DINTMB02
JSON file configuration guide

1. Overview

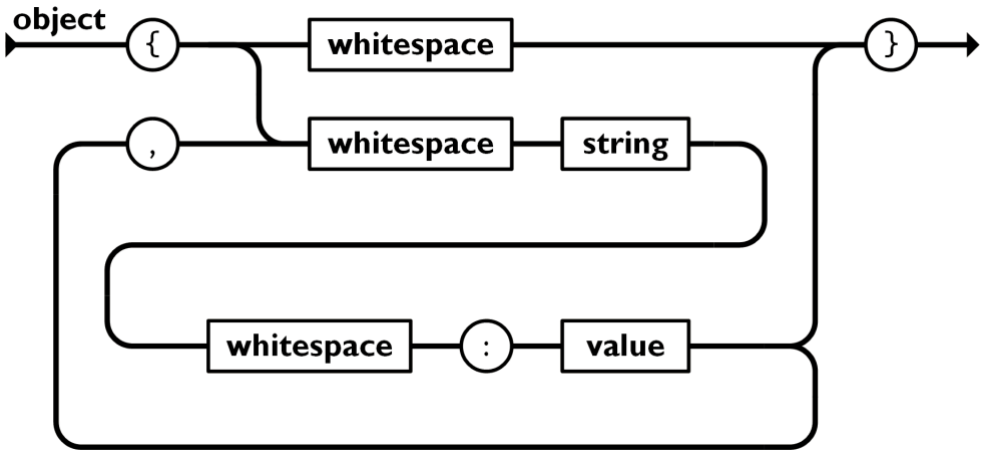
JSON file is an easy way to read/write format to exchange configuration or data between systems based on different architecture/language. It can also be used to import/export configurations to easily modify them without the need of a specific software.

2. JSON formatting

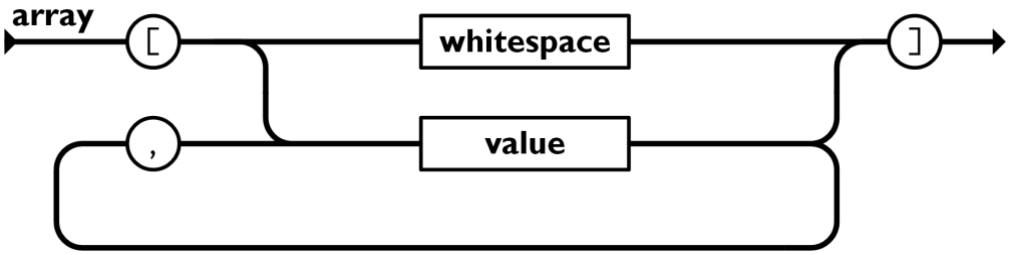
(source: https://www.json.org/json-en.html)

JSON file has a strict formatting specification and must be strictly respected or the file will be unreadable by the system/software.

- An *object* is an unordered set of name/value pairs. An object begins with left brace ('{') and ends with right brace ('}'). Each name is followed by colon (':') and the name/value pairs are separated by comma (',').

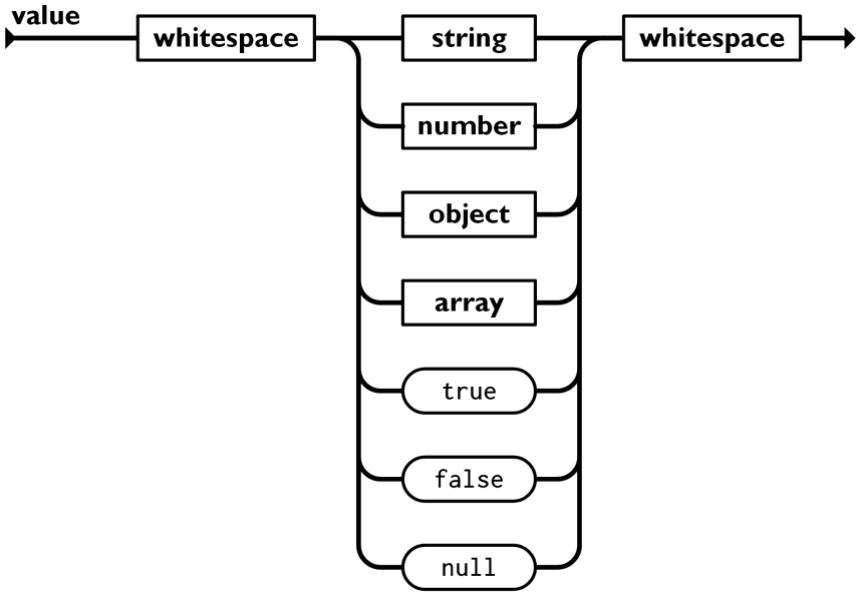


- An array is an ordered collection of values. An array begins with left bracket ('[') and ends with right bracket (']'). Values are separated by comma (',').

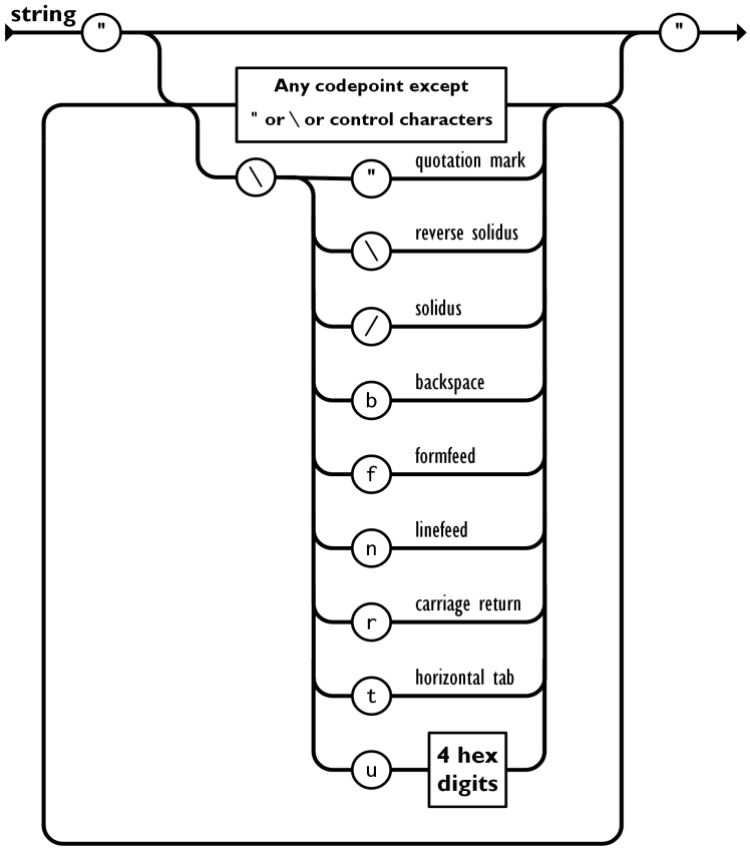


DINTMB02
JSON file configuration guide

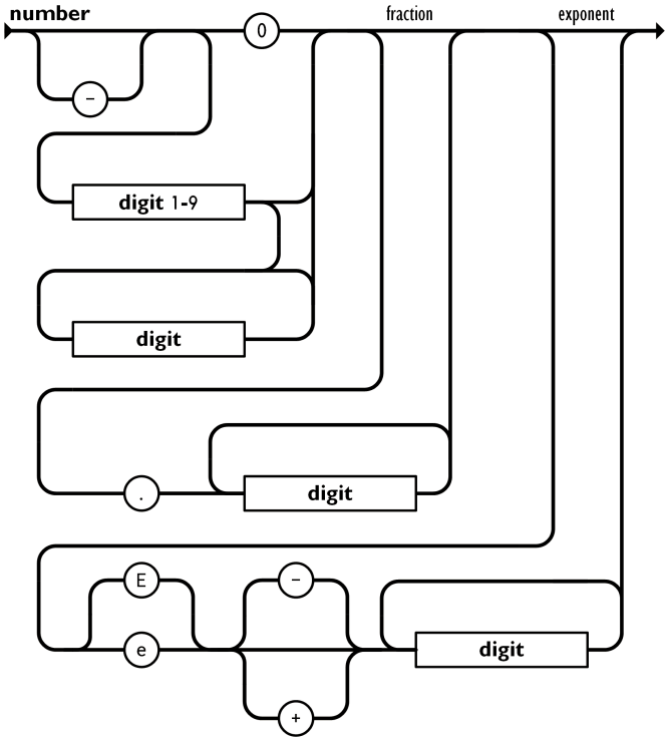
- A *value* can be a *string* in double quotes, or a *number*, or true or false or null, or an *object* or an *array*. These structures can be nested.



- A *string* is a sequence of zero or more Unicode characters, wrapped in double quotes (""), using backslash escapes ('\'). A character is represented as a single character string. A string is like a C or Java string.



- A *number* is very much like a C or Java number, except that the octal and hexadecimal formats are not used.



- Here is an example of JSON snippet:

Example

```
"config": {
    "type": "serial",
    "param": {
        "baudrate": 9600,
     "databits": 8,
     "stopbits": 1,
     "parity":"none"
  }
}
```

Domintell registers

- di_temperature

- Description: Holds the measured temperature in Celsius degrees in decimal floating-point format (e.g. 3.4).
- Associated constant:
  - None
- Associated values/arguments/members:
  - 1st : measure temperature
- Lua assignment example:
  - return 12.3

- di_setpoint
  - Description: Holds the temperature set-point in Celsius degrees in decimal floating-point format (e.g. 3.4).
  - Associated constant:
    - None
  - Associated values/arguments/members:
    - 1st: the current set-point
  - Lua assignment example:
    - return 21.0
- di_mode
  - Description: Holds the regulation mode in integer format (use provided constants) (e.g. DI_DRY).
  - Associated constant:
    - DI_OFF = 0
    - DI_HEAT = 1
    - DI_COOL = 2
    - DI_AUTO = 5
    - DI_DRY = 6
    - DI_FAN = 7
    - DI_ERROR = -1
  - Associated values/arguments/members:
    - 1st: the current regulation mode
  - Lua assignment example:
    - return DI_HEAT
- di_error
  - Description: Holds error information. Format: <error level>, <description>, <error code> (e.g. DI_LVLCRITICAL, "pump defect", 65).
  - Associated constant:
    - DI_LVLNORMAL = 0: Normal/Information level
    - DI_LVLWARNING = 1: Warning level
    - DI_LVLCRITICAL = 3: Error/Critical level
    - DI_ERROR = -1
  - Associated values/arguments/members:

        - 1st: measure temperature
    - Lua assignment example:
        - return DI_LVLCRITICAL, "pump defect", 65
- di_fanspeed
    - Description: Holds the speed of the fan in integer format (provided constants or custom values can be used) (e.g. DI_AUTO or 3).
    - Associated constant:
        - DI_AUTO = 254
    - Associated values/arguments/members:
        - 1st: the current speed of the fan.
    - Lua assignment example:
        - return 1
        - return DI_AUTO
- di_vanespos
    - Description: Holds the speed of the fan in integer format (provided constants or custom values can be used) (e.g. DI_AUTOPOS or 2).
    - Associated constant:
        - DI_AUTOPOS = 252: Auto position that let device automatically choose the position.
        - DI_FROZENPOS = 253: It allows to stop vanes at a specific position while they are swinging.
        - DI_SWING = 254: Swing vanes.
    - Associated values/arguments/members:
        - 1st: the current position of vanes.
    - Lua assignment example:
        - return 2
        - return DI_SWING
- di_elecfreq
    - Description: Holds the frequency of the electrical network in Hz unit. In decimal floating-point format (e.g. 50.1 = 50.1 Hz).
    - Associated constant:
        - None
    - Associated values/arguments/members:
        - 1st: the measured frequency of the grid.
    - Lua assignment example:
        - return 50.1

- di_elecpfactor1, di_elecpfactor2, di_elecpfactor3
    - Description: Holds the power factor of the phase 1, 2 and 3. In decimal floating-point format (e.g. 0.95).
    - Associated constant:
        - None

  - Associated values/arguments/members:
    - 1st: The measured power factor.
  - Lua assignment example:
    - return 0.95
- di_elecvolt1, di_elecvolt2, di_elecvolt3
  - Description: Holds the voltage of the phase 1, 2 and 3 in V unit. In decimal floating-point format (e.g. 239.3 = 239.3 V).
  - Associated constant:
    - None
  - Associated values/arguments/members:
    - 1st: The measured voltage of the line.
  - Lua assignment example:
    - return 239.3
- di_eleccurrent1, di_eleccurrent2, di_eleccurrent3
  - Description: Holds the current of the phase 1, 2 and 3 in A unit. In decimal floating-point format (e.g. 12.4 = 12.4 A).
  - Associated constant:
    - None
  - Associated values/arguments/members:
    - 1st: The measured current of the line.
  - Lua assignment example:
    - return 12.4
- di_elecpower1, di_elecpower2, di_elecpower3
  - Description: Holds the instantaneous power of the phase 1, 2 and 3 in W unit. In integer format (e.g. 12387 = 12.387 kW).
  - Associated constant:
    - None
  - Associated values/arguments/members:
    - 1st: measure temperature
  - Lua assignment example:
    - return 12.3
- di_elecconsumedpower
  - Description: Holds the instantaneous consumed power in W unit. In unsigned integer format (e.g. 12387 = 12.387 kW). Value must be positive.
  - Associated constant:
    - None
  - Associated values/arguments/members:
    - 1st: measure temperature
  - Lua assignment example:
    - return 12.3
- di_elecproducedpower

  - Description: Holds the instantaneous produced power in W unit. In unsigned integer format (e.g. 12387 = 12.387 kW). Value must be positive.
  - Associated constant:
    - None
  - Associated values/arguments/members:
    - 1st: measure temperature
  - Lua assignment example:
    - return 12.3
- di_electotalpower
  - Description: Holds the instantaneous total power in W unit. In signed integer format (e.g. 12387 = 12.387 kW). If value is negative, more power has been produced than consumed.
  - Associated constant:
    - None
  - Associated values/arguments/members:
    - 1st: measure temperature
  - Lua assignment example:
    - return 12.3
- di_elecenergy1, di_elecenergy2, di_elecenergy3
  - Description: Holds the energy of phase 1, 2 and 3 in kWh unit. In signed integer format (e.g. 1238 = 1.238 MWh). If value is negative, more power has been produced than consumed.
  - Associated constant:
    - None
  - Associated values/arguments/members:
    - 1st: measure temperature
  - Lua assignment example:
    - return 12.3
- di_elecfwdenergy
  - Description: Holds the forward energy in kWh unit. In unsigned integer format (e.g. 1238 = 1.238 MWh). Value must be positive.
  - Associated constant:
    - None
  - Associated values/arguments/members:
    - 1st: measure temperature
  - Lua assignment example:
    - return 12.3
- di_elecrevenergy
  - Description: Holds the reversed energy in kWh unit. In unsigned integer format (e.g. 1238 = 1.238 MWh). Value must be positive.
  - Associated constant:
    - None
  - Associated values/arguments/members:

        - 1st: measure temperature
    - Lua assignment example:
        - return 12.3
- di_electotenergy
    - Description: Holds the total energy in kWh unit. In signed integer format (e.g. 1238 = 1.238 MWh). If value is negative, more power has been produced than consumed.
    - Associated constant:
        - None
    - Associated values/arguments/members:
        - 1st: measure temperature
    - Lua assignment example:
        - return 12.3
- di_electotenergyt1, di_electotenergyt2, di_electotenergyt3, di_electotenergyt4
    - Description: Holds the energy in kWh unit consumed in tariffs 1, 2, 3 and 4 period. In signed integer format (e.g. 1238 = 1.238 MWh). If value is negative, more power has been produced than consumed.
    - Associated constant:
        - None
    - Associated values/arguments/members:
        - 1st: measure temperature
    - Lua assignment example:
        - return 12.3
- di_electindicator
    - Description: Holds the current active tariff period. In unsigned integer format (e.g. 2 = period 2 tariff).
    - Associated constant:
        - None
    - Associated values/arguments/members:
        - 1st: measure temperature
    - Lua assignment example:
        - return 12.3
- di_relaystate
    - Description: Holds the current state of the relay. Use provided constants (e.g. DI_ON = relay on).
    - Associated constant:
        - None
    - Associated values/arguments/members:
        - 1st: measure temperature
    - Lua assignment example:
        - return 12.3
- di_version

  - Description: Holds the current version of the device. Format <raw string>, <dot-decimal number> (e.g. "2.1 2024/01/01", 2.1).
  - Associated constant:
    - None
  - Associated values/arguments/members:
    - 1st: measure temperature
  - Lua assignment example:
    - return 12.3
- di_percent
  - Description: Holds the percentage of an analog input. Integer from 0 to 100 (e.g. 70 = 70%).
  - Associated constant:
    - None
  - Associated values/arguments/members:
    - 1st: measure temperature
  - Lua assignment example:
    - return 12.3
- di_analog
  - Description: Holds the value of an analog input. Format <decimal floating-point>, [<unit string>] (e.g. 7.39, "pH" = 7.39 pH).
  - Associated constant:
    - None
  - Associated values/arguments/members:
    - 1st: measure temperature
  - Lua assignment example:
    - return 12.3
- di_serial
  - Description: Holds the serial number of the device. In string format (e.g. "162VX5687T").
  - Associated constant:
    - None
  - Associated values/arguments/members:
    - 1st: measure temperature
  - Lua assignment example:
    - return ''

JSON fields

Fields followed by a red asterisk ('*') are mandatory

DINTMB02
JSON file configuration guide

Global

- name *: Name of the module, string. This name will be assigned to the module in GoldenGate and can also be used to build the name of IOs of the Module.

  Example

  ```
  "name": "Daikin RTD-RA"
  ```

- type *: Type of ModBus device, enum string. Possible values are:
  - Airco: Will create an air-conditioner module containing a temperature sensor, a fan and a vanes IO.
  - Generic: Will create a generic module

  Example

  ```
  "type": "Airco"
  ```

- address *: ModBus address of the device, integer (0-255).

  Example

  ```
  "address": 80
  ```

- description: Description of the ModBus device, string. Ignored when imported to GoldenGate.

  Example

  ```
  "description": "Daikin ModBus interface to interact with Daikin VRV airconditioner"
  ```

Config

- config *: Configuration of the communication, list of JSON objects.
  - config.type *: Type of protocol, enum string. Possible values are:
    - serial: Serial ModBus device using ModBus RTU protocol.
    - tcp: Ethernet ModBus device using ModBus TCP protocol.

    Example

    ```
    "type": "serial"
    ```

  - config.param *: Parameters of the communication, list of JSON objects.
    - For config.type = "serial":
      - config.param.baudrate * : Baudrate, enum integer. Possible values are:
        - 1200: 1200 bauds.

        - 2400: 2400 bauds.
        - 4800: 4800 bauds.
        - 9600: 9600 bauds.
        - 38400: 38400 bauds.
        - 57600: 57600 bauds.
        - 115200: 115200 bauds.
    - config.param.databits *: Number of bits per words, enum interger. Possible values are:
        - 8: 8 bits per word.
    - config.param.stopbits * : Number of stop bits, enum interger. Possible values are:
        - 1: 1 stopbit
        - 3: 1.5 stopbit
        - 2: 2 stopbits
    - config.param.parity *: Type of parity, enum string. Possible values are:
        - none: No parity.
        - even: Even parity.
        - odd: Odd parity.

Example

```
"type": "serial"
"parameters": {
   "baudrate": 9600,
   "databits": 8,
   "stopbits": 1,
   "parity": "none"
}&
```

- For config.type = "tcp":
    - config.param.host *: IP address of the device, IPv4 string.
    - config.param.port *: TCP port, interger (1-65535, usually 502).

Example

```
"type": "tcp"
"parameters": {
   "host": "102.168.1.200",
   "port": 502
}
```

Full example for config

```
"config": {
   "type": "serial",
   "parameters": {
      "baudrate": 9600,
      "databits": 8,
      "stopbits": 1,
```

```
    "parity": "none"
  }
```

- }

Type of IO

- <type of the IO> key *: Declare a list of IOs of the specified type, enum string. Possible values are:
  - temp: Declare a temperature sensor. Mandatory if type = "Airco".
  - fan: Declare a fan IO. Mandatory if type = "Airco".
  - vanes: Declare a vanes IO. Mandatory if type = "Airco".
  - status: Declare a status IO to report issues from the ModBus device.
  - elec: Declare an energy meter IO.
  - relay: Declare a relay output.
  - percentIn: Declare a percentage (0-100%) input.
  - analogIn: Declare an analog input.

- <type of the IO> value *: Array of JSON objects. Each object is an IO. The example contains two temperature sensors and one fan IO.

Example

```
"temp": [
  {
    "param": {
      ...
    },
    "lua": {
      ...
    }
  },
  {
    "param": {
      ...
    },
    "lua": {
      ...
    }
  }
],
"fan": [
  {
    "param": {
      ...
    },
    "lua": {
      ...
    }
  }
[
```

DINTMB02
JSON file configuration guide

IO parameters

- <type of the IO>.param * : Parameters of the IO, list of JSON objects. The key of each objects is an enum string. If a type of IO does not have any parameters, an empty "param" node must be present anyway. Possible values depend on the type of the IO:
  - if <type of the IO> key = "temp" :
    - hasAccurateMeasTemp : Tells if the measured temperature is accurate. Some air-conditioners only have a return air flow temperature which is not exactly the same as the temperature of the room, boolean (true or false). Default: true.

      Example
      "hasAccurateMeasTemp": false

    - minSetpointStep : Minimal incremental step for the set-point, floating-point number. For example, if the value is set to 5.0, set-point can only be incremented by 5.0°C or any of its multiple. Default: 0.5 °C.

      Example
      "minSetpointStep": 0.5

    - minCoolTemp : Minimal temperature in cool mode that the ModBus device can handle, Celcius degrees as floating-point number. Default: 18.0 °C.

      Example
      "minCoolTemp": 18.0

    - maxCoolTemp : Maximal temperature in cool mode that the ModBus device can handle, Celcius degrees as floating-point number. Default: 32.0 °C

      Example
      "maxCoolTemp": 32.0

    - minHeatTemp : Minimal temperature in heat mode that the ModBus device can handle, Celcius degrees as floating-point number. Default: 10.0 °C.

      Example
      "minHeatTemp": 10.0

    - maxHeatTemp : Maximal temperature in heat mode that the ModBus device can handle, Celcius degrees as floating-point number. Default: 30.0 °C.

      Example
      "maxHeatTemp": 30.0

    - hasAutoMode : Tells if the ModBus device can handle the auto regulation mode, boolean (true or false). Default: true.

Example
"hasAutoMode": true

- hasHeatMode : Tells if the ModBus device can handle the heat regulation mode, boolean (true or false). Default: true.

Example
"hasHeatMode": true

- hasCoolMode : Tells if the ModBus device can handle the cool regulation mode, boolean (true or false). Default: true.

Example
"hasCoolMode": true

- hasFanMode : Tells if the ModBus device can handle the fan regulation mode, boolean (true or false). Default: true.

Example
"hasFanMode": true

- hasDryMode : Tells if the ModBus device can handle the dry regulation mode, boolean (true or false). Default: true.

Example
"hasDryMode": true

◦ if <type of the IO> key = "fan" :

- speedMax : Tells how much speed has the ModBus device, integer (0-200). Excluding the off and the auto state. Default: 0.

Example
"speedMax": 5

- hasOffState : Tells if the fan of the ModBus device can be turn off, boolean (true or false). Default: true.

Example
"hasOffState": true

- hasAutoState : Tells if the fan of the ModBus device has an auto state (speed automatically managed by the ModBus device, boolean (true or false). Default: true.

Example
"hasOffState": true

◦ if <type of the IO> key = "vanes" :

- posMax : Tells how much speed has the ModBus device, integer (0-200). Excluding the off, the auto and the swing state. Default: 0.

  Example
  "posMax": 5

- hasOffState : Tells if vanes have a off/closed state, boolean (true or false). Default: false.

  Example
  "hasOffState": true

- hasAutoState : Tells if vanes have an auto state, boolean (true or false). Default: false.

  Example
  "hasAutoState": false

- hasSwingState : Tells if vanes have a swing state, boolean (true or false). Default: true.

  Example
  "hasSwingState": true

- if <type of the IO> key = "elec" :
  - None
- if <type of the IO> key = "evCharger" :
  - None
- if <type of the IO> key = "battery" :
  - None
- if <type of the IO> key = "relay" :
  - None
- if <type of the IO> key = "percentIn" :
  - None
- if <type of the IO> key = "analogIn" :
  - None
- if <type of the IO> key = "status" :
  - None

DINTMB02
JSON file configuration guide

Example when the IO has parameters

```
"temp": {
    "param": {
        "hasAccurateMeasTemp": false,
        "minSetpointStep": 1.0,
        "minCoolTemp": 18.0,
        "maxCoolTemp": 32.0,
        "minHeatTemp": 10.0,
        "maxHeatTemp": 30.0,
        "hasAutoMode": true,
        "hasHeatMode": true,
        "hasCoolMode": true,
        "hasFanMode": true,
        "hasDryMode": true
    },
    "lua": {
        ...
    }
}
```

Example when IO has no parameters

```
"elec" {
    "param": {
    },
    "lua": {
        ...
    }
}
```

DINTMB02
JSON file configuration guide

Category of Lua scripts

- <type of the IO>.lua * : All data and scripts to convert values from/to ModBus using Lua scripts, list of Domintell register as JSON objects. Available Domintell registers (<diRegLua>) depends on <type of the IO> value :

  - if <type of the IO> = "temp", <diRegLua> can be :

    - tempMeas * : Scripts that handle the measured temperature. It is associated to di_temperature register in the Lua editor in GoldenGate. Only mandatory if "temp.param.hasAccurateMeasTemp" = true.

    - tempSetpoint * : Scripts that handle the measured temperature. It is associated to di_setpoint register in the Lua editor in GoldenGate.

    - tempMode * : Scripts that handle the measured temperature. It is associated to di_mode register in the Lua editor in GoldenGate.

Example

```
"temp": [
   {
      "param": {
         ...
      },
      "lua": {
         "tempMeas": {
            ...
         },
         "tempSetPoint": {
            ...
         },
         "tempMode": {
            ...
         }
      }
   }
]
```

  - if <type of the IO> = "fan", <diRegLua> can be :

    - fanSpeed * : Scripts that handle the measured temperature. It is associated to di_fanspeed register in the Lua editor in GoldenGate.

Example

```
"fan": [
   {
      "param": {
         ...
      },
      "lua": {
         "fanSpeed": {
            ...
         }
      }
   }
]
```

- if <type of the IO> = "vanes", <diRegLua> can be :
  - vanesPos * : Scripts that handle the measured temperature. It is associated to di_vanespos register in the Lua editor in GoldenGate.

Full example for a vanes IO

```
"vanes": [
   {
      "param": {
         "posMax": 5,
         "hasOffState": true,
         "hasAutoState": true,
         "hasSwingState": false
      },
      "lua": {
         "vanesPos": {
            "modbus2diOnly": 0,
            "di2modbusVars": [
               "local DI_ERROR = -1",
               "local DI_AUTOPOS = 252",
               "local DI_FROZENPOS = 253",
               "local DI_SWING = 254",
               "local di_vanespos = ..."
            ],
            "di2modbus": [
               "if (di_vanespos == DI_AUTOPOS) then",
               "    return 5",
               "elseif (di_vanespos == DI_FROZENPOS) then",
               "    return 6",
               "else",
               "    return di_vanespos",
               "end"
            ],
            "di2modbusRetInfo": [
               "Return value to be written to mb_vanes ModBus register",
               "eg: return 3"
            ],
            "modbus2diRegs": [
               {
```

```
                    "name": "mb_vanes",
                    "type": "HoldingReg",
                    "address": 21,
                    "description": "Swing/Vane position. 0: Verical; 1: 30°; 2: 45°; 3: 60°; 4: Horizontal; 5: Auto
(swinging); 6: Off"
                }
            ],
            "modbus2diVars": [
                "local DI_ERROR = -1",
                "local DI_AUTOPOS = 252",
                "local DI_FROZENPOS = 253",
                "local DI_SWING = 254",
                "local mb_vanes = ..."
            ],
            "modbus2di": [
                "if (mb_vanes == 5) then",
                "    return DI_AUTOPOS",
                "elseif (mb_vanes == 6) then",
                "    return DI_FROZENPOS",
                "else",
                "    return mb_vanes",
                "end"
            ],
            "modbus2diRetInfo": [
                "Return value to be used in Domintell",
                "eg: return DI_SWING",
                "eg: return 1"
            ]
        }
    }
  }
],
```

- if <type of the IO> = "status", <diRegLua> can be :
  - error : Scripts that handle the error reported by the ModBus device. It is associated to di_error register in the Lua editor in GoldenGate.
  - version : Scripts that handle the version returned by the ModBus device. It is associated to di_version register in the Lua editor in GoldenGate.
  - serial : Scripts that handle the error reported by the ModBus device. It is associated to di_serial register in the Lua editor in GoldenGate.

- if <type of the IO> = "elec", <diRegLua> can be :
  - elecFrequency : Scripts that handle the measured voltage on L1. It is associated to di_elecfreq in the Lua editor in GoldenGate.
  - elecVoltL1 : Scripts that handle the measured voltage on L1. It is associated to di_elecvolt1 in the Lua editor in GoldenGate.
  - elecVoltL2 : Scripts that handle the measured voltage on L2. It is associated to di_elecvolt2 in the Lua editor in GoldenGate.
  - elecVoltL3 : Scripts that handle the measured voltage on L3. It is associated to di_elecvolt3

in the Lua editor in GoldenGate.

- elecPFL1 : Scripts that handle the measured power factor on L1. It is associated to di_elecpfactor1 in the Lua editor in GoldenGate.
- elecPFL2 : Scripts that handle the measured power factor on L2. It is associated to di_elecpfactor2 in the Lua editor in GoldenGate.
- elecPFL3 : Scripts that handle the measured power factor on L3. It is associated to di_elecpfactor3 in the Lua editor in GoldenGate.
- elecCurrentL1 : Scripts that handle the measured current on L1. It is associated to di_eleccurrent1 in the Lua editor in GoldenGate.
- elecCurrentL2 : Scripts that handle the measured current on L2. It is associated to di_eleccurrent2 in the Lua editor in GoldenGate.
- elecCurrentL3 : Scripts that handle the measured current on L3. It is associated to di_eleccurrent3 in the Lua editor in GoldenGate.
- elecPowerL1 : Scripts that handle the measured active power on L1. It is associated to di_elecpower1 in the Lua editor in GoldenGate.
- elecPowerL2 : Scripts that handle the measured active power on L2. It is associated to di_elecpower2 in the Lua editor in GoldenGate.
- elecPowerL3 : Scripts that handle the measured active power on L3. It is associated to di_elecpower3 in the Lua editor in GoldenGate.
- elecConsumedPower : Scripts that handle the global consumed power. It is associated to di_elecconsumedpower in the Lua editor in GoldenGate.
- elecProducedPower : Scripts that handle the global produced power. It is associated to di_elecproducedpower (must be positive) in the Lua editor in GoldenGate.
- elecTotalPower : Scripts that handle the total measured active power (should be equivalent to elecconsumedpower - elecproducedpower). It must be negative if more power has been produced than consumed. It is associated to di_electotalpower in the Lua editor in GoldenGate.
- elecEnergyL1 : Scripts that handle the accumulated active energy on L1. It is associated to di_elecenergy1 in the Lua editor in GoldenGate.
- elecEnergyL2 : Scripts that handle the accumulated active energy on L2. It is associated to di_elecenergy2 in the Lua editor in GoldenGate.
- elecEnergyL3 : Scripts that handle the accumulated active energy on L3. It is associated to di_elecenergy3 in the Lua editor in GoldenGate.
- elecForwardEnergy : Scripts that handle the global accumulated forward energy. It is associated to di_elecfwdenergy in the Lua editor in GoldenGate.
- elecReverseEnergy : Scripts that handle the global accumulated reverse energy. It is associated to di_elecrevenergy (must be positive) in the Lua editor in GoldenGate.
- elecTotalEnergy : Scripts that handle the total global energy (should be equivalent to elecForwardEnergy - elecReverseEnergy). It must be negative if more energy has been injected to the grid than consumed from the grid. It is associated to di_electotenergy in the Lua editor in GoldenGate.
- elecTotalEnergyTariff1 : Scripts that handle the total global energy while tariff 1 period. It must be negative if more energy has been injected to the grid than consumed from the grid during period 1. It is associated to di_electotenergyt1 in the Lua editor in GoldenGate.

- elecTotalEnergyTariff2 : Scripts that handle the total global energy while tariff 2 period. It must be negative if more energy has been injected to the grid than consumed from the grid during period 2. It is associated to di_electotenergyt2 in the Lua editor in GoldenGate.
- elecTotalEnergyTariff3 : Scripts that handle the total global energy while tariff 3 period. It must be negative if more energy has been injected to the grid than consumed from the grid during period 3. It is associated to di_electotenergyt3 in the Lua editor in GoldenGate.
- elecTotalEnergyTariff4 : Scripts that handle the total global energy while tariff 4 period. It must be negative if more energy has been injected to the grid than consumed from the grid during period 4. It is associated to di_electotenergyt4 in the Lua editor in GoldenGate.
- elecTariffIndic : Scripts that handle the current active tariff period It is associated to di_electindicator in the Lua editor in GoldenGate.

◦ if <type of the IO> = "relay", <diRegLua> can be :
- relayState * : Scripts that handle the state of the relay output. It is associated to di_relaystate register in the Lua editor in GoldenGate.

◦ if <type of the IO> = "analogIn", <diRegLua> can be :
- analog * : Scripts that handle the value of a raw analog input. It is associated to di_analog register in the Lua editor in GoldenGate.

◦ if <type of the IO> = "percentIn", <diRegLua> can be :
- percent * : Scripts that handle the value of a percent (0-100%) analog input. It is associated to di_percent register in the Lua editor in GoldenGate.

◦ if <type of the IO> = "evCharger", <diRegLua> can be :
- *Coming soon...*

◦ if <type of the IO> = "battery", <diRegLua> can be :
- *Coming soon...*

Parameters of Lua scripts

- modbusRegs * : Array of JSON objects containing the configuration of ModBus registers. A maximum of five 16-bit registers can be assigned per when using a DINTMB02.
  - modbusRegs.name * : Name of the register in the Lua script, string. **Must start with "mb_" with only lowercases, at least one letter after "mb_" and without special characters or spaces**.
  - modbusRegs.type * : Type of the ModBus register, enum string. Possible values are :
    - HoldingReg : A holding (16 bits) register. ModBus functions : 0x03, 0x06 and 0x10.
    - InputReg : An input (16 bits) register. ModBus functions : 0x04.
    - CoilReg : A coil (1 bit) register. ModBus functions : 0x01, 0x05 and 0x0F.
    - DiscreteReg : A discrete (1 bit) register. ModBus functions : 0x02.
    - ExtensionReg : Virtual register used to declared extra space/register required to store more than 16 bits values. To be used when modbusRegs.nbrItems bigger than 1.

DINTMB02
JSON file configuration guide

- modbusRegs.address * : Address of the ModBus register (0-based), integer (0 - 65535).

- modbusRegs.format : Format/Mapping of the ModBus data, enum string. Possible values are :
  - 1bit : Width of the data is 1 bit.
  - S16 : Width of the data is 16 bits. Default.
  - S32-4321 : Width of the data is 32 bits. If data is 0x44332211, 0x4433 will be stored in 16-bit register with the lowest address and 0x2211 will be stored in the 16-bit register with the highest address. This is the most used 32-bit format.
  - S32-2143 : Signed 32-bit integer. If data is 0x44332211, 0x2211 will be stored in 16-bit register with the lowest address and 0x4433 will be stored in the 16-bit register with the highest address.
  - F32-IEEE : 32-bit floating point number. Stored using IEEE-754 standard (https://en.wikipedia.org/wiki/IEEE_754).
  - raw : raw/string data

- modbusRegs.nbrItems : Number of ModBus 16-bit register required to hold the data. If modbusRegs.type is ExtensionReg, it is the offset to reorder them when joining them together.

- modbusRegs.access : Type of the access to the ModBus register, enum string. Possible values are :
  - rw : ModBus register can be read ans write. Default.
  - ro : ModBus register can only be read.
  - wo : ModBus register can only be written.

- modbusRegs.description : Description of the ModBus register that explains the format of the data based on the datasheet of the ModBus device.

Example of ModBus registers for temp.lua.tempMode

```
"modbusRegs": [
   {
      "name": "mb_op_mode",
      "type": "HoldingReg",
      "address": 3,
      "description": "Mode (0: auto; 1: heating; 2: fan; 3: cooling: 4: dry)"
   },
   {
      "name": "mb_mode_on_off",
      "type": "HoldingReg",
      "address": 5,
      "description": "On/Off the device (0: off; 1: on)"
   }
]
```

Example of ModBus registers for elec.lua.elecCurrentL1

```
"modbus2diRegs": [
   {
```

```
        "name": "mb_curl1",
        "type": "HoldingReg",
        "address": 313,
        "format": "S32-4321",
        "nbrItems": 2,
        "access": "ro",
        "description": "The current of L1 in 0.001 A unit."
    },
    {
        "name": "mb_curl1_ext1",
        "type": "ExtensionReg",
        "address": 314,
        "format": "S32-4321",
        "nbrItems": 1,
        "access": "ro",
        "description": "The current of L1 in 0.001 A unit."
    }
]
```

- di2modbusConst : Array of strings containing declarations of contants that are used in the scripts. This section is useful to notify user that he must review the script when he imports the script to a newer version of GoldenGate while these constants have been added/updated. This section must contain the list of constants related to the register (see chapter) and the name of the related Domintell register (see chapter)

Example for the fan.lua.fanSpeed script

```
"di2modbusConst": [
    "local DI_OFF = 0",
    "local DI_AUTO = 254",
    "local DI_ERROR = -1",
    "local di_fanspeed = ..."
]
```

- di2modbusRetInfo : String/Help about values that are returned by the script. These information must be based on the datasheet of the ModBus device.

Example

```
"di2modbusRetInfo": [
    "Return value to be written to mb_fanspeed ModBus register",
    "eg: return 2"
]
```

- di2modbus : Array of strings of the Lua script that will convert the value from Domintell to a value suited to fit in the 16-bit ModBus register(s). Each line of the script is a item/slot/index of the array.

Example

```
"di2modbus": [
    "if (di_fanspeed == DI_AUTO) then",
    "    return 0",
    "else",
    "    return di_fanspeed",
    "end"
```

```
]
```

- modbus2diConst : Array of strings containing declarations of contants that are used in the scripts. This section is useful to notify user that he must review the script when he imports the script to a newer version of GoldenGate while these constants have been added/updated. This section must contain the list of constants related to the register (see chapter) and the name of ModBus registers defined modbus2diRegs.

- modbus2diRetInfo : String/Help about values that are returned by the script. These information are used to notify user when he imports the JSON file to GoldenGate and returned arguments have been modified (see chapter).

- modbus2di : Array of strings of the Lua script that will convert the value from ModBus register(s) to a value suited to fit in the Domintell register. Each line of the script is a item/slot/index of the array.

Example

```
"modbus2di": [
    "if (mb_fanspeed == 0) then",
    "    return DI_AUTO",
    "else",
    "    return mb_fanspeed",
    "end"
]
```

Registers for simulation

- simulRegs : List of ModBus registers to be used by a ModBus slave simulator, array of JSON objects. **Not used by GoldenGate**.
  - simulRegs[].name * : Name of the ModBus register to simulate, string

    Example
    ```
    "name": "H0002"
    ```

  - simulRegs[].type * : Type of the ModBus register to simulate, enum string. Possible values are:
    - HoldingReg : Declare a holding register.
    - InputReg : Declare an input register.
    - CoilReg : Declare a coil register.
    - DiscreteReg : Declare a discrete input register.

    Example
    ```
    "type" : "HoldingReg"
    ```

  - simulRegs[].address * : Address of the ModBus register to simulate, integer (1-65535). 1-based address.

    Example

```
"address": 2
```

- simulRegs[].description * : Description of the ModBus register to simulate, string.

Example
```
"description": "Setpoint: 10°C to 30°C in heating mode"
```

- simulRegs[].range : Range of valid values that the ModBus register to simulate can accept, array of integers.

Example
```
"range": [ 10, 32 ]
```

- simulRegs[].value : Initial/Default value of the ModBus register to simulate, integer.

Example
```
"value": 7
```

Full example for simulRegs
```
"simulRegs": [
    {
        "name": "H0002",
        "type": "HoldingReg",
        "address": 2,
        "description": "Setpoint: 10°C to 30°C in heating mode; 18°C to 32°C in cooling mode (Degrees C unit)",
        "range": [ 10, 32 ],
        "value": 21
    },
    {
        "name": "H0003",
        "type": "HoldingReg",
        "address": 3,
        "description": "FanSpeed (0: auto; 1-5: speed 1 to 5)",
        "range": [ 0, 5 ],
        "value": 0
    }
]
```

Full example for the Coolmaster air-conditioner
```
{
    "name": "CoolMasterNet SI mode",
    "type": "Airco",
    "address": 80,
    "description": "CoolMasterNet bridge with L1 in SI master mode and L3 in CG5 slave mode.",
    "config": {
        "type": "serial",
        "param": {
            "baudrate": 9600,
            "databits": 8,
            "stopbits": 1,
            "parity": "none"
        }
    },
```

DINTMB02
JSON file configuration guide


```
  "simulRegs": [
    {
      "name": "Op. Mode",
      "type": "HoldingReg",
      "address": 17,
      "description": "Operation mode (0: Cool; 1: heat; 2: Auto; 3: Dry; 4: HAUX; 5: Fan; 6: HH; 8: VAM Auto; 9: VAM Bypass; 10: VAM Heat Exc; 11: VAM normal)",
      "range": [ 0, 11 ],
      "value": 2
    },
    {
      "name": "Fan Speed",
      "type": "HoldingReg",
      "address": 18,
      "description": "FanSpeed (0-2: Lo-Me-Hi; 3: auto; 4: top; 5; very-low; 7: VAM Super Hi; 8; VAM low fresh-up; 9: VAM high fresh-up); VAM value does not work in SI mode",
      "range": [ 0, 9 ],
      "value": 3
    },
    {
      "name": "Setpoint",
      "type": "HoldingReg",
      "address": 19,
      "description": "Setpoint expressed in 10th degrees Celcius",
      "range": [ 0, 400 ],
      "value": 210
    },
    {
      "name": "On/off",
      "type": "HoldingReg",
      "address": 20,
      "description": "On/Off the device (0: off; 1: on)",
      "range": [ 0, 1 ],
      "value": 1
    },
    {
      "name": "Filter sign",
      "type": "HoldingReg",
      "address": 21,
      "description": "This register maybe warns when the filter must be clean clean (read) and ii is also used to clean this state (turn off warning light) (write)",
      "range": [ 0, 1 ],
      "value": 1
    },
    {
      "name": "Vanes",
      "type": "HoldingReg",
      "address": 22,
      "description": "Swing/Vane position. 0: Verical; 1: 30°; 2: 45°; 3: 60°; 4: Horizontal; 5: Auto (swinging); 6: Off",
      "range": [ 0, 6 ],
      "value": 5
    },
    {
      "name": "Room temp",
```

DINTMB02
JSON file configuration guide


```
            "type": "HoldingReg",
            "address": 23,
            "description": "Temperature of the room in 10th degrees Celcius. Some HVAC models allow to set a suggested
temperature room.",
            "range": [ 0, 400 ],
            "value": 210
        },
        {
            "name": "HVAC error code",
            "type": "HoldingReg",
            "address": 24,
            "description": "HVAC error code.",
            "range": [ 0, 65535 ],
            "value": 0
        },
        {
            "name": "Room temp",
            "type": "InputReg",
            "address": 18,
            "description": "Temperature of the room in 10th degrees Celcius.",
            "range": [ 0, 400 ],
            "value": 210
        },
        {
            "name": "HVAC error start of string",
            "type": "InputReg",
            "address": 19,
            "description": "First part of HVAC error string (stored in little endian mode)."
        },
        {
            "name": "HVAC error end of string",
            "type": "InputReg",
            "address": 20,
            "description": "Second part of HVAC error string (stored in little endian mode)."
        }
    ],
    "temp": [
        {
            "param": {
                "hasAccurateMeasTemp": false,
                "minSetpointStep": 1.0,
                "minCoolTemp": 18.0,
                "maxCoolTemp": 32.0,
                "minHeatTemp": 10.0,
                "maxHeatTemp": 30.0,
                "hasAutoMode": true,
                "hasHeatMode": true,
                "hasCoolMode": true,
                "hasFanMode": true,
                "hasDryMode": true
            },
            "lua": {
                "tempMeas": {
                    "modbusRegs": [
```

DINTMB02
JSON file configuration guide

```
            {
                "name": "mb_roomtemp",
                "type": "InputReg",
                "address": 17,
                "description": "Temperature of the room in 10th degrees Celcius."
            }
        ],
        "modbus2diConst": [
            "local mb_roomtemp = ..."
        ],
        "modbus2di": [
            "return mb_roomtemp / 10"
        ],
        "modbus2diRetInfo": [
            "Return temperature in floating decimal format to be used in Domintell",
            "eg: return 19.6",
            "eg: return mb_roomtemp / 10"
        ]
    },
    "tempSetpoint": {
        "di2modbusConst": [
            "local di_setpoint = ..."
        ],
        "di2modbus": [
            "return (di_setpoint * 10)"
        ],
        "di2modbusRetInfo": [
            "Return values to be written to <mb_setpoint> ModBus registers",
            "eg: return 200"
        ],
        "modbusRegs": [
            {
                "name": "mb_setpoint",
                "type": "HoldingReg",
                "address": 18,
                "description": "Setpoint expressed in 10th degrees Celcius"
            }
        ],
        "modbus2diConst": [
            "local mb_setpoint = ..."
        ],
        "modbus2di": [
            "return mb_setpoint / 10"
        ],
        "modbus2diRetInfo": [
            "Return temperature in floating decimal format to be used in Domintell",
            "eg: return 20.0",
            "eg: return mb_setpoint / 10"
        ]
    },
    "tempMode": {
        "di2modbusConst": [
            "local DI_OFF = 0",
            "local DI_HEAT = 1",
```

DINTMB02
JSON file configuration guide


```
            "local DI_COOL = 2",
            "local DI_AUTO = 5",
            "local DI_DRY = 6",
            "local DI_FAN = 7",
            "local DI_ERROR = -1",
            "local di_mode = ..."
        ],
        "di2modbus": [
            "if (di_mode == DI_OFF) then",
            "    return 2,0",
            "elseif (di_mode == DI_HEAT) then",
            "    return 1,1",
            "elseif (di_mode == DI_COOL) then",
            "    return 0,1",
            "elseif (di_mode == DI_AUTO) then",
            "    return 2,1",
            "elseif (di_mode == DI_DRY) then",
            "    return 3,1",
            "elseif (di_mode == DI_FAN) then",
            "    return 5,1",
            "else",
            "    return DI_ERROR, DI_ERROR",
            "end"
        ],
        "di2modbusRetInfo": [
            "Return values to be written to <mb_op_mode> and <mb_mode_on_off> ModBus registers",
            "eg: return 2,0",
            "Attention : value must be returned in the same order than the declaration of the ModBus registers !"
        ],
        "modbusRegs": [
            {
                "name": "mb_op_mode",
                "type": "HoldingReg",
                "address": 16,
                "description": "Operation mode (0: Cool; 1: heat; 2: Auto; 3: Dry; 4: HAUX; 5: Fan; 6: HH; 8: VAM
Auto; 9: VAM Bypass; 10: VAM Heat Exc; 11: VAM normal)"
            },
            {
                "name": "mb_mode_on_off",
                "type": "HoldingReg",
                "address": 19,
                "description": "On/Off the device (0: off; 1: on)"
            }
        ],
        "modbus2diConst": [
            "local DI_OFF = 0",
            "local DI_HEAT = 1",
            "local DI_COOL = 2",
            "local DI_AUTO = 5",
            "local DI_DRY = 6",
            "local DI_FAN = 7",
            "local DI_ERROR = -1",
            "local mb_op_mode, mb_mode_on_off = ..."
        ],
```

DINTMB02
JSON file configuration guide


```
                "modbus2di": [
                    "if (mb_mode_on_off == 0) then",
                    "    return DI_OFF",
                    "elseif (mb_op_mode == 0) then",
                    "    return DI_COOL",
                    "elseif (mb_op_mode == 1) then",
                    "    return DI_HEAT",
                    "elseif (mb_op_mode == 2) then",
                    "    return DI_AUTO",
                    "elseif (mb_op_mode == 3) then",
                    "    return DI_DRY",
                    "elseif (mb_op_mode == 5) then",
                    "    return DI_FAN",
                    "else",
                    "    return DI_ERROR",
                    "end"
                ],
                "modbus2diRetInfo": [
                    "Return value to be used in Domintell",
                    "eg: return DI_HEAT",
                    "DI_xxx variables must be used ! Do not use numeric values !"
                ]
            }
        }
    }
],
"fan": [
    {
        "param": {
            "speedMax": 5,
            "hasOffState": false,
            "hasAutoState": true
        },
        "lua": {
            "fanSpeed": {
                "di2modbusConst": [
                    "local DI_OFF = 0",
                    "local DI_AUTO = 254",
                    "local DI_ERROR = -1",
                    "local di_fanspeed = ..."
                ],
                "di2modbus": [
                    "if (di_fanspeed == DI_AUTO) then",
                    "    return 3",
                    "elseif (di_fanspeed >= 6) then",
                    "    return di_fanspeed + 1",
                    "elseif (di_fanspeed > 3) then",
                    "    return di_fanspeed",
                    "else",
                    "    return di_fanspeed - 1",
                    "end"
                ],
                "di2modbusRetInfo": [
                    "Return value to be written to mb_fan ModBus register",
```

DINTMB02
JSON file configuration guide


```
                    "eg: return 2"
                ],
                "modbusRegs": [
                    {
                        "name": "mb_fan",
                        "type": "HoldingReg",
                        "address": 17,
                        "description": "FanSpeed (0-2: Lo-Me-Hi; 3: auto; 4: top; 5; very-low; 7: VAM Super Hi; 8; VAM low fresh-up; 9: VAM high fresh-up); VAM value does not work in SI mode"
                    }
                ],
                "modbus2diConst": [
                    "local DI_ERROR = -1",
                    "local DI_OFF = 0",
                    "local DI_AUTO = 254",
                    "local mb_fan = ..."
                ],
                "modbus2di": [
                    "if (mb_fan == 3) then",
                    "    return DI_AUTO",
                    "elseif (mb_fan < 3) then",
                    "    return mb_fan + 1",
                    "elseif (mb_fan >= 7) then",
                    "    return mb_fan - 1",
                    "else",
                    "    return mb_fan",
                    "end"
                ],
                "modbus2diRetInfo": [
                    "Return value to be used in Domintell",
                    "eg: return DI_OFF",
                    "eg: return 2"
                ]
            }
        }
    }
  ],
  "vanes": [
    {
        "param": {
            "posMax": 5,
            "hasOffState": true,
            "hasAutoState": true,
            "hasSwingState": false
        },
        "lua": {
            "vanesPos": {
                "di2modbusConst": [
                    "local DI_ERROR = -1",
                    "local DI_AUTOPOS = 252",
                    "local DI_FROZENPOS = 253",
                    "local DI_SWING = 254",
                    "local di_vanespos = ..."
                ],
```

DINTMB02
JSON file configuration guide


```
                "di2modbus": [
                    "if (di_vanespos == DI_AUTOPOS) then",
                    "    return 5",
                    "elseif (di_vanespos == DI_FROZENPOS) then",
                    "    return 6",
                    "else",
                    "    return di_vanespos",
                    "end"
                ],
                "di2modbusRetInfo": [
                    "Return value to be written to mb_vanes ModBus register",
                    "eg: return 3"
                ],
                "modbusRegs": [
                    {
                        "name": "mb_vanes",
                        "type": "HoldingReg",
                        "address": 21,
                        "description": "Swing/Vane position. 0: Verical; 1: 30°; 2: 45°; 3: 60°; 4: Horizontal; 5: Auto
(swinging); 6: Off"
                    }
                ],
                "modbus2diConst": [
                    "local DI_ERROR = -1",
                    "local DI_AUTOPOS = 252",
                    "local DI_FROZENPOS = 253",
                    "local DI_SWING = 254",
                    "local mb_vanes = ..."
                ],
                "modbus2di": [
                    "if (mb_vanes == 5) then",
                    "    return DI_AUTOPOS",
                    "elseif (mb_vanes == 6) then",
                    "    return DI_FROZENPOS",
                    "else",
                    "    return mb_vanes",
                    "end"
                ],
                "modbus2diRetInfo": [
                    "Return value to be used in Domintell",
                    "eg: return DI_SWING",
                    "eg: return 1"
                ]
            }
        }
    }
  ],
  "status": [
    {
        "param": {
        },
        "lua": {
            "error": {
                "modbusRegs": [
```

DINTMB02
JSON file configuration guide

```
                {
                    "name": "mb_error",
                    "type": "HoldingReg",
                    "address": 23,
                    "description": "HVAC error code."
                }
            ],
            "modbus2diConst": [
                "local DI_ERROR = -1",
                "local DI_LVLNORMAL = 0",
                "local DI_LVLWARNING = 1",
                "local DI_LVLCRITICAL = 2",
                "local mb_error = ..."
            ],
            "modbus2di": [
                "if (mb_error == 0) then",
                "    return DI_LVLNORMAL, mb_error, \"Pas d'erreur\"",
                "elseif (mb_error == 666) then",
                "    return DI_LVLCRITICAL, mb_error, \"Devil is inside\"",
                "else",
                "    return DI_LVLWARNING, mb_error, \"Unkonwn error\"",
                "end"
            ],
            "modbus2diRetInfo": [
                "Return values to be used in Domintell",
                "return <error_level>, <error_value>, <error_infostr>",
                "eg: return DI_LVLCRITICAL, 45, \"Water pump is faulty\"",
                "eg: return DI_LVLNORMAL, 0, \"\""
            ]
        }
    }
  }
]
}
```

DINTMB02
JSON file configuration guide

Alphabetical Index

DINTMB02
JSON file configuration guide

DINTMB02
JSON file configuration guide

DINTMB02
JSON file configuration guide

DINTMB02
Guide de configuration du fichier JSON

1. Présentation

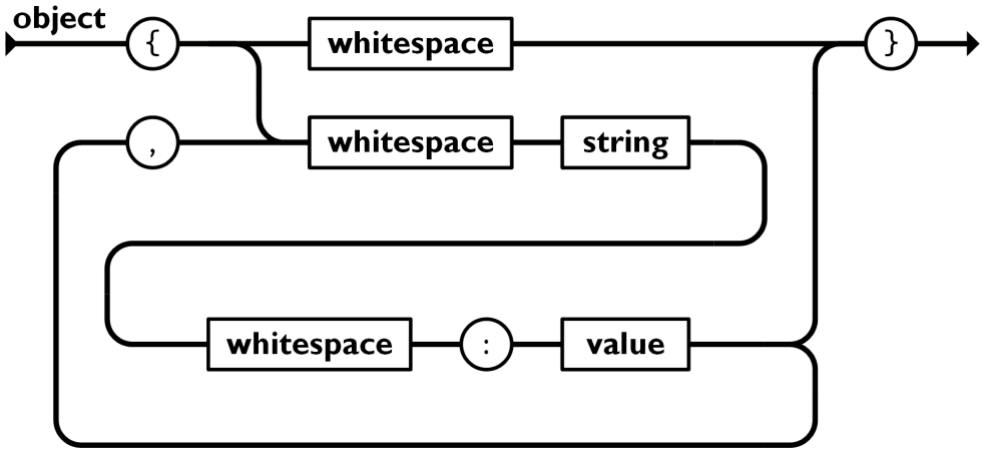
Le fichier JSON est un format facile à lire/écrire pour échanger des configurations ou des données entre des systèmes basés sur des architectures/langages différents. Il peut également être utilisé pour importer/exporter des configurations afin de les modifier facilement sans avoir besoin d'un logiciel spécifique.

2. JSON formatting

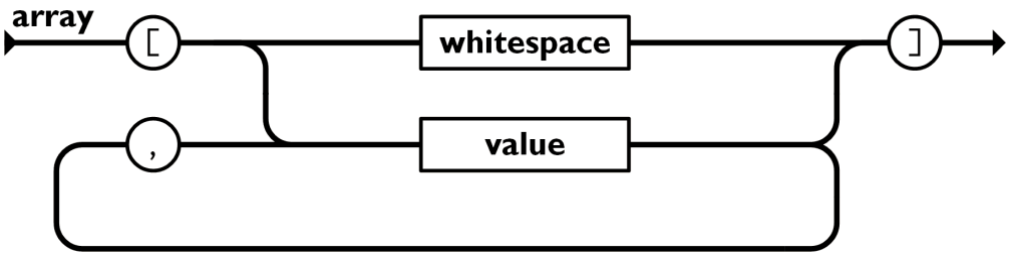
(source: https://www.json.org/json-fr.html)

Le fichier JSON a une spécification de formatage stricte et doit être strictement respecté ou le fichier sera illisible par le système/logiciel.

- Un objet est un ensemble non ordonné de paires nom/valeur. Un objet commence par une accolade gauche ("{") et se termine par une accolade droite ("}"). Chaque nom est suivi de deux points (':') et les paires nom/valeur sont séparées par une virgule (',').

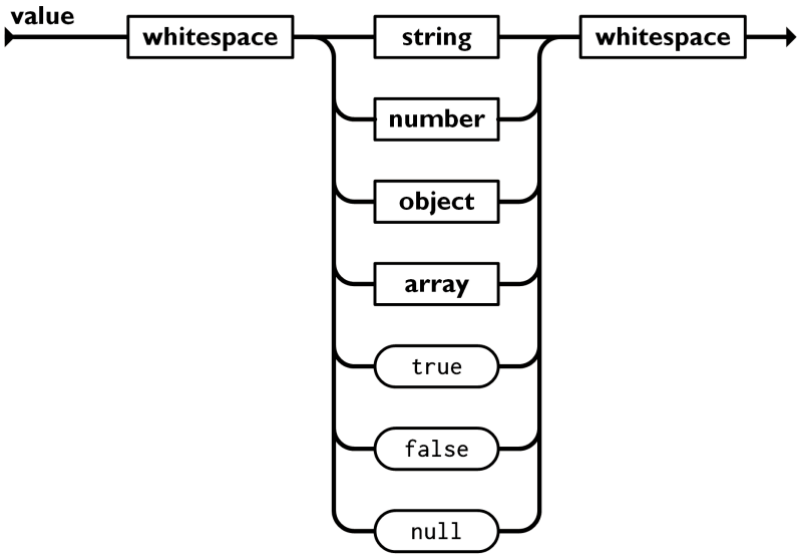


- Un tableau est une collection ordonnée de valeurs. Un tableau commence par un crochet gauche ('[') et se termine par un crochet droit (']'). Les valeurs sont séparées par une virgule (',').

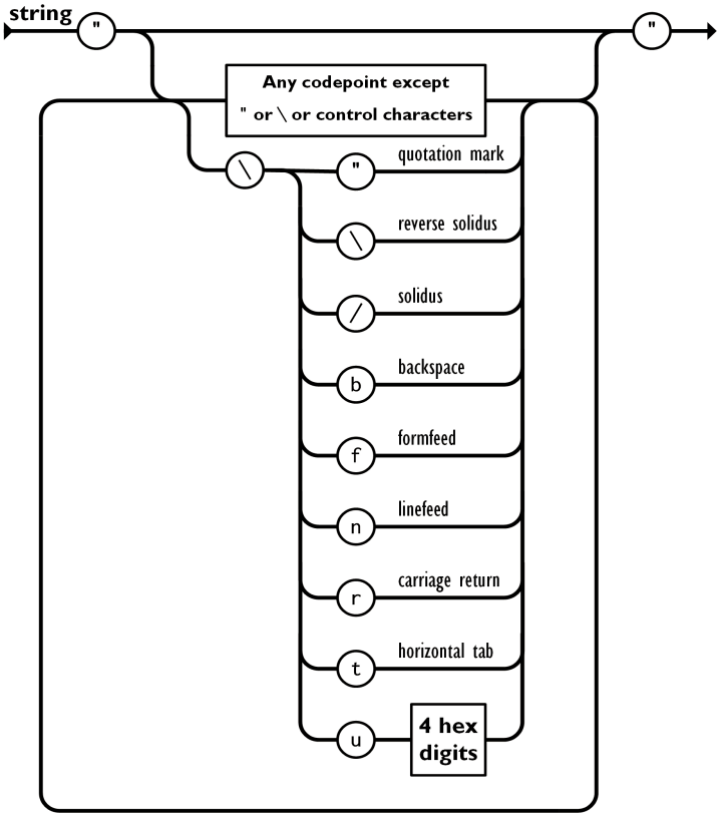


DINTMB02
Guide de configuration du fichier JSON

- Une *valeur* peut être une *chaîne* de caractères entre guillemets, ou un *nombre*, ou true ou false ou null, ou un *objet* ou un tableau. Ces structures peuvent être imbriquées.

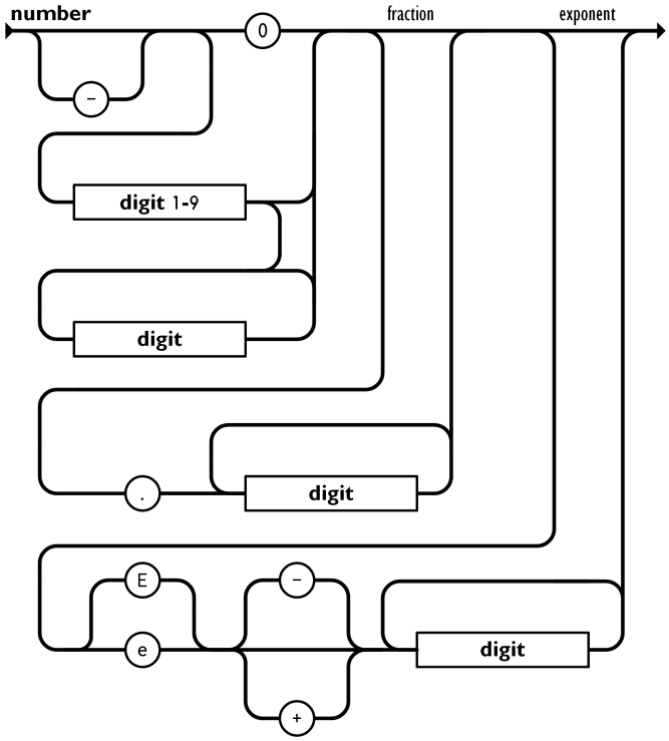


- Une *chaîne* de caractères est une séquence de zéro ou plusieurs caractères Unicode, enveloppés dans des guillemets doubles (""), en utilisant des barres obliques inversées ('\'). Un caractère est représenté sous la forme d'une chaîne de caractères unique. Une chaîne de caractères est semblable à une chaîne de caractères C ou Java.



DINTMB02
Guide de configuration du fichier JSON

- Un *nombre* ressemble beaucoup à un nombre C ou Java, sauf que les formats octal et hexadécimal ne sont pas utilisés.



- Here is an example of JSON snippet:

Exemple

```
"config": {
    "type": "serial",
    "param": {
        "baudrate": 9600,
     "databits": 8,
     "stopbits": 1,
     "parity":"none"
  }
}
```

Registres Domintell

- di_temperature

- Description : Indique la température mesurée en degrés Celsius au format décimal à virgule flottante (par exemple 3,4).
- Constante associée :
  - Aucune
- Valeurs/arguments/membres associés :
  - 1st : mesurer température
- Exemple d'affectation en Lua :
  - return 12.3

- di_setpoint
  - Description: Indique le point de consigne de la température en degrés Celsius au format décimal à virgule flottante (par exemple 3,4).
  - Constante associée:
    - Aucune
  - Valeurs/arguments/membres associés :
    - 1st: le point de consigne actuel
  - Exemple d'affectation en Lua :
    - return 21.0
- di_mode
  - Description: Indique le mode de régulation en format entier (utiliser les constantes fournies) (par exemple DI_DRY).
  - Constante associée:
    - DI_OFF = 0
    - DI_HEAT = 1
    - DI_COOL = 2
    - DI_AUTO = 5
    - DI_DRY = 6
    - DI_FAN = 7
    - DI_ERROR = -1
  - Valeurs/arguments/membres associés :
    - 1st: le mode de régulation actuel
  - Exemple d'affectation en Lua :
    - return DI_HEAT
- di_error
  - Description: Contient des informations sur les erreurs. Format : <niveau d'erreur>, <description>, <code d'erreur> (par exemple DI_LVLCRITICAL, "défaut de la pompe", 65).
  - Constante associée:
    - DI_LVLNORMAL = 0: Niveau normal/information
    - DI_LVLWARNING = 1: Niveau avertissement
    - DI_LVLCRITICAL = 3: Erreur/Niveau critique
    - DI_ERROR = -1

- Valeurs/arguments/membres associés:
  - 1st: mesure température
- Exemple d'affectation en Lua:
  - return DI_LVLCRITICAL, "défaut de la pompe", 65

- di_fanspeed
  - Description: Indique la vitesse du ventilateur au format entier (des constantes fournies ou des valeurs personnalisées peuvent être utilisées) (par exemple DI_AUTO ou 3).Constante associée:
    - DI_AUTO = 254
  - Valeurs/arguments/membres associés:
    - 1st: la Vitesse actuelle du ventilateur
  - Exemple d'affectation en Lua:
    - return 1
    - return DI_AUTO
- di_vanespos
  - Description: Indique la vitesse du ventilateur au format entier (des constantes fournies ou des valeurs personnalisées peuvent être utilisées) (par exemple DI_AUTOPOS ou 2).
  - Constante associée :
    - DI_AUTOPOS = 252: La position automatique permet à l'appareil de choisir automatiquement la position.
    - DI_FROZENPOS = 253: Il permet d'arrêter les aubes à une position spécifique pendant qu'elles se balancent.
    - DI_SWING = 254: Palettes oscillantes.
  - Valeurs/arguments/membres associés:
    - 1st: la position actuelle des ailettes
  - Exemple d'affectation en Lua:
    - return 2
    - return DI_SWING
- di_elecfreq
  - Description: Indique la fréquence du réseau électrique en unité Hz. En format décimal à virgule flottante (par exemple 50.1 = 50.1 Hz).
  - Constante associée:
    - Aucune
  - Valeurs/arguments/membres associés:
    - 1st: la fréquence mesurée de la grille.
  - Exemple d'affectation en Lua:
    - return 50.1

- di_elecpfactor1, di_elecpfactor2, di_elecpfactor3
  - Description: Indique le facteur de puissance des phases 1, 2 et 3. En format décimal à virgule flottante (par exemple 0,95).

- Constante associée:
  - Aucune
- Valeurs/arguments/membres associés:
  - 1st: Le facteur de puissance mesuré.
- Exemple d'affectation en Lua:
  - return 0.95

- di_elecvolt1, di_elecvolt2, di_elecvolt3
  - Description: Indique la tension de la phase 1, 2 et 3 en unité V. En format décimal à virgule flottante (par exemple 239,3 = 239,3 V).
  - Constante associée:
    - Aucune
  - Valeurs/arguments/membres associés:
    - 1st: La tension mesurée de la ligne.
  - Exemple d'affectation en Lua:
    - return 239.3
- di_eleccurrent1, di_eleccurrent2, di_eleccurrent3
  - Description: Indique le courant de la phase 1, 2 et 3 en unité A. En format décimal à virgule flottante (par exemple 12,4 = 12,4 A).
  - Constante associée:
    - Aucune
  - Valeurs/arguments/membres associés:
    - 1st: Le courant mesuré de la ligne.
  - Exemple d'affectation en Lua:
    - return 12.4
- di_elecpower1, di_elecpower2, di_elecpower3
  - Description: Indique la puissance instantanée de la phase 1, 2 et 3 en unité W. Au format entier (par exemple 12387 = 12,387 kW).
  - Constante associée:
    - Aucune
  - Valeurs/arguments/membres associés:
    - 1st: mesurer la température
  - Exemple d'affectation en Lua:
    - return 12.3
- di_elecconsumedpower
  - Description: Indique la puissance consommée instantanée en unité W. Au format entier non signé (par exemple, 12387 = 12,387 kW). La valeur doit être positive.
  - Constante associée:
    - Aucune
  - Valeurs/arguments/membres associés:
    - 1st: mesurer la température
  - Exemple d'affectation en Lua:

        - return 12.3
- di_elecproducedpower
    - Description: Indique la puissance instantanée produite en W. Au format entier non signé (par exemple, 12387 = 12,387 kW). La valeur doit être positive.
    - Constante associée:
        - Aucune
    - Valeurs/arguments/membres associés:
        - 1st: mesurer la température
    - Exemple d'affectation en Lua:
        - return 12.3
- di_electotalpower
    - Description: Indique la puissance totale instantanée en W. Au format d'un nombre entier signé (par exemple, 12387 = 12,387 kW). Si la valeur est négative, la puissance produite est supérieure à la puissance consommée.
    - Constante associée:
        - Aucune
    - Valeurs/arguments/membres associés:
        - 1st: mesurer la température
    - Exemple d'affectation en Lua:
        - return 12.3
- di_elecenergy1, di_elecenergy2, di_elecenergy3
    - Description: Indique l'énergie des phases 1, 2 et 3 en kWh. Au format d'un nombre entier signé (par exemple, 1238 = 1,238 MWh). Si la valeur est négative, la puissance produite est supérieure à la puissance consommée.
    - Constante associée:
        - Aucune
    - Valeurs/arguments/membres associés:
        - 1st: mesurer la température
    - Exemple d'affectation en Lua:
        - return 12.3
- di_elecfwdenergy
    - Description: Indique l'énergie à terme en kWh. Au format entier non signé (par exemple, 1238 = 1,238 MWh). La valeur doit être positive.
    - Constante associée:
        - Aucune
    - Valeurs/arguments/membres associés:
        - 1st: mesurer la température
    - Exemple d'affectation en Lua:
        - return 12.3
- di_elecrevenergy

- Description: Indique l'énergie inversée en kWh. Au format entier non signé (par exemple, 1238 = 1,238 MWh). La valeur doit être positive.
- Constante associée:
  - Aucune
- Valeurs/arguments/membres associés:
  - 1st: mesurer la température
- Exemple d'affectation en Lua:
  - return 12.3

- di_electotenergy
  - Description: Indique l'énergie totale en kWh. Au format d'un nombre entier signé (par exemple, 1238 = 1,238 MWh). Si la valeur est négative, l'énergie produite est supérieure à l'énergie consommée.
  - Constante associée:
    - Aucune
  - Valeurs/arguments/membres associés:
    - 1st: mesurer la température
  - Exemple d'affectation en Lua:
    - return 12.3
- di_electotenergyt1, di_electotenergyt2, di_electotenergyt3, di_electotenergyt4
  - Description: Indique l'énergie en kWh consommée au cours des périodes tarifaires 1, 2, 3 et 4. Au format d'un nombre entier signé (par exemple, 1238 = 1,238 MWh). Si la valeur est négative, l'énergie produite est supérieure à l'énergie consommée.
  - Constante associée:
    - Aucune
  - Valeurs/arguments/membres associés:
    - 1st: mesurer la température
  - Exemple d'affectation en Lua:
    - return 12.3
- di_electindicator
  - Description: Indique la période tarifaire active actuelle. Au format d'un nombre entier non signé (par exemple, 2 = période 2 du tarif).
  - Constante associée:
    - Aucune
  - Valeurs/arguments/membres associés:
    - 1st: mesurer la température
  - Exemple d'affectation en Lua:
    - return 12.3
- di_relaystate
  - Description: Indique l'état actuel du relais. Utilisez les constantes fournies (par exemple DI_ON = relais activé).
  - Constante associée:

        - Aucune
    - Valeurs/arguments/membres associés:
        - 1st: mesurer la température
    - Exemple d'affectation en Lua:
        - return 12.3
- di_version
    - Description: Indique la version actuelle de l'appareil. Format <chaîne brute>, <nombre décimal> (par exemple, "2.1 2024/01/01", 2.1).
    - Constante associée:
        - Aucune
    - Valeurs/arguments/membres associés:
        - 1st: mesurer la température
    - Exemple d'affectation en Lua:
        - return 12.3
- di_percent
    - Description: Indique le pourcentage d'une entrée analogique. Entier de 0 à 100 (par exemple 70 = 70%).
    - Constante associée:
        - Aucune
    - Valeurs/arguments/membres associés:
        - 1st: mesurer la température
    - Exemple d'affectation en Lua:
        - return 12.3
- di_analog
    - Description: Indique la valeur d'une entrée analogique. Format <point flottant décimal>, [<chaîne d'unités>] (par exemple 7,39, "pH" = 7,39 pH).
    - Constante associée:
        - Aucune
    - Valeurs/arguments/membres associés:
        - 1st: mesurer la température
    - Exemple d'affectation en Lua:
        - return 12.3
- di_serial
    - Description: Contient le numéro de série de l'appareil. Au format chaîne de caractères (par exemple "162VX5687T").
    - Constante associée:
        - Aucune
    - Valeurs/arguments/membres associés:
        - 1st: mesurer température
    - Exemple d'affectation en Lua:
        - return "

DINTMB02
Guide de configuration du fichier JSON

Champs JSON

Les champs suivis d'un astérisque rouge ('*') sont obligatoires.

Global

- name *: Nom du module, chaîne de caractères. Ce nom sera attribué au module dans GoldenGate et peut également être utilisé pour construire le nom des IOs du module.

  Exemple

  ```
  "name": "Daikin RTD-RA"
  ```

- type *: Type de dispositif ModBus, chaîne enum. Les valeurs possibles sont les suivantes :
  - Airco: Création d'un module de climatisation contenant un capteur de température, un ventilateur et des ailettes IO.
  - Generic: Crée un module générique

  Exemple

  ```
  "type": "Airco"
  ```

- address *: Adresse ModBus de l'appareil, nombre entier (0-255).

  Exemple

  ```
  "address": 80
  ```

- description: Description de l'appareil ModBus, chaîne de caractères. Ignoré lors de l'importation dans GoldenGate.

  Exemple

  ```
  "description": " Interface Daikin ModBus pour interagir avec le climatiseur Daikin VRV"
  ```

Configuration

- config *: Configuration de la communication, liste d'objets JSON.
  - config.type *: Type de protocole, chaîne de type enum. Les valeurs possibles sont:
    - serial: Dispositif ModBus sériel utilisant le protocole ModBus RTU.
    - tcp: Dispositif ModBus Ethernet utilisant le protocole ModBus TCP.

    Exemple

    ```
    "type": "serial"
    ```

- config.param *: Paramètres de la communication, liste d'objets JSON.
  - For config.type = "serial":
    - config.param.baudrate * : Baudrate, enum integer. Les valeurs possibles sont:
      - 1200: 1200 bauds.
      - 2400: 2400 bauds.
      - 4800: 4800 bauds.
      - 9600: 9600 bauds.
      - 38400: 38400 bauds.
      - 57600: 57600 bauds.
      - 115200: 115200 bauds.
    - config.param.databits *: Nombre de bits par mot, enum interger. Les valeurs possibles sont:
      - 8: 8 bits par mot.
    - config.param.stopbits * : Nombre de bits d'arrêt, enum interger. Les valeurs possibles sont:
      - 1: 1 stopbit
      - 3: 1.5 stopbit
      - 2: 2 stopbits
    - config.param.parity *: Type de parité, chaîne enum. Les valeurs possibles sont:
      - none: Pas de parité.
      - even: Parité paire.
      - odd: Parité impaire.

Exemple

```
"type": "serial"
"parameters": {
    "baudrate": 9600,
    "databits": 8,
    "stopbits": 1,
    "parity": "none"
}&
```

  - Pour config.type = "tcp":
    - config.param.host *: Adresse IP de l'appareil, chaîne IPv4.
    - config.param.port *: Port TCP, nombre intermédiaire (1-65535, généralement 502).

Exemple

```
"type": "tcp"
"parameters": {
    "host": "102.168.1.200",
    "port": 502
}
```

DINTMB02
Guide de configuration du fichier JSON

Full example for config

```
"config": {
   "type": "serial",
   "parameters": {
      "baudrate": 9600,
      "databits": 8,
      "stopbits": 1,
      "parity": "none"
   }
```

- }

Type d'IO

- <type of the IO> key *: Déclare une liste d'OI du type spécifié, enum string. Les valeurs possibles sont:
  - temp: Déclare un capteur de température. Obligatoire si le type = "Airco".
  - fan: Déclare un ventilateur. Obligatoire si le type = "Airco".
  - vanes: Déclare une IO ailette. Obligatoire si le type = "Airco".
  - status: Déclare une IO d'état pour signaler les problèmes de l'appareil ModBus.
  - elec: Déclarer une IO compteur d'énergie.
  - relay: Déclarer une sortie relais.
  - percentIn: Déclare un pourcentage (0-100%)
  - analogIn: Déclarer une entrée analogique.

- <type of the IO> value *: Tableau d'objets JSON. Chaque objet est une IO. L'exemple contient deux capteurs de température et un ventilateur IO.

Exemple

```
"temp": [
   {
      "param": {
         ...
      },
      "lua": {
         ...
      }
   },
   {
      "param": {
         ...
      },
      "lua": {
         ...
      }
   }
],
"fan": [
   {
```

```
      "param": {
         ...
      },
      "lua": {
         ...
      }
   }
[
```

Paramètres IO

- <type of the IO>.param * : Paramètres de l'IO, liste d'objets JSON. La clé de chaque objet est une chaîne enum. Si un type d'OI n'a pas de paramètres, un nœud "param" vide doit être présent de toute façon. Les valeurs possibles dépendent du type de l'IO:
  - Si <type of the IO> key = "temp" :
    - hasAccurateMeasTemp : Indique si la température mesurée est exacte. Certains climatiseurs n'ont qu'une température de retour d'air qui n'est pas exactement la même que la température de la pièce, booléen (vrai ou faux). Valeur par défaut : true.

      Exemple
      "hasAccurateMeasTemp": false

    - minSetpointStep : Incrément minimal du point de consigne, nombre à virgule flottante. Par exemple, si la valeur est fixée à 5,0, la consigne ne peut être incrémentée que de 5,0 °C ou de l'un de ses multiples. Valeur par défaut : 0,5 °C.

      Exemple
      "minSetpointStep": 0.5

    - minCoolTemp : Température minimale en mode froid que le dispositif ModBus peut gérer, en degrés Celcius sous forme de nombre à virgule flottante. Valeur par défaut : 18.0 °C.

      Exemple
      "minCoolTemp": 18.0

    - maxCoolTemp : Température maximale en mode froid que le dispositif ModBus peut gérer, en degrés Celcius sous forme de nombre à virgule flottante. Valeur par défaut : 32.0 °C

      Exemple
      "maxCoolTemp": 32.0

    - minHeatTemp : Température minimale en mode chauffage que le dispositif ModBus peut gérer, en degrés Celcius sous forme de nombre à virgule flottante. Valeur par défaut : 10.0 °C.

      Exemple
      "minHeatTemp": 10.0

- maxHeatTemp : Température maximale en mode chauffage que le dispositif ModBus peut gérer, en degrés Celcius sous forme de nombre à virgule flottante. Valeur par défaut : 30.0 °C.

Exemple
```
"maxHeatTemp": 30.0
```

- hasAutoMode : Indique si le dispositif ModBus peut gérer le mode de régulation automatique, booléen (vrai ou faux). Valeur par défaut : true.

Exemple
```
"hasAutoMode": true
```

- hasHeatMode : Indique si le dispositif ModBus peut gérer le mode de régulation thermique, booléen (vrai ou faux). Valeur par défaut : true.

Exemple
```
"hasHeatMode": true
```

- hasCoolMode : Indique si le dispositif ModBus peut gérer le mode de régulation froide, booléen (true ou false). Valeur par défaut : true.

Exemple
```
"hasCoolMode": true
```

- hasFanMode : Indique si le dispositif ModBus peut gérer le mode de régulation du ventilateur, booléen (vrai ou faux). Valeur par défaut : true.

Exemple
```
"hasFanMode": true
```

- hasDryMode : Indique si le dispositif ModBus peut gérer le mode de régulation à sec, booléen (vrai ou faux). Valeur par défaut : true.

Exemple
```
"hasDryMode": true
```

◦ Si <type of the IO> key = "fan" :

- speedMax : Indique la vitesse du dispositif ModBus, nombre entier (0-200). A l'exception de l'état off et de l'état auto. Valeur par défaut : 0.

Exemple
```
"speedMax": 5
```

- hasOffState : Indique si le ventilateur du dispositif ModBus peut être éteint, booléen (true ou false). Valeur par défaut : true.

DINTMB02
Guide de configuration du fichier JSON

Exemple
"hasOffState": true

- hasAutoState : Indique si le ventilateur du dispositif ModBus a un état auto (vitesse gérée automatiquement par le dispositif ModBus, booléen (true ou false). Valeur par défaut : true.

Exemple
"hasOffState": true

- Si <type of the IO> key = "vanes" :
  - posMax : Indique la vitesse du dispositif ModBus, nombre entier (0-200). A l'exception des états off, auto et swing. Valeur par défaut : 0.

Exemple
"posMax": 5

  - hasOffState : Indique si les ailettes ont un état éteint/fermé, booléen (vrai ou faux). Valeur par défaut : false.

Exemple
"hasOffState": true

  - hasAutoState : Indique si les aillettes ont un état automatique, booléen (vrai ou faux). Valeur par défaut : false.

Exemple
"hasAutoState": false

  - hasSwingState : Indique si les aillettes ont un état d'oscillation, booléen (vrai ou faux). Valeur par défaut : true.

Exemple
"hasSwingState": true

- Si <type of the IO> key = "elec" :
  - Aucun
- Si <type of the IO> key = "evCharger" :
  - Aucun
- Si <type of the IO> key = "battery" :
  - Aucun
- Si <type of the IO> key = "relay" :

- Aucun

- Si <type of the IO> key = "percentIn" :
  - Aucun

- Si <type of the IO> key = "analogIn" :
  - Aucun

- Si <type of the IO> key = "status" :
  - Aucun

Exemple when the IO has parameters

```
"temp": {
    "param": {
        "hasAccurateMeasTemp": false,
        "minSetpointStep": 1.0,
        "minCoolTemp": 18.0,
        "maxCoolTemp": 32.0,
        "minHeatTemp": 10.0,
        "maxHeatTemp": 30.0,
        "hasAutoMode": true,
        "hasHeatMode": true,
        "hasCoolMode": true,
        "hasFanMode": true,
        "hasDryMode": true
    },
    "lua": {
        ...
    }
}
```

Exemple when IO has no parameters

```
"elec" {
    "param": {
    },
    "lua": {
        ...
    }
}
```

DINTMB02
Guide de configuration du fichier JSON

Catégorie de scripts Lua

- <type of the IO>.lua * : Toutes les données et les scripts pour convertir des valeurs de/vers ModBus en utilisant des scripts Lua, la liste des registres Domintell sous forme d'objets JSON. Les registres Domintell disponibles (<diRegLua>) dépendent du <type de la valeur IO> :

  - Si <type of the IO> = "temp", <diRegLua> peut être :

    - tempMeas * : Scripts qui gèrent la température mesurée. Il est associé au registre di_temperature dans l'éditeur Lua de GoldenGate. Il n'est obligatoire que si "temp.param.hasAccurateMeasTemp" = true.

    - tempSetpoint * : Scripts qui gèrent la température mesurée. Il est associé au registre di_setpoint dans l'éditeur Lua de GoldenGate.

    - tempMode * : Scripts qui gèrent la température mesurée. Il est associé au registre di_mode dans l'éditeur Lua de GoldenGate.

Exemple

```
"temp": [
   {
      "param": {
         ...
      },
      "lua": {
         "tempMeas": {
            ...
         },
         "tempSetPoint": {
            ...
         },
         "tempMode": {
            ...
         }
      }
   }
]
```

  - Si <type of the IO> = "fan", <diRegLua> peut être:

    - fanSpeed * : Scripts qui gèrent la température mesurée. Il est associé au registre di_fanspeed dans l'éditeur Lua de GoldenGate.

Exemple

```
"fan": [
   {
      "param": {
         ...
```

```
      },
      "lua": {
         "fanSpeed": {
            ...
         }
      }
   }
]
```

◦ Si <type of the IO> = "vanes", <diRegLua> peut être :

- vanesPos * : Scripts qui gèrent la température mesurée. Il est associé au registre di_vanespos dans l'éditeur Lua de GoldenGate.

Exemple complet pour une IO aillettes

```
"vanes": [
   {
      "param": {
         "posMax": 5,
         "hasOffState": true,
         "hasAutoState": true,
         "hasSwingState": false
      },
      "lua": {
         "vanesPos": {
            "modbus2diOnly": 0,
            "di2modbusVars": [
               "local DI_ERROR = -1",
               "local DI_AUTOPOS = 252",
               "local DI_FROZENPOS = 253",
               "local DI_SWING = 254",
               "local di_vanespos = ..."
            ],
            "di2modbus": [
               "if (di_vanespos == DI_AUTOPOS) then",
               "    return 5",
               "elseif (di_vanespos == DI_FROZENPOS) then",
               "    return 6",
               "else",
               "    return di_vanespos",
               "end"
            ],
            "di2modbusRetInfo": [
               "Return value to be written to mb_vanes ModBus register",
               "eg: return 3"
            ],
            "modbus2diRegs": [
               {
                  "name": "mb_vanes",
                  "type": "HoldingReg",
                  "address": 21,
```

```
                "description": "Swing/Vane position. 0: Verical; 1: 30°; 2: 45°; 3: 60°; 4: Horizontal; 5: Auto
(swinging); 6: Off"
                }
            ],
            "modbus2diVars": [
                "local DI_ERROR = -1",
                "local DI_AUTOPOS = 252",
                "local DI_FROZENPOS = 253",
                "local DI_SWING = 254",
                "local mb_vanes = ..."
            ],
            "modbus2di": [
                "if (mb_vanes == 5) then",
                "    return DI_AUTOPOS",
                "elseif (mb_vanes == 6) then",
                "    return DI_FROZENPOS",
                "else",
                "    return mb_vanes",
                "end"
            ],
            "modbus2diRetInfo": [
                "Return value to be used in Domintell",
                "eg: return DI_SWING",
                "eg: return 1"
            ]
        }
    }
  }
],
```

- Si <type of the IO> = "status", <diRegLua> peut être :
  - error : Scripts qui gèrent les erreurs signalées par le dispositif ModBus. Il est associé au registre di_error dans l'éditeur Lua de GoldenGate.
  - version : Scripts qui gèrent la version renvoyée par le dispositif ModBus. Il est associé au registre di_version dans l'éditeur Lua de GoldenGate.
  - serial : Scripts qui gèrent les erreurs signalées par le dispositif ModBus. Il est associé au registre di_serial dans l'éditeur Lua de GoldenGate.

- Si <type of the IO> = "elec", <diRegLua> peut être :
  - elecFrequency : Scripts qui gèrent la tension mesurée sur L1. Il est associé à di_elecfreq dans l'éditeur Lua de GoldenGate.
  - elecVoltL1 : Scripts qui gèrent la tension mesurée sur L1. Il est associé à di_elecvolt1 dans l'éditeur Lua de GoldenGate.
  - elecVoltL2 : Scripts qui gèrent la tension mesurée sur L2. Il est associé à di_elecvolt2 dans l'éditeur Lua de GoldenGate.
  - elecVoltL3 : Scripts qui gèrent la tension mesurée sur L3. Il est associé à di_elecvolt3 dans l'éditeur Lua de GoldenGate.
  - elecPFL1 : Scripts qui gèrent le facteur de puissance mesuré sur L1. Il est associé à di_elecpfactor1 dans l'éditeur Lua de GoldenGate.

- elecPFL2 : Scripts qui gèrent le facteur de puissance mesuré sur L2. Il est associé à di_elecpfactor2 dans l'éditeur Lua de GoldenGate.
- elecPFL3 : Scripts qui gèrent le facteur de puissance mesuré sur L3. Il est associé à di_elecpfactor3 dans l'éditeur Lua de GoldenGate.
- elecCurrentL1 : Scripts qui gèrent le courant mesuré sur L1. Il est associé à di_eleccurrent1 dans l'éditeur Lua de GoldenGate.
- elecCurrentL2 : Scripts qui gèrent le courant mesuré sur L2. Il est associé à di_eleccurrent2 dans l'éditeur Lua de GoldenGate.
- elecCurrentL3 : Scripts qui gèrent le courant mesuré sur L3. Il est associé à di_eleccurrent3 dans l'éditeur Lua de GoldenGate.
- elecPowerL1 : Scripts qui gèrent la puissance active mesurée sur L1. Il est associé à di_elecpower1 dans l'éditeur Lua de GoldenGate.
- elecPowerL2 : Scripts qui gèrent la puissance active mesurée sur L2. Il est associé à di_elecpower2 dans l'éditeur Lua de GoldenGate.
- elecPowerL3 : Scripts qui gèrent la puissance active mesurée sur L3. Il est associé à di_elecpower3 dans l'éditeur Lua de GoldenGate.
- elecConsumedPower : Scripts qui gèrent la puissance globale consommée. Il est associé à di_elecconsumedpower dans l'éditeur Lua de GoldenGate.
- elecProducedPower : Scripts qui gèrent la puissance globale produite. Il est associé à di_elecproducedpower (doit être positif) dans l'éditeur Lua de GoldenGate.
- elecTotalPower : Scripts qui gèrent la puissance active totale mesurée (devrait être équivalente à elecconsumedpower - elecproducedpower). Elle doit être négative si la puissance produite est supérieure à la puissance consommée. Il est associé à di_electotalpower dans l'éditeur Lua de GoldenGate.
- elecEnergyL1 : Scripts qui gèrent l'énergie active accumulée sur L1. Il est associé à di_elecenergy1 dans l'éditeur Lua de GoldenGate.
- elecEnergyL2 : Scripts qui gèrent l'énergie active accumulée sur L2. Il est associé à di_elecenergy2 dans l'éditeur Lua de GoldenGate.
- elecEnergyL3 : Scripts qui gèrent l'énergie active accumulée sur L3. Il est associé à di_elecenergy3 dans l'éditeur Lua de GoldenGate.
- elecForwardEnergy : Scripts qui gèrent l'énergie globale accumulée vers l'avant. Il est associé à di_elecfwdenergy dans l'éditeur Lua de GoldenGate.
- elecReverseEnergy : Scripts qui gèrent l'énergie inverse accumulée globale. Il est associé à di_elecrevenergy (doit être positif) dans l'éditeur Lua de GoldenGate.
- elecTotalEnergy : Scripts qui gèrent l'énergie globale totale (devrait être équivalent à elecForwardEnergy - elecReverseEnergy). Elle doit être négative si plus d'énergie a été injectée dans le réseau que consommée depuis le réseau. Elle est associée à di_electotenergy dans l'éditeur Lua de GoldenGate.
- elecTotalEnergyTariff1 : Scripts qui gèrent l'énergie globale totale pendant la période tarifaire 1. Il doit être négatif si plus d'énergie a été injectée dans le réseau que consommée depuis le réseau pendant la période 1. Il est associé à di_electotenergyt1 dans l'éditeur Lua de GoldenGate.
- elecTotalEnergyTariff2 : Scripts qui gèrent l'énergie globale totale pendant la période tarifaire 2. Il doit être négatif si plus d'énergie a été injectée dans le réseau que consommée depuis le réseau pendant la période 2. Il est associé à di_electotenergyt2 dans l'éditeur Lua de

GoldenGate.

- elecTotalEnergyTariff3 : Scripts qui gèrent l'énergie globale totale pendant la période tarifaire 3. Il doit être négatif si plus d'énergie a été injectée dans le réseau que consommée depuis le réseau pendant la période 3. Il est associé à di_electotenergyt3 dans l'éditeur Lua de GoldenGate.
- elecTotalEnergyTariff4 : Scripts qui gèrent l'énergie globale totale pendant la période tarifaire 4. Il doit être négatif si plus d'énergie a été injectée dans le réseau que consommée depuis le réseau pendant la période 4. Il est associé à di_electotenergyt4 dans l'éditeur Lua de GoldenGate.
- elecTariffIndic : Scripts qui gèrent la période tarifaire active en cours Il est associé à di_electindicator dans l'éditeur Lua de GoldenGate.

◦ Si <type of the IO> = "relay", <diRegLua> peut être :
- relayState * : Scripts qui gèrent l'état de la sortie du relais. Il est associé au registre di_relaystate dans l'éditeur Lua de GoldenGate.

◦ Si <type of the IO> = "analogIn", <diRegLua> peut être :
- analog * : Scripts qui gèrent la valeur d'une entrée analogique brute. Il est associé au registre di_analog dans l'éditeur Lua de GoldenGate.

◦ Si <type of the IO> = "percentIn", <diRegLua> peut être :
- percent * : Scripts qui gèrent la valeur d'une entrée analogique en pourcentage (0-100%). Il est associé au registre di_percent dans l'éditeur Lua de GoldenGate.

◦ Si <type of the IO> = "evCharger", <diRegLua> peut être :
- *Coming soon...*

◦ Si <type of the IO> = "battery", <diRegLua> peut être :
- *Coming soon...*

Paramètres des scripts Lua

◦ modbusRegs * : Tableau d'objets JSON contenant la configuration des registres ModBus. Un maximum de cinq registres de 16 bits peut être attribué par Erreur : La source de référence n'a pas été trouvée lors de l'utilisation d'un DINTMB02.
◦ modbusRegs.name * : Nom du registre dans le script Lua, chaîne de caractères. **Doit commencer par "mb_", en minuscules uniquement, avec au moins une lettre après "mb_" et sans caractères spéciaux ni espaces.**
◦ modbusRegs.type * : Type de registre ModBus, chaîne enum. Les valeurs possibles sont :
- HoldingReg : Registre de maintien (16 bits). Fonctions ModBus : 0x03, 0x06 et 0x10.
- InputReg : Registre d'entrée (16 bits). Fonctions ModBus : 0x04.
- CoilReg : Registre à bobine (1 bit). Fonctions ModBus : 0x01, 0x05 et 0x0F.
- DiscreteReg : Registre discret (1 bit). Fonctions ModBus : 0x02.
- ExtensionReg : Registre virtuel utilisé pour déclarer l'espace/registre supplémentaire requis

pour stocker des valeurs de plus de 16 bits. A utiliser lorsque modbusRegs.nbrItems est supérieur à 1.

- modbusRegs.address * : Adresse du registre ModBus (basée sur 0), nombre entier (0 - 65535).

- modbusRegs.format : Format/mappage des données ModBus, chaîne enum. Les valeurs possibles sont :
  - 1bit : La largeur des données est de 1 bit.
  - S16 : La largeur des données est de 16 bits. Défaut.
  - S32-4321 : La largeur des données est de 32 bits. Si les données sont 0x44332211, 0x4433 sera stocké dans le registre 16 bits avec l'adresse la plus basse et 0x2211 sera stocké dans le registre 16 bits avec l'adresse la plus haute. Il s'agit du format 32 bits le plus utilisé.
  - S32-2143 : Entier signé de 32 bits. Si les données sont 0x44332211, 0x2211 sera stocké dans le registre 16 bits avec l'adresse la plus basse et 0x4433 sera stocké dans le registre 16 bits avec l'adresse la plus haute..
  - F32-IEEE : Nombre à virgule flottante de 32 bits. Stocké selon la norme IEEE-754 (https://fr.wikipedia.org/wiki/IEEE_754).
  - raw : données brutes/chaînes

- modbusRegs.nbrItems : Nombre de registres ModBus 16 bits nécessaires pour contenir les données.
- Si modbusRegs.type est ExtensionReg, c'est le décalage qui permet de les réorganiser lorsqu'on les assemble.

- modbusRegs.access : Type d'accès au registre ModBus, enum string. Les valeurs possibles sont :
  - rw : Le registre ModBus peut être lu et écrit. Défaut.
  - ro : Le registre ModBus ne peut être que lu.
  - wo : Le registre ModBus ne peut être qu'écrit.

- modbusRegs.description : Description du registre ModBus qui explique le format des données sur la base de la fiche technique du dispositif ModBus.

Exemple de registres ModBus pour temp.lua.tempMode

```
"modbusRegs": [
   {
      "name": "mb_op_mode",
      "type": "HoldingReg",
      "address": 3,
      "description": "Mode (0: auto; 1: heating; 2: fan; 3: cooling: 4: dry)"
   },
   {
      "name": "mb_mode_on_off",
      "type": "HoldingReg",
      "address": 5,
```

```
        "description": "On/Off the device (0: off; 1: on)"
    }
]
```

Exemple de registres ModBus pour elec.lua.elecCurrentL1

```
"modbus2diRegs": [
    {
        "name": "mb_curl1",
        "type": "HoldingReg",
        "address": 313,
        "format": "S32-4321",
        "nbrItems": 2,
        "access": "ro",
        "description": "The current of L1 in 0.001 A unit."
    },
    {
        "name": "mb_curl1_ext1",
        "type": "ExtensionReg",
        "address": 314,
        "format": "S32-4321",
        "nbrItems": 1,
        "access": "ro",
        "description": "The current of L1 in 0.001 A unit."
    }
]
```

- di2modbusConst : Tableau de chaînes de caractères contenant les déclarations des constantes utilisées dans les scripts. Cette section est utile pour notifier à l'utilisateur qu'il doit revoir le script lorsqu'il importe le script dans une version plus récente de GoldenGate alors que ces constantes ont été ajoutées/mises à jour. Cette section doit contenir la liste des constantes liées au registre et le nom du registre Domintell correspondant et le nom du registre Domintell correspondant.

Exemple pour le script de ventilateur fan.lua.fanSpeed

```
"di2modbusConst": [
    "local DI_OFF = 0",
    "local DI_AUTO = 254",
    "local DI_ERROR = -1",
    "local di_fanspeed = ..."
]
```

- di2modbusRetInfo : Chaîne/Aide concernant les valeurs renvoyées par le script. Ces informations doivent être basées sur la fiche technique du dispositif ModBus.

Exemple

```
"di2modbusRetInfo": [
    "Return value to be written to mb_fanspeed ModBus register",
    "eg: return 2"
]
```

- di2modbus : Tableau de chaînes du script Lua qui convertira la valeur de Domintell en une valeur adaptée au(x) registre(s) ModBus 16 bits. Chaque ligne du script est un élément/emplacement/index du tableau.

Exemple

```
"di2modbus": [
    "if (di_fanspeed == DI_AUTO) then",
    "    return 0",
    "else",
    "    return di_fanspeed",
    "end"
]
```

- modbus2diConst : Tableau de chaînes de caractères contenant les déclarations des constantes utilisées dans les scripts. Cette section est utile pour notifier à l'utilisateur qu'il doit revoir le script lorsqu'il importe le script vers une version plus récente de GoldenGate alors que ces constantes ont été ajoutées/mises à jour. Cette section doit contenir la liste des constantes liées au registre et le nom des registres ModBus définis modbus2diRegs
- modbus2diRetInfo : Chaîne/Aide sur les valeurs retournées par le script. Ces informations sont utilisées pour notifier l'utilisateur lorsqu'il importe le fichier JSON vers GoldenGate et que les arguments retournés ont été modifies.

- modbus2di : Tableau de chaînes du script Lua qui convertira la valeur du (des) registre(s) ModBus en une valeur adaptée au registre Domintell. Chaque ligne du script est un élément/emplacement/index du tableau.

Exemple

```
"modbus2di": [
    "if (mb_fanspeed == 0) then",
    "    return DI_AUTO",
    "else",
    "    return mb_fanspeed",
    "end"
]
```

Registres pour simulation

- simulRegs : Liste des registres ModBus à utiliser par un simulateur d'esclave ModBus, tableau d'objets JSON. **Non utilisé par GoldenGate**.
  - simulRegs[].name * : Nom du registre ModBus à simuler, chaîne de caractères

    Exemple

    ```
    "name": "H0002"
    ```

  - simulRegs[].type * : Type de registre ModBus à simuler, chaîne enum. Les valeurs possibles sont :
    - HoldingReg : Déclarer un registre d'attente.
    - InputReg : Déclarer un registre d'entrée.
    - CoilReg : Déclarer un registre de bobines.
    - DiscreteReg : Déclarer un registre d'entrée discret.

Exemple
"type" : "HoldingReg"

- simulRegs[].address * : Adresse du registre ModBus à simuler, nombre entier (1-65535). Adresse basée sur 1.

Exemple
"address": 2

- simulRegs[].description * : Description du registre ModBus à simuler, chaîne de caractères.

Exemple
"description": "Setpoint: 10°C to 30°C in heating mode"

- simulRegs[].range : Plage de valeurs valides que le registre ModBus à simuler peut accepter, tableau d'entiers.

Exemple
"range": [10, 32]

- simulRegs[].value : Valeur initiale/par défaut du registre ModBus à simuler, nombre entier.

Exemple
"value": 7

Exemple complet pour simulRegs

```
"simulRegs": [
   {
      "name": "H0002",
      "type": "HoldingReg",
      "address": 2,
      "description": "Setpoint: 10°C to 30°C in heating mode; 18°C to 32°C in cooling mode (Degrees C unit)",
      "range": [ 10, 32 ],
      "value": 21
   },
   {
      "name": "H0003",
      "type": "HoldingReg",
      "address": 3,
      "description": "FanSpeed (0: auto; 1-5: speed 1 to 5)",
      "range": [ 0, 5 ],
      "value": 0
   }
]
```

Exemple complet pour l'airco Coolmaster

```
{
   "name": "CoolMasterNet SI mode",
   "type": "Airco",
   "address": 80,
   "description": "CoolMasterNet bridge with L1 in SI master mode and L3 in CG5 slave mode.",
   "config": {
```

DINTMB02
Guide de configuration du fichier JSON


```
      "type": "serial",
      "param": {
         "baudrate": 9600,
         "databits": 8,
         "stopbits": 1,
         "parity": "none"
      }
   },
   "simulRegs": [
      {
         "name": "Op. Mode",
         "type": "HoldingReg",
         "address": 17,
         "description": "Operation mode (0: Cool; 1: heat; 2: Auto; 3: Dry; 4: HAUX; 5: Fan; 6: HH; 8: VAM Auto; 9: VAM Bypass; 10: VAM Heat Exc; 11: VAM normal)",
         "range": [ 0, 11 ],
         "value": 2
      },
      {
         "name": "Fan Speed",
         "type": "HoldingReg",
         "address": 18,
         "description": "FanSpeed (0-2: Lo-Me-Hi; 3: auto; 4: top; 5; very-low; 7: VAM Super Hi; 8; VAM low fresh-up; 9: VAM high fresh-up); VAM value does not work in SI mode",
         "range": [ 0, 9 ],
         "value": 3
      },
      {
         "name": "Setpoint",
         "type": "HoldingReg",
         "address": 19,
         "description": "Setpoint expressed in 10th degrees Celcius",
         "range": [ 0, 400 ],
         "value": 210
      },
      {
         "name": "On/off",
         "type": "HoldingReg",
         "address": 20,
         "description": "On/Off the device (0: off; 1: on)",
         "range": [ 0, 1 ],
         "value": 1
      },
      {
         "name": "Filter sign",
         "type": "HoldingReg",
         "address": 21,
         "description": "This register maybe warns when the filter must be clean clean (read) and ii is also used to clean this state (turn off warning light) (write)",
         "range": [ 0, 1 ],
         "value": 1
      },
      {
         "name": "Vanes",
```

```
            "type": "HoldingReg",
            "address": 22,
            "description": "Swing/Vane position. 0: Verical; 1: 30°; 2: 45°; 3: 60°; 4: Horizontal; 5: Auto (swinging); 6: Off",
            "range": [ 0, 6 ],
            "value": 5
        },
        {
            "name": "Room temp",
            "type": "HoldingReg",
            "address": 23,
            "description": "Temperature of the room in 10th degrees Celcius. Some HVAC models allow to set a suggested temperature room.",
            "range": [ 0, 400 ],
            "value": 210
        },
        {
            "name": "HVAC error code",
            "type": "HoldingReg",
            "address": 24,
            "description": "HVAC error code.",
            "range": [ 0, 65535 ],
            "value": 0
        },
        {
            "name": "Room temp",
            "type": "InputReg",
            "address": 18,
            "description": "Temperature of the room in 10th degrees Celcius.",
            "range": [ 0, 400 ],
            "value": 210
        },
        {
            "name": "HVAC error start of string",
            "type": "InputReg",
            "address": 19,
            "description": "First part of HVAC error string (stored in little endian mode)."
        },
        {
            "name": "HVAC error end of string",
            "type": "InputReg",
            "address": 20,
            "description": "Second part of HVAC error string (stored in little endian mode)."
        }
    ],
    "temp": [
        {
            "param": {
                "hasAccurateMeasTemp": false,
                "minSetpointStep": 1.0,
                "minCoolTemp": 18.0,
                "maxCoolTemp": 32.0,
                "minHeatTemp": 10.0,
                "maxHeatTemp": 30.0,
                "hasAutoMode": true,
```

```
        "hasHeatMode": true,
        "hasCoolMode": true,
        "hasFanMode": true,
        "hasDryMode": true
    },
    "lua": {
        "tempMeas": {
            "modbusRegs": [
                {
                    "name": "mb_roomtemp",
                    "type": "InputReg",
                    "address": 17,
                    "description": "Temperature of the room in 10th degrees Celcius."
                }
            ],
            "modbus2diConst": [
                "local mb_roomtemp = ..."
            ],
            "modbus2di": [
                "return mb_roomtemp / 10"
            ],
            "modbus2diRetInfo": [
                "Return temperature in floating decimal format to be used in Domintell",
                "eg: return 19.6",
                "eg: return mb_roomtemp / 10"
            ]
        },
        "tempSetpoint": {
            "di2modbusConst": [
                "local di_setpoint = ..."
            ],
            "di2modbus": [
                "return (di_setpoint * 10)"
            ],
            "di2modbusRetInfo": [
                "Return values to be written to <mb_setpoint> ModBus registers",
                "eg: return 200"
            ],
            "modbusRegs": [
                {
                    "name": "mb_setpoint",
                    "type": "HoldingReg",
                    "address": 18,
                    "description": "Setpoint expressed in 10th degrees Celcius"
                }
            ],
            "modbus2diConst": [
                "local mb_setpoint = ..."
            ],
            "modbus2di": [
                "return mb_setpoint / 10"
            ],
            "modbus2diRetInfo": [
                "Return temperature in floating decimal format to be used in Domintell",
```

```
                "eg: return 20.0",
                "eg: return mb_setpoint / 10"
            ]
        },
        "tempMode": {
            "di2modbusConst": [
                "local DI_OFF = 0",
                "local DI_HEAT = 1",
                "local DI_COOL = 2",
                "local DI_AUTO = 5",
                "local DI_DRY = 6",
                "local DI_FAN = 7",
                "local DI_ERROR = -1",
                "local di_mode = ..."
            ],
            "di2modbus": [
                "if (di_mode == DI_OFF) then",
                "    return 2,0",
                "elseif (di_mode == DI_HEAT) then",
                "    return 1,1",
                "elseif (di_mode == DI_COOL) then",
                "    return 0,1",
                "elseif (di_mode == DI_AUTO) then",
                "    return 2,1",
                "elseif (di_mode == DI_DRY) then",
                "    return 3,1",
                "elseif (di_mode == DI_FAN) then",
                "    return 5,1",
                "else",
                "    return DI_ERROR, DI_ERROR",
                "end"
            ],
            "di2modbusRetInfo": [
                "Return values to be written to <mb_op_mode> and <mb_mode_on_off> ModBus registers",
                "eg: return 2,0",
                "Attention : value must be returned in the same order than the declaration of the ModBus registers !"
            ],
            "modbusRegs": [
                {
                    "name": "mb_op_mode",
                    "type": "HoldingReg",
                    "address": 16,
                    "description": "Operation mode (0: Cool; 1: heat; 2: Auto; 3: Dry; 4: HAUX; 5: Fan; 6: HH; 8: VAM
Auto; 9: VAM Bypass; 10: VAM Heat Exc; 11: VAM normal)"
                },
                {
                    "name": "mb_mode_on_off",
                    "type": "HoldingReg",
                    "address": 19,
                    "description": "On/Off the device (0: off; 1: on)"
                }
            ],
            "modbus2diConst": [
                "local DI_OFF = 0",
```

```
                "local DI_HEAT = 1",
                "local DI_COOL = 2",
                "local DI_AUTO = 5",
                "local DI_DRY = 6",
                "local DI_FAN = 7",
                "local DI_ERROR = -1",
                "local mb_op_mode, mb_mode_on_off = ..."
            ],
            "modbus2di": [
                "if (mb_mode_on_off == 0) then",
                "    return DI_OFF",
                "elseif (mb_op_mode == 0) then",
                "    return DI_COOL",
                "elseif (mb_op_mode == 1) then",
                "    return DI_HEAT",
                "elseif (mb_op_mode == 2) then",
                "    return DI_AUTO",
                "elseif (mb_op_mode == 3) then",
                "    return DI_DRY",
                "elseif (mb_op_mode == 5) then",
                "    return DI_FAN",
                "else",
                "    return DI_ERROR",
                "end"
            ],
            "modbus2diRetInfo": [
                "Return value to be used in Domintell",
                "eg: return DI_HEAT",
                "DI_xxx variables must be used ! Do not use numeric values !"
            ]
        }
    }
  }
],
"fan": [
  {
    "param": {
      "speedMax": 5,
      "hasOffState": false,
      "hasAutoState": true
    },
    "lua": {
      "fanSpeed": {
        "di2modbusConst": [
          "local DI_OFF = 0",
          "local DI_AUTO = 254",
          "local DI_ERROR = -1",
          "local di_fanspeed = ..."
        ],
        "di2modbus": [
          "if (di_fanspeed == DI_AUTO) then",
          "    return 3",
          "elseif (di_fanspeed >= 6) then",
          "    return di_fanspeed + 1",
```

```
                    "elseif (di_fanspeed > 3) then",
                    "    return di_fanspeed",
                    "else",
                    "    return di_fanspeed - 1",
                    "end"
                ],
                "di2modbusRetInfo": [
                    "Return value to be written to mb_fan ModBus register",
                    "eg: return 2"
                ],
                "modbusRegs": [
                    {
                        "name": "mb_fan",
                        "type": "HoldingReg",
                        "address": 17,
                        "description": "FanSpeed (0-2: Lo-Me-Hi; 3: auto; 4: top; 5; very-low; 7: VAM Super Hi; 8; VAM low
fresh-up; 9: VAM high fresh-up); VAM value does not work in SI mode"
                    }
                ],
                "modbus2diConst": [
                    "local DI_ERROR = -1",
                    "local DI_OFF = 0",
                    "local DI_AUTO = 254",
                    "local mb_fan = ..."
                ],
                "modbus2di": [
                    "if (mb_fan == 3) then",
                    "    return DI_AUTO",
                    "elseif (mb_fan < 3) then",
                    "    return mb_fan + 1",
                    "elseif (mb_fan >= 7) then",
                    "    return mb_fan - 1",
                    "else",
                    "    return mb_fan",
                    "end"
                ],
                "modbus2diRetInfo": [
                    "Return value to be used in Domintell",
                    "eg: return DI_OFF",
                    "eg: return 2"
                ]
            }
        }
    }
],
"vanes": [
    {
        "param": {
            "posMax": 5,
            "hasOffState": true,
            "hasAutoState": true,
            "hasSwingState": false
        },
        "lua": {
```

DINTMB02
Guide de configuration du fichier JSON


```
            "vanesPos": {
               "di2modbusConst": [
                  "local DI_ERROR = -1",
                  "local DI_AUTOPOS = 252",
                  "local DI_FROZENPOS = 253",
                  "local DI_SWING = 254",
                  "local di_vanespos = ..."
               ],
               "di2modbus": [
                  "if (di_vanespos == DI_AUTOPOS) then",
                  "    return 5",
                  "elseif (di_vanespos == DI_FROZENPOS) then",
                  "    return 6",
                  "else",
                  "    return di_vanespos",
                  "end"
               ],
               "di2modbusRetInfo": [
                  "Return value to be written to mb_vanes ModBus register",
                  "eg: return 3"
               ],
               "modbusRegs": [
                  {
                     "name": "mb_vanes",
                     "type": "HoldingReg",
                     "address": 21,
                     "description": "Swing/Vane position. 0: Verical; 1: 30°; 2: 45°; 3: 60°; 4: Horizontal; 5: Auto
(swinging); 6: Off"
                  }
               ],
               "modbus2diConst": [
                  "local DI_ERROR = -1",
                  "local DI_AUTOPOS = 252",
                  "local DI_FROZENPOS = 253",
                  "local DI_SWING = 254",
                  "local mb_vanes = ..."
               ],
               "modbus2di": [
                  "if (mb_vanes == 5) then",
                  "    return DI_AUTOPOS",
                  "elseif (mb_vanes == 6) then",
                  "    return DI_FROZENPOS",
                  "else",
                  "    return mb_vanes",
                  "end"
               ],
               "modbus2diRetInfo": [
                  "Return value to be used in Domintell",
                  "eg: return DI_SWING",
                  "eg: return 1"
               ]
            }
         }
      }
```

DINTMB02
Guide de configuration du fichier JSON

```
  ],
  "status": [
    {
      "param": {
      },
      "lua": {
        "error": {
          "modbusRegs": [
            {
              "name": "mb_error",
              "type": "HoldingReg",
              "address": 23,
              "description": "HVAC error code."
            }
          ],
          "modbus2diConst": [
            "local DI_ERROR = -1",
            "local DI_LVLNORMAL = 0",
            "local DI_LVLWARNING = 1",
            "local DI_LVLCRITICAL = 2",
            "local mb_error = ..."
          ],
          "modbus2di": [
            "if (mb_error == 0) then",
            "    return DI_LVLNORMAL, mb_error, \"Pas d'erreur\"",
            "elseif (mb_error == 666) then",
            "    return DI_LVLCRITICAL, mb_error, \"Devil is inside\"",
            "else",
            "    return DI_LVLWARNING, mb_error, \"Unkonwn error\"",
            "end"
          ],
          "modbus2diRetInfo": [
            "Return values to be used in Domintell",
            "return <error_level>, <error_value>, <error_infostr>",
            "eg: return DI_LVLCRITICAL, 45, \"Water pump is faulty\"",
            "eg: return DI_LVLNORMAL, 0, \"\""
          ]
        }
      }
    }
  ]
}
```

DINTMB02
Guide de configuration du fichier JSON

Alphabetical Index

DINTMB02
Guide de configuration du fichier JSON

DINTMB02
Guide de configuration du fichier JSON

DINTMB02
JSON-bestand configuratiegids

1. Presentatie

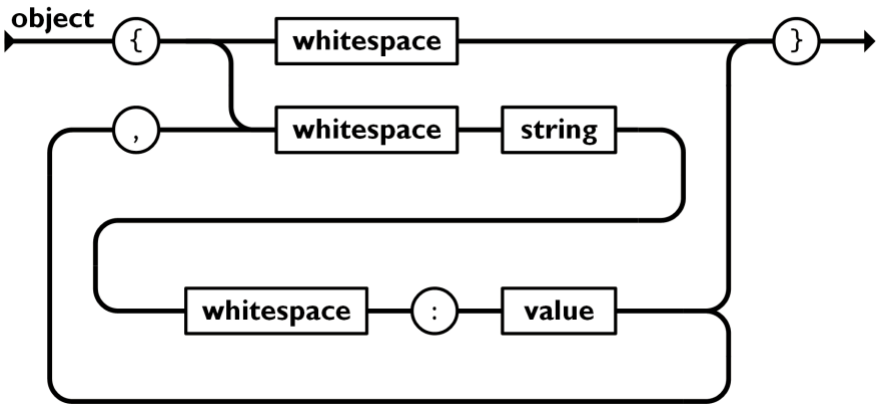
JSON-bestanden zijn een eenvoudig lees/schrijfformaat voor het uitwisselen van configuraties of gegevens tussen systemen met een verschillende architectuur/taal. Het kan ook worden gebruikt om configuraties te importeren/exporteren om ze eenvoudig aan te passen zonder dat er specifieke software nodig is.

2. JSON-opmaak

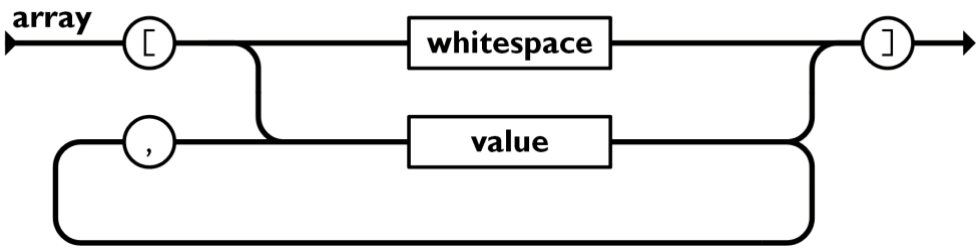
(source: https://www.json.org/json-nl.html)

JSON-bestanden hebben een strikte opmaakspecificatie en moeten strikt worden gerespecteerd, anders is het bestand onleesbaar voor het systeem/de software.

- Een object is een ongeordende verzameling van naam/waardeparen. Een object begint met een linker accolade ('{') en eindigt met een rechter accolade ('}'). Elke naam wordt gevolgd door een dubbele punt (':') en de naam/waardeparen worden gescheiden door een komma (',').

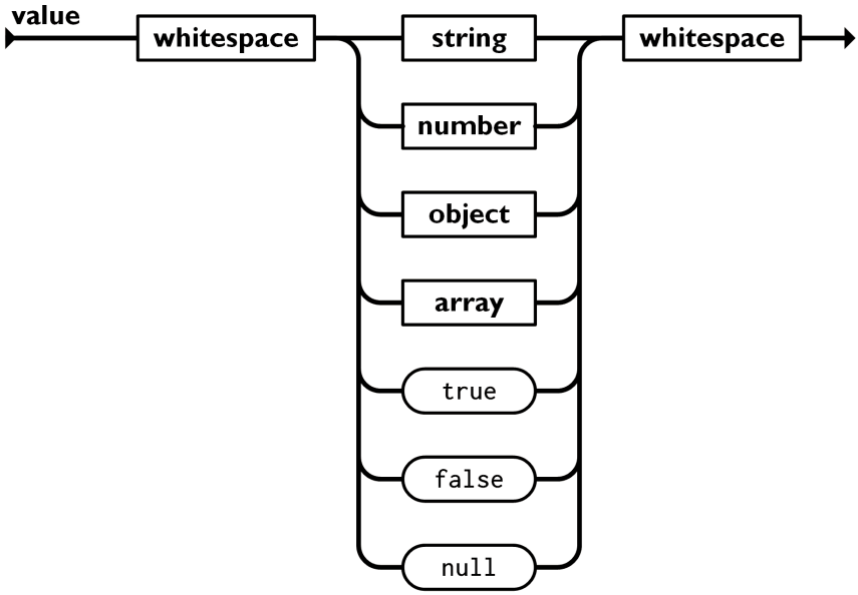


- Een array is een geordende verzameling van waarden. Een array begint met een haakje links ('[') en eindigt met een haakje rechts (']'). Waarden worden gescheiden door een komma (',').

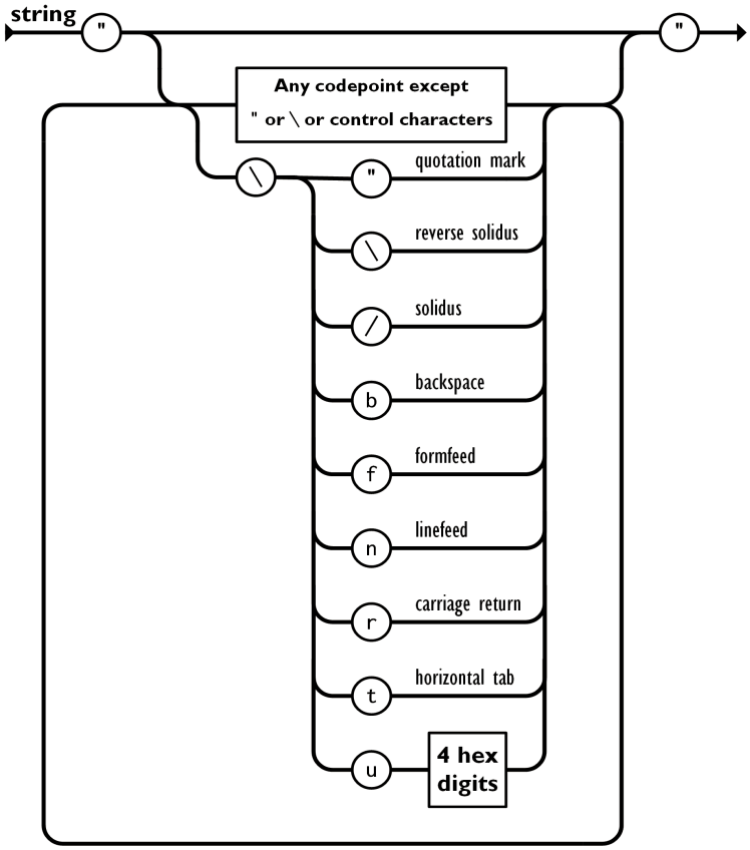


DINTMB02
JSON-bestand configuratiegids

- Een *waarde* kan een *tekenreeks* tussen dubbele aanhalingstekens zijn, of een *getal*, of waar of onwaar of nul, of een *object* of een array. Deze structuren kunnen genest worden.

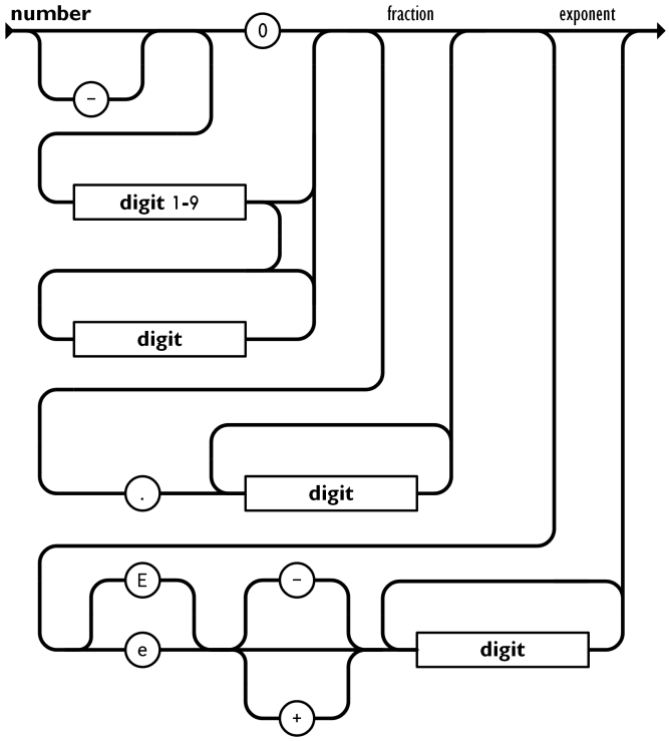


- Een *tekenreeks* is een opeenvolging van nul of meer Unicode-tekens, tussen dubbele aanhalingstekens (""), met backslash-escapes ('\'). Een karakter wordt weergegeven als een enkele tekenreeks. Een string



is als een C of Java string.

- Een *getal* lijkt veel op een C- of Java-nummer, behalve dat de octale en hexadecimale formaten niet worden gebruikt.



- Hier is een voorbeeld van een JSON fragment:

Voorbeeld

```
"config": {
    "type": "serial",
    "param": {
        "baudrate": 9600,
     "databits": 8,
     "stopbits": 1,
     "parity":"none"
  }
}
```

Domintell registers

- di_temperature
  - Beschrijving: Bevat de gemeten temperatuur in Celsius graden in decimale drijvende komma-indeling (bijv. 3,4).
  - Geassocieerde constante:
    - Geen
  - Bijbehorende waarden/argumenten/leden:
    - 1st : temperatuur meten
  - Lua opdracht voorbeeld:
    - return 12.3
- di_setpoint
  - Beschrijving: Bevat het temperatuurinstelpunt in Celsius graden in decimale drijvende komma-indeling (bijv. 3,4).
  - Geassocieerde constante:
    - Geen
  - Bijbehorende waarden/argumenten/leden:
    - 1st: the current set-point
  - Lua opdracht voorbeeld:
    - return 21.0
- di_mode
  - Beschrijving: Bevat de regelmodus in integer formaat (gebruik de meegeleverde constanten) (bijv. DI_DRY).
  - Geassocieerde constante:
    - DI_OFF = 0
    - DI_HEAT = 1
    - DI_COOL = 2
    - DI_AUTO = 5
    - DI_DRY = 6
    - DI_FAN = 7
    - DI_ERROR = -1
  - Bijbehorende waarden/argumenten/leden:
    - 1st: de huidige reguleringsmodus
  - Lua opdracht voorbeeld:
    - return DI_HEAT
- di_error
  - Beschrijving: Bevat foutinformatie. Formaat: <error level>, <description>, <error code> (bijv. DI_LVLCRITICAL, "pump defect", 65).
  - Geassocieerde constante:
    - DI_LVLNORMAL = 0: Normaal/Informatieniveau
    - DI_LVLWARNING = 1: Waarschuwingsniveau

        - DI_LVLCRITICAL = 3: Fout/kritiek niveau
        - DI_ERROR = -1
    - Bijbehorende waarden/argumenten/leden:
        - 1st: temperatuur meten
    - Lua opdracht voorbeeld:
        - return DI_LVLCRITICAL, "pump defect", 65
- di_fanspeed
    - Beschrijving: Bevat de snelheid van de ventilator in integer formaat (opgegeven constanten of aangepaste waarden kunnen worden gebruikt) (bijv. DI_AUTO of 3).
    - Geassocieerde constante:
        - DI_AUTO = 254
    - Bijbehorende waarden/argumenten/leden:
        - 1st: de huidige snelheid van de ventilator.
    - Lua opdracht voorbeeld:
        - return 1
        - return DI_AUTO
- di_vanespos
    - Beschrijving: Bevat de snelheid van de ventilator in integer formaat (opgegeven constanten of aangepaste waarden kunnen worden gebruikt) (bijv. DI_AUTOPOS of 2).
    - Geassocieerde constante:
        - DI_AUTOPOS = 252: Autopositie waarmee het apparaat automatisch de positie kiest.
        - DI_FROZENPOS = 253: Hiermee kunnen schoepen op een specifieke positie worden stilgezet terwijl ze slingeren..
        - DI_SWING = 254: Zwenkschoepen.
    - Bijbehorende waarden/argumenten/leden:
        - 1st: de huidige positie van de schoepen.
    - Lua opdracht voorbeeld:
        - return 2
        - return DI_SWING
- di_elecfreq
    - Beschrijving: Bevat de frequentie van het elektrische netwerk in de eenheid Hz. In decimaal floating-point formaat (bijv. 50.1 = 50.1 Hz).
    - Geassocieerde constante:
        - Geen
    - Bijbehorende waarden/argumenten/leden:
        - 1st: de gemeten frequentie van het net.
    - Lua opdracht voorbeeld:
        - return 50.1
- di_elecpfactor1, di_elecpfactor2, di_elecpfactor3
    - Beschrijving: Bevat de arbeidsfactor van fase 1, 2 en 3. In decimaal floating-point formaat (bijv. 0,95).

- Geassocieerde constante:
  - Geen
- Bijbehorende waarden/argumenten/leden:
  - 1st: De gemeten arbeidsfactor.
- Lua opdracht voorbeeld:
  - return 0.95

- di_elecvolt1, di_elecvolt2, di_elecvolt3
  - Beschrijving: Bevat de spanning van fase 1, 2 en 3 in de eenheid V. In decimaal floating-point formaat (bijv. 239,3 = 239,3 V).
  - Geassocieerde constante:
    - Geen
  - Bijbehorende waarden/argumenten/leden:
    - 1st: De gemeten spanning van de lijn.
  - Lua opdracht voorbeeld:
    - return 239.3
- di_eleccurrent1, di_eleccurrent2, di_eleccurrent3
  - Beschrijving: Bevat de stroom van fase 1, 2 en 3 in eenheid A. In decimaal floating-point formaat (bijv. 12,4 = 12,4 A).
  - Geassocieerde constante:
    - Geen
  - Bijbehorende waarden/argumenten/leden:
    - 1st: De gemeten stroom van de lijn.
  - Lua opdracht voorbeeld:
    - return 12.4
- di_elecpower1, di_elecpower2, di_elecpower3
  - Beschrijving: Bevat het momentane vermogen van fase 1, 2 en 3 in de eenheid W. In geheel getalformaat (bijv. 12387 = 12,387 kW).
  - Geassocieerde constante:
    - Geen
  - Bijbehorende waarden/argumenten/leden:
    - 1st: temperatuur meten
  - Lua opdracht voorbeeld:
    - return 12.3
- di_elecconsumedpower
  - Beschrijving: Bevat het momentane verbruikte vermogen in de eenheid W. In het formaat unsigned integer (bv. 12387 = 12,387 kW). De waarde moet positief zijn.
  - Geassocieerde constante:
    - Geen
  - Bijbehorende waarden/argumenten/leden:
    - 1st: temperatuur meten

- Lua opdracht voorbeeld:
  - return 12.3

- di_elecproducedpower
  - Beschrijving: Bevat het momentane geproduceerde vermogen in de eenheid W. In het formaat unsigned integer (bv. 12387 = 12,387 kW). De waarde moet positief zijn.
  - Geassocieerde constante:
    - Geen
  - Bijbehorende waarden/argumenten/leden:
    - 1st: temperatuur meten
  - Lua opdracht voorbeeld:
    - return 12.3
- di_electotalpower
  - Beschrijving: Bevat het momentane totale vermogen in de eenheid W. In afgetekend geheel getalformaat (bv. 12387 = 12,387 kW). Als de waarde negatief is, is er meer vermogen geproduceerd dan verbruikt.
  - Geassocieerde constante:
    - Geen
  - Bijbehorende waarden/argumenten/leden:
    - 1st: temperatuur meten
  - Lua opdracht voorbeeld:
    - return 12.3
- di_elecenergy1, di_elecenergy2, di_elecenergy3
  - Beschrijving: Bevat de energie van fase 1, 2 en 3 in de eenheid kWh. In getekend geheel getalformaat (bijv. 1238 = 1,238 MWh). Als de waarde negatief is, is er meer vermogen geproduceerd dan verbruikt.
  - Geassocieerde constante:
    - Geen
  - Bijbehorende waarden/argumenten/leden:
    - 1st: temperatuur meten
  - Lua opdracht voorbeeld:
    - return 12.3
- di_elecfwdenergy
  - Beschrijving: Bevat de voorwaartse energie in de eenheid kWh. In het formaat unsigned integer (bijv. 1238 = 1,238 MWh). De waarde moet positief zijn.
  - Geassocieerde constante:
    - Geen
  - Bijbehorende waarden/argumenten/leden:
    - 1st: temperatuur meten
  - Lua opdracht voorbeeld:
    - return 12.3
- di_elecrevenergy

- Beschrijving: Bevat de omgekeerde energie in de eenheid kWh. In het formaat unsigned integer (bijv. 1238 = 1,238 MWh). De waarde moet positief zijn..
- Geassocieerde constante:
  - Geen
- Bijbehorende waarden/argumenten/leden:
  - 1st: temperatuur meten
- Lua opdracht voorbeeld:
  - return 12.3

- di_electotenergy
  - Beschrijving: Bevat de totale energie in de eenheid kWh. In getekend geheel getalformaat (bijv. 1238 = 1,238 MWh). Als de waarde negatief is, is er meer energie geproduceerd dan verbruikt.
  - Geassocieerde constante:
    - Geen
  - Bijbehorende waarden/argumenten/leden:
    - 1st: temperatuur meten
  - Lua opdracht voorbeeld:
    - return 12.3
- di_electotenergyt1, di_electotenergyt2, di_electotenergyt3, di_electotenergyt4
  - Beschrijving: Bevat de energie in kWh eenheid verbruikt in de tarieven 1, 2, 3 en 4 periode. In getekend geheel getalformaat (bijv. 1238 = 1,238 MWh). Als de waarde negatief is, is er meer energie geproduceerd dan verbruikt.
  - Geassocieerde constante:
    - Geen
  - Bijbehorende waarden/argumenten/leden:
    - 1st: temperatuur meten
  - Lua opdracht voorbeeld:
    - return 12.3
- di_electindicator
  - Beschrijving: Bevat de huidige actieve tariefperiode. In het formaat unsigned integer (bv. 2 = tarief periode 2).
  - Geassocieerde constante:
    - Geen
  - Bijbehorende waarden/argumenten/leden:
    - 1st: temperatuur meten
  - Lua opdracht voorbeeld:
    - return 12.3
- di_relaystate
  - Beschrijving: Bevat de huidige status van het relais. Gebruik de opgegeven constanten (bijv. DI_ON = relais aan).
  - Geassocieerde constante:

        - Geen
    - Bijbehorende waarden/argumenten/leden:
        - 1st: temperatuur meten
    - Lua opdracht voorbeeld:
        - return 12.3
- di_version
    - Beschrijving: Bevat de huidige versie van het apparaat. Formaat <raw string>, <dot-decimaal getal> (bijv. "2.1 2024/01/01", 2.1).
    - Geassocieerde constante:
        - Geen
    - Bijbehorende waarden/argumenten/leden:
        - 1st: temperatuur meten
    - Lua opdracht voorbeeld:
        - return 12.3
- di_percent
    - Beschrijving: Bevat het percentage van een analoge ingang. Een geheel getal van 0 tot 100 (bijv. 70 = 70%).
    - Geassocieerde constante:
        - Geen
    - Bijbehorende waarden/argumenten/leden:
        - 1st: temperatuur meten
    - Lua opdracht voorbeeld:
        - return 12.3
- di_analog
    - Beschrijving: Bevat de waarde van een analoge ingang. Formaat <decimaal zweven-punt>, [<eenheidstekenreeks>] (bijv. 7,39, "pH" = 7,39 pH).
    - Geassocieerde constante:
        - Geen
    - Bijbehorende waarden/argumenten/leden:
        - 1st: temperatuur meten
    - Lua opdracht voorbeeld:
        - return 12.3
- di_serial
    - Beschrijving: Bevat het serienummer van het apparaat. In tekenreeksformaat (bijvoorbeeld "162VX5687T").
    - Geassocieerde constante:
        - Geen
    - Bijbehorende waarden/argumenten/leden:
        - 1st: temperatuur meten
    - Lua opdracht voorbeeld:
        - return "

DINTMB02
JSON-bestand configuratiegids

JSON fields

Velden gevolgd door een rood sterretje ('*') zijn verplicht

Wereldwijd

- name *: Naam van de module, string. Deze naam wordt toegewezen aan de module in GoldenGate en kan ook worden gebruikt om de naam van IO's van de module op te bouwen.

  Voorbeeld

  ```
  "name": "Daikin RTD-RA"
  ```

- type *: Type ModBus apparaat, enum string. Mogelijke waarden zijn:
  - Airco: Maakt een airconditionermodule met een temperatuursensor, een ventilator en een schoepen IO.
  - Generic: Maakt een generieke module

  Voorbeeld

  ```
  "type": "Airco"
  ```

- address *: ModBus-adres van het apparaat, geheel getal (0-255).

  Voorbeeld

  ```
  "address": 80
  ```

- Beschrijving: Beschrijving van het ModBus apparaat, string. Wordt genegeerd bij het importeren naar GoldenGate.

  Voorbeeld

  ```
  "description": "Daikin ModBus interface to interact with Daikin VRV airconditioner"
  ```

Configuratie

- config *: Configuratie van de communicatie, lijst van JSON objecten.
  - config.type *: Type protocol, enum string. Mogelijke waarden zijn:
    - serial: Serieel ModBus-apparaat met ModBus RTU-protocol.
    - tcp: Ethernet ModBus-apparaat met ModBus TCP-protocol.

    Voorbeeld

    ```
    "type": "serial"
    ```

- config.param *: Parameters van de communicatie, lijst van JSON objecten.
  - For config.type = "serial":
    - config.param.baudrate *: Baudrate, enum integer. Mogelijke waarden zijn:
      - 1200: 1200 bauds.
      - 2400: 2400 bauds.
      - 4800: 4800 bauds.
      - 9600: 9600 bauds.
      - 38400: 38400 bauds.
      - 57600: 57600 bauds.
      - 115200: 115200 bauds.
    - config.param.databits *: Aantal bits per woord, enum interger. Mogelijke waarden zijn:
      - 8: 8 bits per woord.
    - config.param.stopbits *: Aantal stopbits, enum interger. Mogelijke waarden zijn:
      - 1: 1 stopbit
      - 3: 1.5 stopbit
      - 2: 2 stopbits
    - config.param.parity *: Type pariteit, enum string. Mogelijke waarden zijn:
      - none: Geen pariteit.
      - even: Even pariteit.
      - odd: Oneven pariteit.

Voorbeeld

```
"type": "serial"
"parameters": {
   "baudrate": 9600,
   "databits": 8,
   "stopbits": 1,
   "parity": "none"
}&
```

  - Voor config.type = "tcp":
    - config.param.host *: IP-adres van het apparaat, IPv4-string.
    - config.param.port *: TCP poort, interger (1-65535, meestal 502).

Voorbeeld

```
"type": "tcp"
"parameters": {
   "host": "102.168.1.200",
   "port": 502
}
```

Volledig voorbeeld voor configuratie

```
"config": {
```

```
    "type": "serial",
    "parameters": {
        "baudrate": 9600,
        "databits": 8,
        "stopbits": 1,
        "parity": "none"
    }
```

- }

Type IO

- <type of the IO> key *: Declareert een lijst met IO's van het gespecificeerde type, enum string. Mogelijke waarden zijn:
    - temp: Een temperatuursensor aangeven. Verplicht indien type = "Airco".
    - fan: Declareer een ventilator IO. Verplicht indien type = "Airco".
    - vanes: Geef een schoepen IO op. Verplicht indien type = "Airco".
    - status: Declareer een status-IO om problemen van het ModBus-apparaat te rapporteren.
    - elec: Een energiemeter IO aangeven.
    - relay: Een relaisuitgang declareren.
    - percentIn: Geef een percentage (0-100%) op als invoer.
    - analogIn: Een analoge ingang declareren.

- <type of the IO> value *: Array van JSON objecten. Elk object is een IO. Het voorbeeld bevat twee temperatuursensoren en een ventilator-IO.

Voorbeeld

```
"temp": [
    {
        "param": {
            ...
        },
        "lua": {
            ...
        }
    },
    {
        "param": {
            ...
        },
        "lua": {
            ...
        }
    }
],
"fan": [
    {
        "param": {
            ...
        },
```

```
    "lua": {
      ...
    }
  }
[
```

IO-parameters

- <type of the IO>.param *: Parameters van de IO, lijst van JSON objecten. De sleutel van elk object is een enum string. Als een type IO geen parameters heeft, moet er toch een leeg "param" knooppunt aanwezig zijn. Mogelijke waarden hangen af van het type IO:
  - als <type of the IO> key = "temp":
    - hasAccurateMeasTemp: Geeft aan of de gemeten temperatuur nauwkeurig is. Sommige airconditioners hebben alleen een retourluchttemperatuur die niet precies hetzelfde is als de temperatuur in de kamer, booleaans (waar of onwaar). Standaard: true.

      Voorbeeld
      ```
      "hasAccurateMeasTemp": false
      ```

    - minSetpointStep : Minimale incrementele stap voor de gewenste waarde, drijvend puntgetal. Als de waarde bijvoorbeeld is ingesteld op 5,0, kan de gewenste waarde alleen worden verhoogd met 5,0 °C of een veelvoud daarvan. Standaard: 0,5 °C.

      Voorbeeld
      ```
      "minSetpointStep": 0.5
      ```

    - minCoolTemp: Minimale temperatuur in koelmodus die het ModBus-apparaat aankan, Celcius graden als getal met drijvende komma. Standaard: 18,0 °C.

      Voorbeeld
      ```
      "minCoolTemp": 18.0
      ```

    - maxCoolTemp: Maximale temperatuur in koelmodus die het ModBus-apparaat aankan, Celcius graden als getal met drijvende komma. Standaard: 32,0 °C

      Voorbeeld
      ```
      "maxCoolTemp": 32.0
      ```

    - minHeatTemp: Minimale temperatuur in de warmtemodus die het ModBus-apparaat aankan, Celcius graden als floating-point getal. Standaard: 10,0 °C.

      Voorbeeld
      ```
      "minHeatTemp": 10.0
      ```

    - maxHeatTemp: Maximale temperatuur in de warmtemodus die het ModBus-apparaat aankan, Celcius graden als drijvend komma getal. Standaard: 30,0 °C.

      Voorbeeld

```
"maxHeatTemp": 30.0
```

- hasAutoMode: Geeft aan of het ModBus-apparaat de automatische regelmodus aankan, booleaans (true of false). Standaard: true.

Voorbeeld
```
"hasAutoMode": true
```

- hasHeatMode: Geeft aan of het ModBus-apparaat de warmteregelingsmodus aankan, booleaans (true of false). Standaard: true.

Voorbeeld
```
"hasHeatMode": true
```

- hasCoolMode: Geeft aan of het ModBus-apparaat de koelregelmodus kan verwerken, booleaans (true of false). Standaard: true.

Voorbeeld
```
"hasCoolMode": true
```

- hasFanMode: Geeft aan of het ModBus-apparaat de ventilatorregelmodus kan verwerken, booleaans (true of false). Standaard: true.

Voorbeeld
```
"hasFanMode": true
```

- hasDryMode: Geeft aan of het ModBus-apparaat de droge regelmodus aankan, booleaans (true of false). Standaard: true.

Voorbeeld
```
"hasDryMode": true
```

◦ als <type of the IO> key = "fan" :

- speedMax: Geeft aan hoeveel snelheid het ModBus-apparaat heeft, geheel getal (0-200). Uitgezonderd de uit- en de auto-status. Standaard: 0.

Voorbeeld
```
"speedMax": 5
```

- hasOffState: Geeft aan of de ventilator van het ModBus-apparaat kan worden uitgeschakeld, booleaans (true of false). Standaard: true.

Voorbeeld
```
"hasOffState": true
```

- hasAutoState: Geeft aan of de ventilator van het ModBus-apparaat een autostatus heeft (snelheid automatisch beheerd door het ModBus-apparaat, booleaans (true of false).

Standaard: true.

Voorbeeld
"hasOffState": true

- als <type of the IO> key = "vanes":
  - posMax: Geeft aan hoeveel snelheid het ModBus-apparaat heeft, geheel getal (0-200). Uitgezonderd de uit-, de auto- en de schommeltoestand. Standaard: 0.

    Voorbeeld
    "posMax": 5
  - hasOffState: Geeft aan of schoepen een uit/gesloten status hebben, booleaans (true of false). Standaard: false.

    Voorbeeld
    "hasOffState": true
  - hasAutoState: Geeft aan of vinnen een auto-status hebben, booleaans (true of false). Standaard: false.

    Voorbeeld
    "hasAutoState": false
  - hasSwingState: Geeft aan of schoepen een schommelstand hebben, booleaans (true of false). Standaard: true.

    Voorbeeld
    "hasSwingState": true
- als <type of the IO> key = "elec":
  - Geen
- als <type of the IO> key = "evCharger":
  - Geen
- als <type of the IO> key = "battery":
  - Geen
- als <type of the IO> key = "relay":
  - Geen
- als <type of the IO> key = "percentIn" :

- Geen

◦ als <type of the IO> key = "analogIn" :
- Geen

◦ als <type of the IO> key = "status" :
- Geen

Voorbeeld wanneer de IO parameters heeft

```
"temp": {
    "param": {
        "hasAccurateMeasTemp": false,
        "minSetpointStep": 1.0,
        "minCoolTemp": 18.0,
        "maxCoolTemp": 32.0,
        "minHeatTemp": 10.0,
        "maxHeatTemp": 30.0,
        "hasAutoMode": true,
        "hasHeatMode": true,
        "hasCoolMode": true,
        "hasFanMode": true,
        "hasDryMode": true
    },
    "lua": {
        ...
    }
}
```

Voorbeeld wanneer IO geen parameters heeft

```
"elec" {
    "param": {
    },
    "lua": {
        ...
    }
}
```

DINTMB02
JSON-bestand configuratiegids

Categorie van Lua scripts

- <type of the IO>.lua * : Alle gegevens en scripts om waarden van/naar ModBus te converteren met Lua-scripts, lijst van Domintell-registers als JSON-objecten. Beschikbare Domintell-registers (<diRegLua>) afhankelijk van < type of the IO > waarde:

  - als <type of the IO> = "temp", <diRegLua> kunnen:

    - tempMeas * : Scripts die de gemeten temperatuur verwerken. Het is gekoppeld aan het register di_temperature in de Lua-editor in GoldenGate. Alleen verplicht als "temp.param.hasAccurateMeasTemp" = true.

    - tempSetpoint * : Scripts die de gemeten temperatuur verwerken. Het is gekoppeld aan het register di_setpoint in de Lua-editor in GoldenGate.

    - tempMode * : Scripts die de gemeten temperatuur verwerken. Het is gekoppeld aan het di_mode register in de Lua editor in GoldenGate.

Voorbeeld

```
"temp": [
   {
      "param": {
         ...
      },
      "lua": {
         "tempMeas": {
            ...
         },
         "tempSetPoint": {
            ...
         },
         "tempMode": {
            ...
         }
      }
   }
]
```

  - als <type of the IO> = "fan", <diRegLua> kunnen:

    - fanSpeed * : Scripts die de gemeten temperatuur verwerken. Het is gekoppeld aan het register di_fanspeed in de Lua-editor in GoldenGate.

Voorbeeld

```
"fan": [
   {
      "param": {
         ...
```

```
      },
      "lua": {
         "fanSpeed": {
            ...
         }
      }
   }
]
```

◦ als <type of the IO> = "vanes", <diRegLua> kunnen:

▪ vanesPos * : Scripts die de gemeten temperatuur verwerken. Het is gekoppeld aan het register di_vanespos in de Lua-editor in GoldenGate.

Volledig voorbeeld voor een vanen IO

```
"vanes": [
   {
      "param": {
         "posMax": 5,
         "hasOffState": true,
         "hasAutoState": true,
         "hasSwingState": false
      },
      "lua": {
         "vanesPos": {
            "modbus2diOnly": 0,
            "di2modbusVars": [
               "local DI_ERROR = -1",
               "local DI_AUTOPOS = 252",
               "local DI_FROZENPOS = 253",
               "local DI_SWING = 254",
               "local di_vanespos = ..."
            ],
            "di2modbus": [
               "if (di_vanespos == DI_AUTOPOS) then",
               "    return 5",
               "elseif (di_vanespos == DI_FROZENPOS) then",
               "    return 6",
               "else",
               "    return di_vanespos",
               "end"
            ],
            "di2modbusRetInfo": [
               "Return value to be written to mb_vanes ModBus register",
               "eg: return 3"
            ],
            "modbus2diRegs": [
               {
                  "name": "mb_vanes",
                  "type": "HoldingReg",
                  "address": 21,
```

```
                "description": "Swing/Vane position. 0: Verical; 1: 30°; 2: 45°; 3: 60°; 4: Horizontal; 5: Auto
(swinging); 6: Off"
                }
            ],
            "modbus2diVars": [
                "local DI_ERROR = -1",
                "local DI_AUTOPOS = 252",
                "local DI_FROZENPOS = 253",
                "local DI_SWING = 254",
                "local mb_vanes = ..."
            ],
            "modbus2di": [
                "if (mb_vanes == 5) then",
                "    return DI_AUTOPOS",
                "elseif (mb_vanes == 6) then",
                "    return DI_FROZENPOS",
                "else",
                "    return mb_vanes",
                "end"
            ],
            "modbus2diRetInfo": [
                "Return value to be used in Domintell",
                "eg: return DI_SWING",
                "eg: return 1"
            ]
        }
    }
  }
],
```

- als <type of the IO> = "status", <diRegLua> kunnen:
  - error: Scripts die de fout afhandelen die wordt gemeld door het ModBus-apparaat. Het is gekoppeld aan het register di_error in de Lua-editor in GoldenGate.
  - version: Scripts die de versie afhandelen die wordt geretourneerd door het ModBus-apparaat. Het is gekoppeld aan het register di_version in de Lua-editor in GoldenGate.
  - serial: Scripts die de fout afhandelen die wordt gemeld door het ModBus-apparaat. Het is gekoppeld aan het register di_serial in de Lua-editor in GoldenGate.

- if <type of the IO> = "elec", <diRegLua> kunnen:
  - elecFrequency : Scripts die de gemeten spanning op L1 verwerken. Het is gekoppeld aan di_elecfreq in de Lua-editor in GoldenGate.
  - elecVoltL1: Scripts die de gemeten spanning op L1 verwerken. Het is gekoppeld aan di_elecvolt1 in de Lua-editor in GoldenGate.
  - elecVoltL2: Scripts die de gemeten spanning op L2 verwerken. Het is gekoppeld aan di_elecvolt2 in de Lua-editor in GoldenGate.
  - elecVoltL3: Scripts die de gemeten spanning op L3 verwerken. Het is gekoppeld aan di_elecvolt3 in de Lua-editor in GoldenGate.
  - elecPFL1: Scripts die de gemeten arbeidsfactor op L1 verwerken. Het is gekoppeld aan di_elecpfactor1 in de Lua-editor in GoldenGate.

- elecPFL2: Scripts die de gemeten arbeidsfactor op L2 verwerken. Het is gekoppeld aan di_elecpfactor2 in de Lua-editor in GoldenGate.
- elecPFL3: Scripts die de gemeten arbeidsfactor op L3 verwerken. Het is gekoppeld aan di_elecpfactor3 in de Lua-editor in GoldenGate.
- elecCurrentL1: Scripts die de gemeten stroom op L1 verwerken. Het is gekoppeld aan di_eleccurrent1 in de Lua-editor in GoldenGate.
- elecCurrentL2: Scripts die de gemeten stroom op L2 verwerken. Het is gekoppeld aan di_eleccurrent2 in de Lua-editor in GoldenGate.
- elecCurrentL3: Scripts die de gemeten stroom op L3 verwerken. Het is gekoppeld aan di_eleccurrent3 in de Lua-editor in GoldenGate.
- elecPowerL1: Scripts die het gemeten actieve vermogen op L1 verwerken. Het is gekoppeld aan di_elecpower1 in de Lua-editor in GoldenGate.
- elecPowerL2: Scripts die het gemeten actieve vermogen op L2 verwerken. Het is gekoppeld aan di_elecpower2 in de Lua-editor in GoldenGate.
- elecPowerL3: Scripts die het gemeten actieve vermogen op L3 verwerken. Het is gekoppeld aan di_elecpower3 in de Lua-editor in GoldenGate.
- elecConsumedPower: Scripts die de globale verbruikte stroom afhandelen. Het is gekoppeld aan di_elecconsumedpower in de Lua-editor in GoldenGate.
- elecProducedPower: Scripts die de globale geproduceerde stroom verwerken. Het is gekoppeld aan di_elecproducedpower (moet positief zijn) in de Lua-editor in GoldenGate.
- elecTotalPower: Scripts die het totale gemeten actieve vermogen verwerken (moet gelijk zijn aan elecconsumedpower - elecproducedpower). Het moet negatief zijn als er meer vermogen is geproduceerd dan verbruikt. Het wordt geassocieerd met di_electotalpower in de Lua-editor in GoldenGate.
- elecEnergyL1: Scripts die de geaccumuleerde actieve energie op L1 verwerken. Het is gekoppeld aan di_elecenergy1 in de Lua-editor in GoldenGate.
- elecEnergyL2: Scripts die de geaccumuleerde actieve energie op L2 verwerken. Het is gekoppeld aan di_elecenergy2 in de Lua-editor in GoldenGate.
- elecEnergyL3: Scripts die de geaccumuleerde actieve energie op L3 verwerken. Het is gekoppeld aan di_elecenergy3 in de Lua-editor in GoldenGate.
- elecForwardEnergy: Scripts die de globale geaccumuleerde voorwaartse energie verwerken. Het is gekoppeld aan di_elecfwdenergy in de Lua-editor in GoldenGate.
- elecReverseEnergy: Scripts die de globale geaccumuleerde omgekeerde energie verwerken. Het is gekoppeld aan di_elecrevenergy (moet positief zijn) in de Lua-editor in GoldenGate.
- elecTotalEnergy: Scripts die de totale globale energie afhandelen (moet gelijk zijn aan elecForwardEnergy - elecReverseEnergy). Het moet negatief zijn als er meer energie is geïnjecteerd in het net dan er is verbruikt van het net. Het wordt geassocieerd met di_electotenergy in de Lua editor in GoldenGate.
- elecTotalEnergyTariff1: Scripts die de totale globale energie tijdens tarief 1 periode afhandelen. Het moet negatief zijn als er meer energie is geïnjecteerd in het net dan er is verbruikt van het net gedurende periode 1. Het wordt geassocieerd met di_electotenergyt1 in de Lua-editor in GoldenGate.
- elecTotalEnergyTariff2: Scripts die de totale globale energie tijdens tarief 2-periode afhandelen. Het moet negatief zijn als er meer energie is geïnjecteerd in het net dan er is

verbruikt van het net in periode 2. Het wordt geassocieerd met di_electotenergyt2 in de Lua-editor in GoldenGate.
- elecTotalEnergyTariff3: Scripts die de totale globale energie tijdens tariefperiode 3 verwerken. Het moet negatief zijn als er meer energie is geïnjecteerd in het net dan er is verbruikt van het net in periode 3. Het wordt geassocieerd met di_electotenergyt3 in de Lua-editor in GoldenGate.
- elecTotalEnergyTariff4: Scripts die de totale globale energie tijdens tariefperiode 4 verwerken.Het moet negatief zijn als er meer energie is geïnjecteerd in het net dan er is verbruikt van het net gedurende periode 4.Het wordt geassocieerd met di_electotenergyt4 in de Lua-editor in GoldenGate.
- elecTariffIndic: Scripts die de huidige actieve tariefperiode afhandelen Deze is gekoppeld aan di_electindicator in de Lua-editor in GoldenGate.

◦ als <type of the IO> = "relay", <diRegLua> kunnen:
- relayState *: Scripts die de status van de relaisuitgang afhandelen. Het is gekoppeld aan het register di_relaystate in de Lua-editor in GoldenGate.

◦ als <type of the IO> = "analogIn", <diRegLua> kunnen:
- analog *: Scripts die de waarde van een ruwe analoge ingang verwerken. Het is gekoppeld aan het register di_analog in de Lua-editor in GoldenGate.

◦ als <type of the IO> = "percentIn", <diRegLua> kunnen:
- percent *: Scripts die de waarde van een procent (0-100%) analoge ingang verwerken. Het is gekoppeld aan het register di_percent in de Lua-editor in GoldenGate.

◦ als <type of the IO> = "evCharger", <diRegLua> kunnen:
- *Prochainement...*

◦ als <type of the IO> = "battery", <diRegLua> can be:
- *Prochainement...*

Parameters van Lua scripts

- modbusRegs *: Array van JSON-objecten die de configuratie van ModBus-registers bevatten. Bij gebruik van een DINTMB02 kunnen maximaal vijf 16-bits registers worden toegewezen.
  - modbusRegs.name *: Naam van het register in het Lua script, string. **Moet beginnen met "mb_" met alleen kleine letters, minstens één letter na "mb_" en zonder speciale tekens of spaties**.
  - modbusRegs.type *: Type van het ModBus register, enum string. Mogelijke waarden zijn:
    - HoldingReg: Een holding (16 bits) register. ModBus-functies: 0x03, 0x06 en 0x10.
    - InputReg: Een ingangsregister (16 bits). ModBus-functies: 0x04.
    - CoilReg: Een spoel (1 bit) register. ModBus-functies: 0x01, 0x05 en 0x0F.
    - DiscreteReg: Een discreet (1 bit) register. ModBus-functies: 0x02.
    - ExtensionReg: Virtueel register dat wordt gebruikt om extra ruimte/register aan te geven die

nodig is om meer dan 16-bits waarden op te slaan. Te gebruiken wanneer modbusRegs.nbrItems groter is dan 1.

- modbusRegs.address *: Adres van het ModBus-register (0-gebaseerd), geheel getal (0 - 65535).

- modbusRegs.format: Formaat/toekenning van de ModBus data, enum string. Mogelijke waarden zijn:
  - 1bit: Breedte van de gegevens is 1 bit.
  - S16: De breedte van de gegevens is 16 bits. Standaard.
  - S32-4321: De breedte van de gegevens is 32 bits. Als de data 0x44332211 is, dan wordt 0x4433 opgeslagen in het 16-bits register met het laagste adres en 0x2211 in het 16-bits register met het hoogste adres. Dit is het meest gebruikte 32-bits formaat.
  - S32-2143: Getekend 32-bits geheel getal. Als de data 0x44332211 is, wordt 0x2211 opgeslagen in het 16-bits register met het laagste adres en wordt 0x4433 opgeslagen in het 16-bits register met het hoogste adres.
  - F32-IEEE: 32-bits drijvendkomma getal. Opgeslagen met behulp van IEEE-754 standaard (https://nl.wikipedia.org/wiki/IEEE_754).
  - raw: ruwe gegevens/stringgegevens

- modbusRegs.nbrItems: Aantal ModBus 16-bits registers dat nodig is voor de gegevens. Als modbusRegs.type is ExtensionReg, s dit de offset om ze opnieuw te rangschikken wanneer ze worden samengevoegd.

- modbusRegs.access: Type van de toegang tot het ModBus register, enum string. Mogelijke waarden zijn:
  - rw: ModBus register kan worden gelezen en geschreven. Standaard.
  - ro: ModBus-register kan alleen worden gelezen.
  - wo: ModBus-register kan alleen worden geschreven.

- modbusRegs.description: Beschrijving van het ModBus-register met uitleg over het formaat van de gegevens op basis van het gegevensblad van het ModBus-apparaat.

Voorbeeld van odBus registers voor temp.lua.tempMode

```
"modbusRegs": [
   {
      "name": "mb_op_mode",
      "type": "HoldingReg",
      "address": 3,
      "description": "Mode (0: auto; 1: heating; 2: fan; 3: cooling: 4: dry)"
   },
   {
      "name": "mb_mode_on_off",
      "type": "HoldingReg",
      "address": 5,
```

```
        "description": "On/Off the device (0: off; 1: on)"
    }
]
```

Voorbeeld van ModBus-registers voor elec.lua.elecCurrentL1

```
"modbus2diRegs": [
    {
        "name": "mb_curl1",
        "type": "HoldingReg",
        "address": 313,
        "format": "S32-4321",
        "nbrItems": 2,
        "access": "ro",
        "description": "The current of L1 in 0.001 A unit."
    },
    {
        "name": "mb_curl1_ext1",
        "type": "ExtensionReg",
        "address": 314,
        "format": "S32-4321",
        "nbrItems": 1,
        "access": "ro",
        "description": "The current of L1 in 0.001 A unit."
    }
]
```

- di2modbusConst: Array van strings met declaraties van constanten die in de scripts worden gebruikt. Deze sectie is handig om de gebruiker te laten weten dat hij het script moet herzien als hij het script importeert naar een nieuwere versie van GoldenGate terwijl deze constanten zijn toegevoegd/bijgewerkt. Deze sectie moet de lijst van constanten bevatten die gerelateerd zijn aan het register en de naam van het gerelateerde Domintell-register.

Voorbeeld voor het fan.lua.fanSpeed script

```
"di2modbusConst": [
    "local DI_OFF = 0",
    "local DI_AUTO = 254",
    "local DI_ERROR = -1",
    "local di_fanspeed = ..."
]
```

- di2modbusRetInfo: String/Help over waarden die worden geretourneerd door het script. Deze informatie moet gebaseerd zijn op de datasheet van het ModBus-apparaat.

Voorbeeld

```
"di2modbusRetInfo": [
    "Return value to be written to mb_fanspeed ModBus register",
    "eg: return 2"
]
```

- di2modbus: Array van strings van het Lua script dat de waarde van Domintell converteert naar een waarde die past in de 16-bit ModBus register(s). Elke regel van het script is een item/slot/index van de array.

Voorbeeld

```
"di2modbus": [
    "if (di_fanspeed == DI_AUTO) then",
    "    return 0",
    "else",
    "    return di_fanspeed",
    "end"
]
```

- modbus2diConst: Array van strings met declaraties van constanten die in de scripts worden gebruikt. Deze sectie is handig om de gebruiker te laten weten dat hij het script moet herzien als hij het script importeert naar een nieuwere versie van GoldenGate terwijl deze constanten zijn toegevoegd/bijgewerkt. Deze sectie moet de lijst van constanten bevatten die gerelateerd zijn aan het register en de naam van de ModBus-registers die modbus2diRegs definieert.

- modbus2diRetInfo: String/Help over waarden die worden geretourneerd door het script. Deze informatie wordt gebruikt om de gebruiker te waarschuwen wanneer hij het JSON-bestand importeert in GoldenGate en de geretourneerde argumenten zijn gewijzigd..

- modbus2di: Array van strings van het Lua script dat de waarde van ModBus register(s) zal omzetten naar een waarde die geschikt is om in het Domintell register te passen. Elke regel van het script is een item/slot/index van de array.

Voorbeeld

```
"modbus2di": [
    "if (mb_fanspeed == 0) then",
    "    return DI_AUTO",
    "else",
    "    return mb_fanspeed",
    "end"
]
```

Registers voor simulatie

- simulRegs: Lijst van ModBus registers die moeten worden gebruikt door een ModBus slave simulator, array van JSON objecten. **Niet gebruikt door GoldenGate**.
  - simulRegs[].name *: Naam van het ModBus-register dat moet worden gesimuleerd, string

    Voorbeeld

    ```
    "name": "H0002"
    ```

  - simulRegs[].type * : Type van het ModBus register dat gesimuleerd moet worden, enum string. Mogelijke waarden zijn:
    - HoldingReg: Een bedrijfsregister aangeven.
    - InputReg: Een invoerregister declareren.
    - CoilReg: Een spoelregister declareren.
    - DiscreteReg: Een discreet invoerregister declareren.

Voorbeeld
"type" : "HoldingReg"

- simulRegs[].address *: Adres van het ModBus-register dat moet worden gesimuleerd, geheel getal (1-65535). 1-gebaseerd adres.

Voorbeeld
"address": 2

- simulRegs[].description *: Beschrijving van het ModBus-register dat moet worden gesimuleerd, string.

Voorbeeld
"description": "Setpoint: 10°C to 30°C in heating mode"

- simulRegs[].range: Bereik van geldige waarden die het te simuleren ModBus-register kan accepteren, matrix van gehele getallen.

Voorbeeld
"range": [10, 32]

- simulRegs[].value: Begin-/standaardwaarde van het te simuleren ModBus register, integer.

Voorbeeld
"value": 7

Volledig voorbeeld voor simulRegs

```
"simulRegs": [
   {
      "name": "H0002",
      "type": "HoldingReg",
      "address": 2,
      "description": "Setpoint: 10°C to 30°C in heating mode; 18°C to 32°C in cooling mode (Degrees C unit)",
      "range": [ 10, 32 ],
      "value": 21
   },
   {
      "name": "H0003",
      "type": "HoldingReg",
      "address": 3,
      "description": "FanSpeed (0: auto; 1-5: speed 1 to 5)",
      "range": [ 0, 5 ],
      "value": 0
   }
]
```

Volledig voorbeeld voor de Coolmaster-airconditioner

```
{
   "name": "CoolMasterNet SI mode",
   "type": "Airco",
   "address": 80,
   "description": "CoolMasterNet bridge with L1 in SI master mode and L3 in CG5 slave mode.",
```

DINTMB02
JSON-bestand configuratiegids


```
  "config": {
    "type": "serial",
    "param": {
      "baudrate": 9600,
      "databits": 8,
      "stopbits": 1,
      "parity": "none"
    }
  },
  "simulRegs": [
    {
      "name": "Op. Mode",
      "type": "HoldingReg",
      "address": 17,
      "description": "Operation mode (0: Cool; 1: heat; 2: Auto; 3: Dry; 4: HAUX; 5: Fan; 6: HH; 8: VAM Auto; 9: VAM Bypass; 10: VAM Heat Exc; 11: VAM normal)",
      "range": [ 0, 11 ],
      "value": 2
    },
    {
      "name": "Fan Speed",
      "type": "HoldingReg",
      "address": 18,
      "description": "FanSpeed (0-2: Lo-Me-Hi; 3: auto; 4: top; 5; very-low; 7: VAM Super Hi; 8; VAM low fresh-up; 9: VAM high fresh-up); VAM value does not work in SI mode",
      "range": [ 0, 9 ],
      "value": 3
    },
    {
      "name": "Setpoint",
      "type": "HoldingReg",
      "address": 19,
      "description": "Setpoint expressed in 10th degrees Celcius",
      "range": [ 0, 400 ],
      "value": 210
    },
    {
      "name": "On/off",
      "type": "HoldingReg",
      "address": 20,
      "description": "On/Off the device (0: off; 1: on)",
      "range": [ 0, 1 ],
      "value": 1
    },
    {
      "name": "Filter sign",
      "type": "HoldingReg",
      "address": 21,
      "description": "This register maybe warns when the filter must be clean clean (read) and ii is also used to clean this state (turn off warning light) (write)",
      "range": [ 0, 1 ],
      "value": 1
    },
    {
```

DINTMB02
JSON-bestand configuratiegids


```
            "name": "Vanes",
            "type": "HoldingReg",
            "address": 22,
            "description": "Swing/Vane position. 0: Verical; 1: 30°; 2: 45°; 3: 60°; 4: Horizontal; 5: Auto (swinging); 6: Off",
            "range": [ 0, 6 ],
            "value": 5
        },
        {
            "name": "Room temp",
            "type": "HoldingReg",
            "address": 23,
            "description": "Temperature of the room in 10th degrees Celcius. Some HVAC models allow to set a suggested
temperature room.",
            "range": [ 0, 400 ],
            "value": 210
        },
        {
            "name": "HVAC error code",
            "type": "HoldingReg",
            "address": 24,
            "description": "HVAC error code.",
            "range": [ 0, 65535 ],
            "value": 0
        },
        {
            "name": "Room temp",
            "type": "InputReg",
            "address": 18,
            "description": "Temperature of the room in 10th degrees Celcius.",
            "range": [ 0, 400 ],
            "value": 210
        },
        {
            "name": "HVAC error start of string",
            "type": "InputReg",
            "address": 19,
            "description": "First part of HVAC error string (stored in little endian mode)."
        },
        {
            "name": "HVAC error end of string",
            "type": "InputReg",
            "address": 20,
            "description": "Second part of HVAC error string (stored in little endian mode)."
        }
    ],
    "temp": [
        {
            "param": {
                "hasAccurateMeasTemp": false,
                "minSetpointStep": 1.0,
                "minCoolTemp": 18.0,
                "maxCoolTemp": 32.0,
                "minHeatTemp": 10.0,
                "maxHeatTemp": 30.0,
```

DINTMB02
JSON-bestand configuratiegids


```
            "hasAutoMode": true,
            "hasHeatMode": true,
            "hasCoolMode": true,
            "hasFanMode": true,
            "hasDryMode": true
        },
        "lua": {
            "tempMeas": {
                "modbusRegs": [
                    {
                        "name": "mb_roomtemp",
                        "type": "InputReg",
                        "address": 17,
                        "description": "Temperature of the room in 10th degrees Celcius."
                    }
                ],
                "modbus2diConst": [
                    "local mb_roomtemp = ..."
                ],
                "modbus2di": [
                    "return mb_roomtemp / 10"
                ],
                "modbus2diRetInfo": [
                    "Return temperature in floating decimal format to be used in Domintell",
                    "eg: return 19.6",
                    "eg: return mb_roomtemp / 10"
                ]
            },
            "tempSetpoint": {
                "di2modbusConst": [
                    "local di_setpoint = ..."
                ],
                "di2modbus": [
                    "return (di_setpoint * 10)"
                ],
                "di2modbusRetInfo": [
                    "Return values to be written to <mb_setpoint> ModBus registers",
                    "eg: return 200"
                ],
                "modbusRegs": [
                    {
                        "name": "mb_setpoint",
                        "type": "HoldingReg",
                        "address": 18,
                        "description": "Setpoint expressed in 10th degrees Celcius"
                    }
                ],
                "modbus2diConst": [
                    "local mb_setpoint = ..."
                ],
                "modbus2di": [
                    "return mb_setpoint / 10"
                ],
                "modbus2diRetInfo": [
```

DINTMB02
JSON-bestand configuratiegids


```
                "Return temperature in floating decimal format to be used in Domintell",
                "eg: return 20.0",
                "eg: return mb_setpoint / 10"
            ]
        },
        "tempMode": {
            "di2modbusConst": [
                "local DI_OFF = 0",
                "local DI_HEAT = 1",
                "local DI_COOL = 2",
                "local DI_AUTO = 5",
                "local DI_DRY = 6",
                "local DI_FAN = 7",
                "local DI_ERROR = -1",
                "local di_mode = ..."
            ],
            "di2modbus": [
                "if (di_mode == DI_OFF) then",
                "    return 2,0",
                "elseif (di_mode == DI_HEAT) then",
                "    return 1,1",
                "elseif (di_mode == DI_COOL) then",
                "    return 0,1",
                "elseif (di_mode == DI_AUTO) then",
                "    return 2,1",
                "elseif (di_mode == DI_DRY) then",
                "    return 3,1",
                "elseif (di_mode == DI_FAN) then",
                "    return 5,1",
                "else",
                "    return DI_ERROR, DI_ERROR",
                "end"
            ],
            "di2modbusRetInfo": [
                "Return values to be written to <mb_op_mode> and <mb_mode_on_off> ModBus registers",
                "eg: return 2,0",
                "Attention : value must be returned in the same order than the declaration of the ModBus registers !"
            ],
            "modbusRegs": [
                {
                    "name": "mb_op_mode",
                    "type": "HoldingReg",
                    "address": 16,
                    "description": "Operation mode (0: Cool; 1: heat; 2: Auto; 3: Dry; 4: HAUX; 5: Fan; 6: HH; 8: VAM
Auto; 9: VAM Bypass; 10: VAM Heat Exc; 11: VAM normal)"
                },
                {
                    "name": "mb_mode_on_off",
                    "type": "HoldingReg",
                    "address": 19,
                    "description": "On/Off the device (0: off; 1: on)"
                }
            ],
            "modbus2diConst": [
```

DINTMB02
JSON-bestand configuratiegids

```
                "local DI_OFF = 0",
                "local DI_HEAT = 1",
                "local DI_COOL = 2",
                "local DI_AUTO = 5",
                "local DI_DRY = 6",
                "local DI_FAN = 7",
                "local DI_ERROR = -1",
                "local mb_op_mode, mb_mode_on_off = ..."
            ],
            "modbus2di": [
                "if (mb_mode_on_off == 0) then",
                "    return DI_OFF",
                "elseif (mb_op_mode == 0) then",
                "    return DI_COOL",
                "elseif (mb_op_mode == 1) then",
                "    return DI_HEAT",
                "elseif (mb_op_mode == 2) then",
                "    return DI_AUTO",
                "elseif (mb_op_mode == 3) then",
                "    return DI_DRY",
                "elseif (mb_op_mode == 5) then",
                "    return DI_FAN",
                "else",
                "    return DI_ERROR",
                "end"
            ],
            "modbus2diRetInfo": [
                "Return value to be used in Domintell",
                "eg: return DI_HEAT",
                "DI_xxx variables must be used ! Do not use numeric values !"
            ]
        }
    }
  }
],
"fan": [
  {
    "param": {
      "speedMax": 5,
      "hasOffState": false,
      "hasAutoState": true
    },
    "lua": {
      "fanSpeed": {
        "di2modbusConst": [
          "local DI_OFF = 0",
          "local DI_AUTO = 254",
          "local DI_ERROR = -1",
          "local di_fanspeed = ..."
        ],
        "di2modbus": [
          "if (di_fanspeed == DI_AUTO) then",
          "    return 3",
          "elseif (di_fanspeed >= 6) then",
```

```
                    "    return di_fanspeed + 1",
                    "elseif (di_fanspeed > 3) then",
                    "    return di_fanspeed",
                    "else",
                    "    return di_fanspeed - 1",
                    "end"
                ],
                "di2modbusRetInfo": [
                    "Return value to be written to mb_fan ModBus register",
                    "eg: return 2"
                ],
                "modbusRegs": [
                    {
                        "name": "mb_fan",
                        "type": "HoldingReg",
                        "address": 17,
                        "description": "FanSpeed (0-2: Lo-Me-Hi; 3: auto; 4: top; 5; very-low; 7: VAM Super Hi; 8; VAM low fresh-up; 9: VAM high fresh-up); VAM value does not work in SI mode"
                    }
                ],
                "modbus2diConst": [
                    "local DI_ERROR = -1",
                    "local DI_OFF = 0",
                    "local DI_AUTO = 254",
                    "local mb_fan = ..."
                ],
                "modbus2di": [
                    "if (mb_fan == 3) then",
                    "    return DI_AUTO",
                    "elseif (mb_fan < 3) then",
                    "    return mb_fan + 1",
                    "elseif (mb_fan >= 7) then",
                    "    return mb_fan - 1",
                    "else",
                    "    return mb_fan",
                    "end"
                ],
                "modbus2diRetInfo": [
                    "Return value to be used in Domintell",
                    "eg: return DI_OFF",
                    "eg: return 2"
                ]
            }
        }
    }
],
"vanes": [
    {
        "param": {
            "posMax": 5,
            "hasOffState": true,
            "hasAutoState": true,
            "hasSwingState": false
        },
```

DINTMB02
JSON-bestand configuratiegids


```
        "lua": {
            "vanesPos": {
                "di2modbusConst": [
                    "local DI_ERROR = -1",
                    "local DI_AUTOPOS = 252",
                    "local DI_FROZENPOS = 253",
                    "local DI_SWING = 254",
                    "local di_vanespos = ..."
                ],
                "di2modbus": [
                    "if (di_vanespos == DI_AUTOPOS) then",
                    "    return 5",
                    "elseif (di_vanespos == DI_FROZENPOS) then",
                    "    return 6",
                    "else",
                    "    return di_vanespos",
                    "end"
                ],
                "di2modbusRetInfo": [
                    "Return value to be written to mb_vanes ModBus register",
                    "eg: return 3"
                ],
                "modbusRegs": [
                    {
                        "name": "mb_vanes",
                        "type": "HoldingReg",
                        "address": 21,
                        "description": "Swing/Vane position. 0: Verical; 1: 30°; 2: 45°; 3: 60°; 4: Horizontal; 5: Auto
(swinging); 6: Off"
                    }
                ],
                "modbus2diConst": [
                    "local DI_ERROR = -1",
                    "local DI_AUTOPOS = 252",
                    "local DI_FROZENPOS = 253",
                    "local DI_SWING = 254",
                    "local mb_vanes = ..."
                ],
                "modbus2di": [
                    "if (mb_vanes == 5) then",
                    "    return DI_AUTOPOS",
                    "elseif (mb_vanes == 6) then",
                    "    return DI_FROZENPOS",
                    "else",
                    "    return mb_vanes",
                    "end"
                ],
                "modbus2diRetInfo": [
                    "Return value to be used in Domintell",
                    "eg: return DI_SWING",
                    "eg: return 1"
                ]
            }
        }
```

```
      }
   ],
   "status": [
      {
         "param": {
         },
         "lua": {
            "error": {
               "modbusRegs": [
                  {
                     "name": "mb_error",
                     "type": "HoldingReg",
                     "address": 23,
                     "description": "HVAC error code."
                  }
               ],
               "modbus2diConst": [
                  "local DI_ERROR = -1",
                  "local DI_LVLNORMAL = 0",
                  "local DI_LVLWARNING = 1",
                  "local DI_LVLCRITICAL = 2",
                  "local mb_error = ..."
               ],
               "modbus2di": [
                  "if (mb_error == 0) then",
                  "    return DI_LVLNORMAL, mb_error, \"Pas d'erreur\"",
                  "elseif (mb_error == 666) then",
                  "    return DI_LVLCRITICAL, mb_error, \"Devil is inside\"",
                  "else",
                  "    return DI_LVLWARNING, mb_error, \"Unkonwn error\"",
                  "end"
               ],
               "modbus2diRetInfo": [
                  "Return values to be used in Domintell",
                  "return <error_level>, <error_value>, <error_infostr>",
                  "eg: return DI_LVLCRITICAL, 45, \"Water pump is faulty\"",
                  "eg: return DI_LVLNORMAL, 0, \"\""
               ]
            }
         }
      }
   ]
}
```

Alfabetische index

DINTMB02
JSON-bestand configuratiegids

DINTMB02
JSON-bestand configuratiegids

DINTMB02
JSON-bestand configuratiegids