", "gen_3/output/active"  ]
            }
         }
    }
```

Figure 23: An example of network time series block

8 Stochastic Time Series Data

A `stochastic_time_series` object describes stochastic time-varying data for network components. The object extends the `network_time_series` defined above and further introduces scenarios and their probabilities. The JSON schema of `stochastic_time_series` is defined below.

GRG schema: stochastic_time_series
```
    "stochastic_time_series":{
        "type" : "object",
        "patternProperties" :{
            ".*" :{
                "type" : "object",
                "required" : ["step_duration", "steps", "step_duration_units",
                              "var", "scenario", "scenario_probability"],
                "additionalProperties": true,
                "properties" :{
                    "step_duration" : {"$ref": "#/values/basic_values/extended_positive_number"},
                    "steps": {"$ref": "#/values/basic_values/extended_positive_number"},
                    "step_duration_units": {"enum": [
                                                "microseconds",
                                                "milliseconds",
                                                "seconds",
                                                "minutes",
                                                "hours",
                                                "days",
                                                "months",
                                                "years"
                                                ]},
                    "var" : { "type" : "string"},
                    "scenario": {
                        "type": "object",
                        "patternProperties":{
                            ".*":{
                                "type": "array",
                                "items":{
                                    "oneOf": [
                                              {"$ref": "#/values/basic_values/extended_number"},
                                              {"enum": ["off", "on"]}]
                                },
                                "minItems" : 1
                            }
                        }
                    },
                    "scenario_probability": {
                        "type": "object",
                        "patternProperties":{
                            ".*":{
                                    "type": "string"
                            }
                        }
                    }
                }
            }
        }
    }
```

The `patternProperties` field specifies that zero or more sets of stochastic time series data are allowed to be specified. Similar to the time series data block defined in previous section, we require fields `steps`, `step_duration`, and `step_duration_units` for each of the stochastic time series data block. We further require fields `var`, `scenario`, and `scenario_probability` to describe stochastic data.

`var` is used specify the network component corresponding to the stochastic data of interests. `scenario` is used to describe all the time-varying scenarios for the specified component. Each scenario is labeled with a name and specified by the time series data in JSON array format. To specify the probability for each scenario, we use `scenario_probability` to list the probabilities for all of the scenarios. The probability data will be represented as string data in percentages, and ending with the '%' character. `steps` is used to specify the number of time series data for the scenarios defined in `scenario`. `step_duration` and `step_duration_units` are used to specify the time interval and units for each adjacent pair of time series data across all scenarios.

Figure 24 shows an example of stochastic network time series represented in our GRG format. In this example, we specify the stochastic time series data for the active demand for load 1. The data consists of three scenarios: scenario_1, scenario_2, and scenario_3, each with probability 50%, 40%, and 10% respectively. The number of data points in each scenario is four (`steps`), at a one hour interval (`step_duration` & `step_duration_units`).

```
Example: Stochastic time series
    "stochastic_time_series":{
        "stochastic_dataSet_1": {
            "step_duration": 1,
            "steps": 4,
            "step_duration_units": "hours",
            "var": "load_1/demand/active",
            "scenario": {
                "scenario_1": [0.9415429966400392,
                               0.914192766758504,
                               0.8891846836347795,
                               0.8789094591488987
                               ],
                "scenario_2": [1.41231449496,
                               1.37128915014,
                               1.33377702545,
                               1.31836418872
                               ],
                "scenario_3": [1.88308599328,
                               1.82838553352,
                               1.77836936727,
                               1.7578189183
                               ]
            },
            "scenario_probability": {
                "scenario_1": "50%",
                "scenario_2": "40%",
                "scenario_3": "10%"
            }
        }
    }
```

Figure 24: An example of stochastic time series block

9 Time Series Constraints

A `time_series_constraints` object describes constraints with temporal time steps (e.g. ramping constraints for unit committment problems). It can also be used to provide extra information for the time series data or the stochastic time series data.

GRG schema: time_series_constraints

```
"time_series_constraints" :{
    "type" : "object",
    "patternProperties" :{
        ".*" :{
            "type" : "object",
            "required" : ["units", "assignments"],
            "additionalProperties": true,
            "properties" :{
                "units": { "type" : "string"},
                "assignments": { "$ref": "#/network/network_assignments"}
            }
        }
    },
    "additionalProperties": true
}
```

The `patternProperties` field specifies that zero or more sets of time series constraints are allowed to be specified. In `time_series_constraints`, we require fields `units` and `assignments`. Field `units` is used specify the unit for the time series constraints. For example, the ramp up/down rate unit can be specified in `units`. `assignments` block provides a list of assignments, where each assignment provides constraint data for each network component.

Figure 25 shows an example of a time series constraint block represented in the GRG format. In this example, we specify the ramping constraints for generator `gen_1`. The ramp up rate for the generator are 50 (active) and 30 (reactive) MVA/hour. Similarly, the ramp down rate are 40 (active) and 20 (reactive) MVA/hour. Again, the schema allows users to customize and extend the GRG format. Users are encouraged to create other JSON blocks to describe additional data.

Example: Time series constraint

```
"time_series_constraints": {
        "ramp_constraints": {
            "units" : "MVA/hours",
            "assignments": {
                "gen_1/output/active/ramp_up_rate": 50,
                "gen_1/output/active/ramp_down_rate": -40,
                "gen_1/output/reactive/ramp_up_rate": 30,
                "gen_1/output/reactive/ramp_down_rate": -20
            }
        }
}
```

Figure 25: An example of a time series constraint block

10 Contingencies

The `contingencies` object describes a set of contingency scenarios. Each scenario can be either defined as a customized list of components specified by their GRG ID, or defined as a traditional contingency requirement (e.g. N-1 contingency for transmission line). The JSON schema of `contingencies`, including the two contingency types `custom_contingency` and `traditional_contingency`, are defined below.

GRG schema:contingencies

```
    "contingencies" : {
        "type": "object",
        "additionalProperties": true,
        "patternProperties": {
            ".*" :{
                "oneOf": [
                          {
                          "$ref": "#/contingencies/custom_contingency"
                          },
                          {
                          "$ref": "#/contingencies/traditional_contingency"
                          }
                          ]
            }
        },
        "custom_contingency" : {
            "type" : "object",
            "required" : ["failure-list"],
            "additionalProperties": true,
            "properties" :{
                "failure-list" : {   "type": "array",
                                     "items" : {
                                         "type" : "string"
                                     },
                                    "minItems" : 1
                }
            }
        },
        "traditional_contingency" : {
            "type" : "object",
            "required" : ["criteria", "component"],
            "additionalProperties": true,
            "properties" :{
                "criteria" : { "type": "string"},
                "component" : {   "type": "string" }
            }
        }
    }
```

The `patternProperties` field specifies that zero or more sets of contingency scenarios are allowed to be specified. Each contingency scenario will be either defined as a customized contingency scenario (`custom_contingency`), or a classical contingency scenario (`traditional_contingency`). A classical contingency scenario requires fields `criteria` and `component`. `component` is used to specify the type of network components (e.g. generator, ac_line, or bus) for the contingency scenario. `criteria` is used to describe the requirement on the type of equipments specified in `component` (e.g. N-1 requirement). A customized contingency scenario requires field `failure-list` to specify a list (in JSON array) of contingency components by their GRG IDs. Extra fields and data are allowed and recommended to be added to customized scenario to provide more data and description on the listed contingency components.

Figure 26 shows an example of four contingency scenarios represented in our GRG format. In this example, we specify two customized contingency scenarios and two classical contingency scenarios. The first scenario contains a `failure-list` of three lines (`line_1`, `line_2`, and `line_3`), and the second scenario contains a `failure-list` of one generator, one bus, and one transformer. The third and fourth scenarios describe two classical contingencies: the N-2 contingency for AC transmisison line, and the N-1 contingency for generators.

11 Group block: Zones, Areas, and owners

The `groups` object records zones, areas, and ownership information of a transmission network. The object is useful in providing information for network grouping, as well as information for classification. An object in the `groups` block will be either a zone record, an area record, or an ownership record (or a user-defined extended record). The JSON schema for `groups`, as well as the JSON schema for `zone`, `area`, and `owner` are defined below.

GRG schema:groups

```
"groups" : {
    "type": "object",
    Example: Contingency scenarios
    "contingencies": {
        "scenario_1": {
            "failure-list" : ["line_1", "line_2", "line_3"]
        },
        "scenario_2": {
            "failure-list" : ["gen_1", "bus_75", "transformer_84"]
        },
        "scenario_3": {
            "criteria": "N-2",
            "component": "ac_line"
        },
        "scenario_4": {
            "criteria": "N-1",
            "component": "generator"
        }
    }
```

Figure 26: An example of contingencies block

```
"additionalProperties": true,
"patternProperties": {
    ".*" :{
        "oneOf": [
                    {
                    "$ref": "#/groups/zone"
                    },
                    {
                    "$ref": "#/groups/area"
                    },
                    {
                    "$ref": "#/groups/owner"
                    }
                    ]
    }
},
"zone" : {
    "type" : "object",
    "required" : ["type", "name", "source_id", "component_ids"],
    "additionalProperties": true,
    "properties" :{
        "type" : { "enum" : ["zone"]},
        "name" : { "type" : "string"},
        "source_id" : { "type" : "string"},
        "component_ids" : {
            "type": "array",
            "items" : {"type" : "string"},
            "minItems" : 1
        }
    }
},
"area" : {
    "type" : "object",
    "required" : ["type", "name", "ptol", "source_id", "component_ids"],
    "additionalProperties": true,
    "properties" :{
        "type" : { "enum" : ["area"]},
        "name" : { "type" : "string"},
        "ptol" : {"$ref": "#/values/basic_values/extended_number"},
        "source_id" : { "type" : "string"},
        "component_ids" : {
            "type": "array",
            "items" : {"type" : "string"},
            "minItems" : 1
        }
    }
    },
    "owner" : {
        "type" : "object",
        "required" : ["type", "name", "source_id", "component_ids"],
        "additionalProperties": true,
        "properties" :{
            "type" : { "enum" : ["owner"]},
            "name" : { "type" : "string"},
            "source_id" : { "type" : "string"},
            "component_ids" : {
                "type": "array",
                "items" : {"type" : "string"},
                "minItems" : 1
            }
        }
    }
}
```

The `patternProperties` field specifies that zero or more sets of group records are allowed to be specified. Each record will be either defined as a zone record `zone`, an area record `area`, or an ownership record `owner`. All of these records require `type`, `name`, `source_id`, and `component_ids`. An area record also require `ptol`. `type` identified if the record is a zone, area, or owner record. `name` and `source_id` gives the name and identifier of the record. `component_ids` is a JSON array listing the network components (in GRG IDs) associating to the specific record. Finally, `ptol` is the area interchance tolerance (in MW) specifically design for the area record. Extra fields and data are again allowed and recommended to be added to these records to provide more data and description. Figure 27 shows an example of a zone, an area, and an ownership record represented in the GRG format.

Example: Group block

```
"groups":{
    "zone_01":{
        "type": "zone",
        "name": "SOUTHWST",
        "source_id": "10",
        "component_ids":["bus_1", "bus_3", "gen_1"]
     },
     "area_02":{
        "type": "area",
        "name": "MEXICO",
        "ptol": 5.0,
        "source_id": "20",
        "component_ids":["bus_1", "gen_2"]
     },
     "owner_01":{
         "type": "owner",
         "name": "PG&E",
         "source_id": "20",
         "component_ids":["line_1", "line_3", "transformer_4"]
    }
}
```

Figure 27: An example of group block

12 GRG Document

A *GRG document* is defined as a collection of JSON blocks, and includes a `network` block, a `mapping` block, a `market` block, a `units` block, an `operation_constraints` block, a `network_time_series` block, a `time_series_constraints` block, a `contingencies` block, a `stochastic_time_series` block, and a `groups` block. Its schema is as follows.

GRG schema: GRG document

```
{
    "type": "object",
    "required": ["grid_version", "network", "units"],
    "additionalProperties": true,

    "properties": {
        "grg_version": {"type": "string"},
        "description": {"type": "string"},
        "network": {"$ref": "#/network"},
        "mappings": {"$ref": "#/mappings"},
        "market": {"$ref": "#/market"},
        "units": {"$ref": "#/units"},
        "operation_constraints": {"$ref": "#/operation_constraints"},
        "network_time_series" : {"$ref": "#/network_time_series"},
        "time_series_constraints" : {"$ref": "#/time_series_constraints"},
        "contingencies" : {"$ref": "#/contingencies"},
        "stochastic_time_series" : {"$ref": "#/stochastic_time_series"},
        "groups" :{"$ref": "#/groups"}
    }
}
```

13 Network Topology

A network may be represented in one of three different topologies, from finer to coarser level of detail: node-breaker, bus-breaker, or bus-branch. Table 18 summarizes these representations, indicating which ones capture the voltage level topology (i.e., whether all the elements and their connections are physical ones) or not (i.e., merely logical connections); which switch types are captured, and the representation of buses. We will now discuss each representation in detail.

	Node-breaker	Bus-breaker	Bus-branch
Topology	yes	yes	no
Breakers	yes	yes	no
Disconnectors	yes	no	no
Bus Type	busbar	logical bus	bus

Table 18: Network Topologies.

13.1 Node-Breaker Topology

A node-breaker topology contains the highest level of detail for a network. All components are physical elements, including busbar sections, breakers, and disconnectors. In this topology, a voltage level is a collection of network components and connection *nodes*. Nodes are logical connection points labeled with values in $\mathbb{N} \cup \{0\}$. Each component is directly connected to one node (if it is a bus, shunt, generator, or load), or to two nodes (if it is an AC line, transformer, or switch). The connection of a network component to a node is described in its `link` attribute (or `link_1` and `link_2` attributes for a connector). The set of nodes in a voltage level is described implicitly: It is the union of `link` values of all network components in that voltage level. Figure 28 (a) shows a voltage level in the node-breaker topology where breakers are illustrated as white squares, and disconnectors as pairs of white circles.

13.2 Bus-Breaker Topology

A network in bus-breaker form also represents a voltage level as a collection of components connected through nodes. However, the bus-breaker representation contains no disconnectors. Algorithm 1 describes the procedure for constructing a bus-breaker topology from a network in node-breaker form. The algorithm accepts a node-breaker network $\mathcal{N}$ as input. Lines (1–2) call the procedure Transform-VoltageLevel for each voltage level $\mathcal{V}$ in the network $\mathcal{N}$. The result of this operation is to (1) merge busbars of the voltage level which are connected by closed disconnectors, (2) remove all disconnectors from the voltage level, and (3) remove components that have no path to a logical bus. Lines

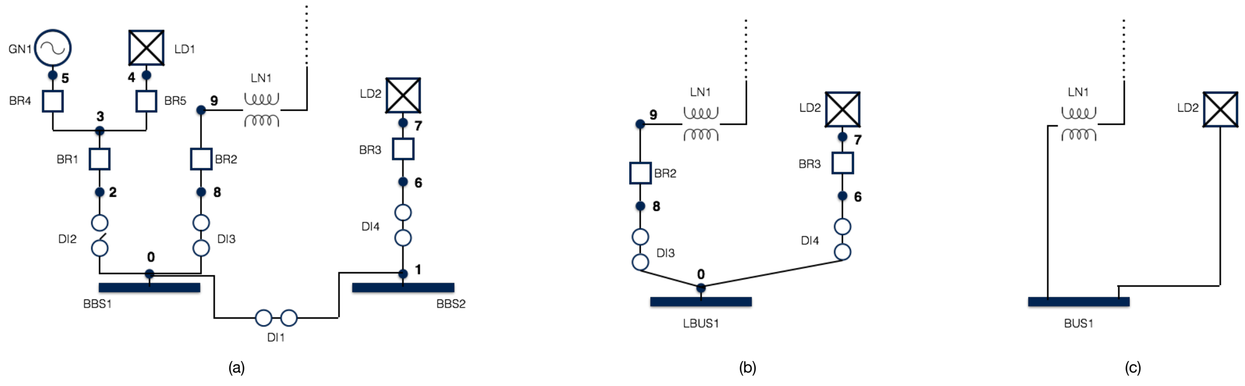

Figure 28: Illustration of a voltage level in node-breaker (a), bus-breaker (b), and bus-branch (c) topologies.

Algorithm 1: NODE-BREAKER TO BUS-BREAKER($\mathcal{N}$)

```
1 foreach voltage level $\mathcal{V} \in \mathcal{N}$ do
2     Transform-VoltageLevel($\mathcal{V}$);
3 foreach line and transformer $l \in \mathcal{L}_{\mathcal{N}} \cup \mathcal{T}_{\mathcal{N}}$ do
4     if $\nexists$ paths through close switches from $l$ to some $b_1 \in \mathcal{B}_{\mathcal{V}_1}$ and $b_2 \in \mathcal{B}_{\mathcal{V}_2}$ then
5         Remove $l$ from $\mathcal{V}$;
```

(6–12) show greater detail for the removal process of a set of disconnectors $\mathcal{D}_{\mathcal{V}}$ of the voltage level $\mathcal{V}$. A disconnector is removed from the network by linking all voltage level components attached to its node on side 2 (`link_2`) to its node on side 1 (`link_1`) (lines 9–11). This operation may result in multiple busbars connected to the same node. In Figure 28 (a), for instance, busbars BBS1 and BBS2 will be connected to node 0 when disconnector DI1 is removed. The effect of the loop in lines 13–15 is to merge such busbar sets into single logical buses. This is done by preserving one busbar of the set B, changing its type to a logical bus, and removing all other busbars in B.

The effect of executing lines 16–18 is to remove all components of the voltage level for which there is no path through closed switches or non-assigned breakers to a logical bus of the voltage level. For instance, in Figure 28 (a), the breakers BR1, BR4, B5, the generator GN1, and the load LD1 have no closed path to a voltage level bus, since disconnector DI2 is open. Thus, these components are removed from the voltage level. The result of this operation is illustrated in Figure 28 (b).

Finally, in lines (3–5) of Algorithm 1, all lines and transformers (connectors) of the network are processed. As was done for voltage levels, a connector is removed if it has no closed path to logical buses on both sides.

Figure 28 (b) illustrates an example voltage level in the bus-breaker topology obtained from the node-breaker topology of Figure 28 (a).

13.3 Bus-Branch Topology

A network in bus-branch form differs from one in a node- or bus-breaker form in that switches are not represented, and all network components are directly connected to a bus. Thus, the bus-branch topology has no concept of connection nodes.

A bus-branch topology can be obtained from a node- or a bus-breaker topology by applying a variation of Algorithm 1. Rather than processing disconnectors (line 6), all switches of the voltage level are processed. Finally, after the algorithm is executed, the `link` attribute of each generator, load, and shunt, and the `link_1` and `link_2` attributes of lines and transformers are updated to take the ID of the bus to which they are connected.

Figure 28 (c) illustrates an example voltage level in the bus-branch topology, obtained from the bus-breaker topology of Figure 28 (b).

```
Procedure Transform-VoltageLevel($\mathcal{V}$)
6  foreach disconnectors $d \in \mathcal{D}_\mathcal{V}$ do
7      $n_1 \leftarrow d[link\_1]$;
8      $n_2 \leftarrow d[link\_2]$;
9      foreach component $c \in \mathcal{V}$ s.t. $c$ is connected to $n_2$ do
10         Let link be link_i if $c$ is a switch, line, or transformer, and $link\_i \neq n_2$;
11         $c[link] \leftarrow n_1$;
12     Remove $d$ from $\mathcal{V}$ ;
13 foreach busbar $b \in \mathcal{B}_\mathcal{V}$ do
14     Let $B = \{b' \in \mathcal{B}_\mathcal{V} \ : \ b'[link] = b[link]\}$ ;
15     Remove all $b' \in B \setminus \{b\}$ ;
16 foreach component $c \in \mathcal{V}$ do
17     if $\nexists$ path through close switches from $c$ to some $b \in \mathcal{B}_\mathcal{V}$ then
18         Remove $c$ from $\mathcal{V}$;
```

14 Per-Unit Transformations

This section describes all per-unit transformations for network components.

AC Line

The per-unit transformation of an AC line (see also Figure 1) is shown in Table 19. The nominal current magnitude I_{nom} of the AC line is defined as:

$$I_{\text{nom}} = \frac{S_{\text{nom}}}{V_{\text{nom}}}, \tag{28}$$

where V_{nom} is voltage magnitude on either side of the line, and S_{nom} is the nominal power magnitude of the network. The nominal impedance of the AC line is defined as:

$$Z_{\text{nom}} = \frac{(V_{\text{nom}})^2}{S_{\text{nom}}} \tag{29}$$

GRG name	Parameter	Per-unit transformation
`impedance→resistance`	R	$\frac{1}{Z_{\text{nom}}} \cdot R$
`impedance→reactance`	X	$\frac{1}{Z_{\text{nom}}} \cdot X$
`shunt_1→conductance`	G_1	$Z_{\text{nom}} \cdot G_1$
`shunt_1→susceptance`	B_1	$Z_{\text{nom}} \cdot B_1$
`shunt_2→conductance`	G_2	$Z_{\text{nom}} \cdot G_2$
`shunt_2→susceptance`	B_2	$Z_{\text{nom}} \cdot B_2$
`current_limits_1[k]→min`	$I_{1_k}^{min}$	$\frac{1}{I_{\text{nom}}} \cdot I_{1_k}^{min}$
`current_limits_1[k]→max`	$I_{1_k}^{max}$	$\frac{1}{I_{\text{nom}}} \cdot I_{1_k}^{max}$
`current_limits_2[k]→min`	$I_{2_k}^{min}$	$\frac{1}{I_{\text{nom}}} \cdot I_{2_k}^{min}$
`current_limits_2[k]→max`	$I_{2_k}^{max}$	$\frac{1}{I_{\text{nom}}} \cdot I_{2_k}^{max}$
`thermal_limits_1[k]→min`	$S_{1_k}^{min}$	$\frac{1}{S_{\text{nom}}} \cdot S_{1_k}^{min}$
`thermal_limits_1[k]→max`	$S_{1_k}^{max}$	$\frac{1}{S_{\text{nom}}} \cdot S_{1_k}^{max}$
`thermal_limits_2[k]→min`	$S_{2_k}^{min}$	$\frac{1}{S_{\text{nom}}} \cdot S_{2_k}^{min}$
`thermal_limits_2[k]→max`	$S_{2_k}^{max}$	$\frac{1}{S_{\text{nom}}} \cdot S_{2_k}^{max}$

Table 19: AC line per-unit transformations.

DC Line

The per-unit transformation of an DC line is shown in Table 20. The nominal current magnitude I_{nom} of the DC line is defined as:

$$I_{\text{nom}} = \frac{S_{\text{nom}}}{V_{\text{nom}}}, \tag{30}$$

where V_{nom} is voltage magnitude on either side of the line, and S_{nom} is the nominal power magnitude of the network. The nominal resistance of the DC line is defined as:

$$R_{\text{nom}} = \frac{(V_{\text{nom}})^2}{S_{\text{nom}}} \tag{31}$$

GRG name	Parameter	Per-unit transformation
`resistance`	R	$\frac{1}{R_{nom}} \cdot R$
`losses_1→min`	l_{min}	$\frac{1}{S_{nom}} \cdot l_{\text{min}}$
`losses_1→max`	l_{max}	$\frac{1}{S_{nom}} \cdot l_{\text{max}}$
`losses_1→c_0`	c_0	$\frac{1}{S_{nom}} \cdot c_0$
`losses_1→c_1`	c_1	$\frac{1}{S_{nom}} \cdot c_1$
`losses_2→min`	l_{min}	$\frac{1}{S_{nom}} \cdot l_{\text{min}}$
`losses_2→max`	l_{max}	$\frac{1}{S_{nom}} \cdot l_{\text{max}}$
`losses_2→c_0`	c_0	$\frac{1}{S_{nom}} \cdot c_0$
`losses_2→c_1`	c_1	$\frac{1}{S_{nom}} \cdot c_1$
`output_1→reactive`	Q	$\frac{1}{S_{nom}} \cdot Q$
`output_2→reactive`	Q	$\frac{1}{S_{nom}} \cdot Q$
`current_limits_1[k]→min`	$I_{1_k}^{min}$	$\frac{1}{I_{nom}} \cdot I_{1_k}^{min}$
`current_limits_1[k]→max`	$I_{1_k}^{max}$	$\frac{1}{I_{nom}} \cdot I_{1_k}^{max}$
`current_limits_2[k]→min`	$I_{2_k}^{min}$	$\frac{1}{I_{nom}} \cdot I_{2_k}^{min}$
`current_limits_2[k]→max`	$I_{2_k}^{max}$	$\frac{1}{I_{nom}} \cdot I_{2_k}^{max}$
`thermal_limits_1[k]→min`	$S_{1_k}^{min}$	$\frac{1}{S_{nom}} \cdot S_{1_k}^{min}$
`thermal_limits_1[k]→max`	$S_{1_k}^{max}$	$\frac{1}{S_{nom}} \cdot S_{1_k}^{max}$
`thermal_limits_2[k]→min`	$S_{2_k}^{min}$	$\frac{1}{S_{nom}} \cdot S_{2_k}^{min}$
`thermal_limits_2[k]→max`	$S_{2_k}^{max}$	$\frac{1}{S_{nom}} \cdot S_{2_k}^{max}$

Table 20: DC line per-unit transformations.

Two winding transformer

For a two winding transformer (see Figure 5), the per-unit transformation is defined in Table 21. The nominal current magnitudes I^1_{nom} on side 1 and I^2_{nom} on side 2 are defined as:

$$I^1_{\text{nom}} = \frac{S_{\text{nom}}}{V^1_{\text{nom}}} \tag{32}$$

$$I^2_{\text{nom}} = \frac{S_{\text{nom}}}{V^2_{\text{nom}}}, \tag{33}$$

where V^1_{nom} and V^2_{nom} are the voltage magnitudes on sides 1 and 2 of the transformer. The nominal impedance magnitude on side 2 (low voltage side) of the transformer is defined as:

$$Z_{\text{nom}} = \frac{(V^2_{\text{nom}})^2}{S_{\text{nom}}} \tag{34}$$

GRG name	Parameter	Per-unit transformation
tap_changer→impedance→resistance	R_k	$\frac{1}{Z_{\text{nom}}} \cdot R_k$
tap_changer→impedance→reactance	X_k	$\frac{1}{Z_{\text{nom}}} \cdot X_k$
tap_changer→shunt→conductance	G_k	$Z_{\text{nom}} \cdot G_k$
tap_changer→shunt→susceptance	B_k	$Z_{\text{nom}} \cdot B_k$
tap_changer→transform→tap_ratio (T model)	$(\frac{V_2^{\text{nom}}}{V_1^{\text{nom}}})r_k$	r_k
tap_changer→transform→tap_ratio (PI model)	$(\frac{V_1^{\text{nom}}}{V_2^{\text{nom}}})r_k$	r_k
tap_changer→transform→angle_shift	δ_k	$\frac{\pi}{180} \cdot \delta_k$
tap_changer→steps[k]→impedance→resistance	R_k	$\frac{1}{Z_{\text{nom}}} \cdot R_k$
tap_changer→steps[k]→impedance→reactance	X_k	$\frac{1}{Z_{\text{nom}}} \cdot X_k$
tap_changer→steps[k]→shunt→conductance	G_k	$Z_{\text{nom}} \cdot G_k$
tap_changer→steps[k]→shunt→susceptance	B_k	$Z_{\text{nom}} \cdot B_k$
tap_changer→steps[k]→transform→tap_ratio (T model)	$(\frac{V_2^{\text{nom}}}{V_1^{\text{nom}}})r_k$	r_k
tap_changer→steps[k]→transform→tap_ratio (PI model)	$(\frac{V_1^{\text{nom}}}{V_2^{\text{nom}}})r_k$	r_k
tap_changer→steps[k]→transform→angle_shift	δ_k	$\frac{\pi}{180} \cdot \delta_k$
current_limits_1[k]→min	$I_{1_k}^{min}$	$\frac{1}{I_{\text{nom}}} \cdot I_{1_k}^{min}$
current_limits_1[k]→max	$I_{1_k}^{max}$	$\frac{1}{I_{\text{nom}}} \cdot I_{1_k}^{max}$
current_limits_2[k]→min	$I_{2_k}^{min}$	$\frac{1}{I_{\text{nom}}} \cdot I_{2_k}^{min}$
current_limits_2[k]→max	$I_{2_k}^{max}$	$\frac{1}{I_{\text{nom}}} \cdot I_{2_k}^{max}$
thermal_limits_1→min	$S_{1_k}^{min}$	$\frac{1}{S_{\text{nom}}} \cdot S_{1_k}^{min}$
thermal_limits_1→max	$S_{1_k}^{max}$	$\frac{1}{S_{\text{nom}}} \cdot S_{1_k}^{max}$
thermal_limits_2→min	$S_{2_k}^{min}$	$\frac{1}{S_{\text{nom}}} \cdot S_{2_k}^{min}$
thermal_limits_2→max	$S_{2_k}^{max}$	$\frac{1}{S_{\text{nom}}} \cdot S_{2_k}^{max}$

Table 21: Two winding transformer per-unit transformations.

Three winding transformer

For a three winding transformer (see Figure 8), the per-unit transformation is defined in Table 22. The nominal current magnitudes I^1_{nom}, I^2_{nom}, and I^3_{nom} on side 1, 2, and 3 are defined as:

$$I^1_{\text{nom}} = \frac{S_{\text{nom}}}{V^1_{\text{nom}}}, \tag{35}$$

$$I^2_{\text{nom}} = \frac{S_{\text{nom}}}{V^2_{\text{nom}}}, \tag{36}$$

$$I^3_{\text{nom}} = \frac{S_{\text{nom}}}{V^3_{\text{nom}}} \tag{37}$$

$$\tag{38}$$

where V^1_{nom}, V^2_{nom}, and V^3_{nom} are the voltage magnitudes on sides 1, 2, and 3 of the transformer. The nominal impedance magnitude of the transformer is defined as:

$$Z_{\text{nom}} = \frac{(V^m_{\text{nom}})^2}{S_{\text{nom}}}, \tag{39}$$

where V^m_{nom} is the chosen nominal voltage magnitudes at star middle point of the transformer. V^m_{nom} can be set to V^1_{nom}, V^2_{nom}, or V^3_{nom} depending on usage.

Bus

The per-unit transformation for a bus is defined in Table 23, where V_{nom} is voltage magnitude at the voltage level of the bus. If the network is in *flat* representation, then the voltage component of the bus is transformed in per-unit according to the description provided in the last three rows of Table 23.

GRG name	Parameter	Per-unit transformation
tap_changer_1→impedance→resistance	R_{1_k}	$\frac{1}{Z_{nom}} \cdot R_{1_k}$
tap_changer_1→impedance→reactance	X_{1_k}	$\frac{1}{Z_{nom}} \cdot X_{1_k}$
tap_changer_1→shunt→conductance	G_{1_k}	$Z_{nom} \cdot G_{1_k}$
tap_changer_1→shunt→susceptance	B_{1_k}	$Z_{nom} \cdot B_{1_k}$
tap_changer_1→transform→tap_ratio	$(\frac{V_m^{\text{nom}}}{V_1^{\text{nom}}})r_{1_k}$	r_{1_k}
tap_changer_1→transform→angle_shift	δ_{1_k}	$\frac{\pi}{180} \cdot \delta_{1_k}$
tap_changer_1→steps[k]→impedance→resistance	R_{1_k}	$\frac{1}{Z_{nom}} \cdot R_{1_k}$
tap_changer_1→steps[k]→impedance→reactance	X_{1_k}	$\frac{1}{Z_{nom}} \cdot X_{1_k}$
tap_changer_1→steps[k]→shunt→conductance	G_{1_k}	$Z_{nom} \cdot G_{1_k}$
tap_changer_1→steps[k]→shunt→susceptance	B_{1_k}	$Z_{nom} \cdot B_{1_k}$
tap_changer_1→steps[k]→transform→tap_ratio	$(\frac{V_m^{\text{nom}}}{V_1^{\text{nom}}})r_{1_k}$	r_{1_k}
tap_changer_1→steps[k]→transform→angle_shift	δ_{1_k}	$\frac{\pi}{180} \cdot \delta_{1_k}$
tap_changer_2→impedance→resistance	R_{2_k}	$\frac{1}{Z_{nom}} \cdot R_{2_k}$
tap_changer_2→impedance→reactance	X_{2_k}	$\frac{1}{Z_{nom}} \cdot X_{2_k}$
tap_changer_2→transform→tap_ratio	$(\frac{V_m^{\text{nom}}}{V_2^{\text{nom}}})r_{2_k}$	r_{2_k}
tap_changer_2→transform→angle_shift	δ_{2_k}	$\frac{\pi}{180} \cdot \delta_{2_k}$
tap_changer_2→steps[k]→impedance→resistance	R_{2_k}	$\frac{1}{Z_{nom}} \cdot R_{2_k}$
tap_changer_2→steps[k]→impedance→reactance	X_{2_k}	$\frac{1}{Z_{nom}} \cdot X_{2_k}$
tap_changer_2→steps[k]→transform→tap_ratio	$(\frac{V_m^{\text{nom}}}{V_2^{\text{nom}}})r_{2_k}$	r_{2_k}
tap_changer_2→steps[k]→transform→angle_shift	δ_{2_k}	$\frac{\pi}{180} \cdot \delta_{2_k}$
tap_changer_3→impedance→resistance	R_{3_k}	$\frac{1}{Z_{nom}} \cdot R_{3_k}$
tap_changer_3→impedance→reactance	X_{3_k}	$\frac{1}{Z_{nom}} \cdot X_{3_k}$
tap_changer_3→transform→tap_ratio	$(\frac{V_m^{\text{nom}}}{V_3^{\text{nom}}})r_{3_k}$	r_{3_k}
tap_changer_3→transform→angle_shift	δ_{3_k}	$\frac{\pi}{180} \cdot \delta_{3_k}$
tap_changer_3→steps[k]→impedance→resistance	R_{3_k}	$\frac{1}{Z_{nom}} \cdot R_{3_k}$
tap_changer_3→steps[k]→impedance→reactance	X_{3_k}	$\frac{1}{Z_{nom}} \cdot X_{3_k}$
tap_changer_3→steps[k]→transform→tap_ratio	$(\frac{V_m^{\text{nom}}}{V_3^{\text{nom}}})r_{3_k}$	r_{3_k}
tap_changer_3→steps[k]→transform→angle_shift	δ_{3_k}	$\frac{\pi}{180} \cdot \delta_{3_k}$
current_limits_1[k]→min	$I_{1_k}^{min}$	$\frac{1}{I_{nom}} \cdot I_{1_k}^{min}$
current_limits_1[k]→max	$I_{1_k}^{max}$	$\frac{1}{I_{nom}} \cdot I_{1_k}^{max}$
current_limits_2[k]→min	$I_{2_k}^{min}$	$\frac{1}{I_{nom}} \cdot I_{2_k}^{min}$
current_limits_2[k]→max	$I_{2_k}^{max}$	$\frac{1}{I_{nom}} \cdot I_{2_k}^{max}$
current_limits_3[k]→min	$I_{3_k}^{min}$	$\frac{1}{I_{nom}} \cdot I_{3_k}^{min}$
current_limits_3[k]→max	$I_{3_k}^{max}$	$\frac{1}{I_{nom}} \cdot I_{3_k}^{max}$
thermal_limits_1[k]→min	$S_{1_k}^{min}$	$\frac{1}{S_{nom}} \cdot S_{1_k}^{min}$
thermal_limits_1[k]→max	$S_{1_k}^{max}$	$\frac{1}{S_{nom}} \cdot S_{1_k}^{max}$
thermal_limits_2[k]→min	$S_{2_k}^{min}$	$\frac{1}{S_{nom}} \cdot S_{2_k}^{min}$
thermal_limits_2[k]→max	$S_{2_k}^{max}$	$\frac{1}{S_{nom}} \cdot S_{2_k}^{max}$
thermal_limits_3[k]→min	$S_{3_k}^{min}$	$\frac{1}{S_{nom}} \cdot S_{3_k}^{min}$
thermal_limits_3[k]→max	$S_{3_k}^{max}$	$\frac{1}{S_{nom}} \cdot S_{3_k}^{max}$

Table 22: Three winding transformer per-unit transformations.

GRG name	Parameter	Per-unit transformation
voltage→magnitude	v	$\frac{1}{V_{\text{nom}}} \cdot v$
voltage→angle→lb	θ^l	$\frac{\pi}{180} \cdot \theta^l$
voltage→angle→ub	θ^u	$\frac{\pi}{180} \cdot \theta^u$
voltage→nominal_value	V_{nom}	1
voltage→lower_limit	V^L	$\frac{1}{V_{nom}} \cdot V^L$
voltage→upper_limit	V^U	$\frac{1}{V_{nom}} \cdot V^U$

Table 23: Voltage component per-unit transformations.

Shunt

For a shunt (see Figure 13), the per-unit transformation is defined in Table 24. The nominal impedance of the AC line is defined as:

$$Z_{nom} = \frac{(V_{nom})^2}{S_{nom}} \tag{40}$$

GRG name	Parameter	Per-unit transformation
shunt→conductance	G	$\frac{1}{Z_{nom}} \cdot G$
shunt→susceptance	B	$\frac{1}{Z_{nom}} \cdot B$

Table 24: Shunt per-unit transformations.

Load

For a load, the per-unit transformation is defined in Table 25.

GRG name	Parameter	Per-unit transformation
demand→active	P	$\frac{1}{S_{nom}} \cdot P$
demand→reactive	Q	$\frac{1}{S_{nom}} \cdot Q$

Table 25: Load per-unit transformations.

Generator

For a generator, the per-unit transformation is defined in Table 26.

GRG name	Parameter	Per-unit transformation
output→active	P	$\frac{1}{S_{nom}} \cdot P$
output→reactive	Q	$\frac{1}{S_{nom}} \cdot Q$
PQ_curve→values[k][0]	P_k	$\frac{1}{S_{nom}} \cdot P_k$
PQ_curve→values[k][1]	Q_k^{min}	$\frac{1}{S_{nom}} \cdot Q_k^{min}$
PQ_curve→values[k][2]	Q_k^{max}	$\frac{1}{S_{nom}} \cdot Q_k^{max}$

Table 26: Generator per-unit transformations.

Synchronous Condenser

For a synchronous condenser, the per-unit transformation is defined in Table 27.

GRG name	Parameter	Per-unit transformation
output→reactive	Q	$\frac{1}{S_{nom}} \cdot Q$

Table 27: Synchronous condenser per-unit transformations.

Voltage Level

Each voltage component of a voltage level is transformed according to Table 28.

GRG name	Parameter	Per-unit transformation
voltage→nominal_value	V_{nom}	1
voltage→lower_limit	V^L	$\frac{1}{V_{nom}} \cdot V^L$
voltage→upper_limit	V^U	$\frac{1}{V_{nom}} \cdot V^U$

Table 28: Voltage Level per-unit transformations.