Gridworks Protocol

Jessica Millar

May 10, 2024

CODE SUPPORT

Protocol used by a GridWorks Spaceheat SCADA

CHAPTER

ONE

INSTALLATION

Note: gridworks requires python 3.10 or higher.

```
(venv)$ pip install gridworks-protocol
```

1.1 GridWorks lexicon

1.1.1 SpaceheatActor

This enum captures the role of the pro-actor for a Spaceheat Node. Spaceheat Nodes with no actor are assigned the enum *NoActor*

Back to Lexicon

1.1.2 ActorClass

The opensource GridWorks Spaceheat Scada software is organized internally as a set of actors, with the prime actor being the SCADA itself. The SCADA must be able to deal with and process a wide variety of heating system arrangements. This is done in a Hardware Layout, which among other things organizes relevant physical devices in the heating system (thermal stores, circulator pumps, the heat pump, resistive heating elements, relays for heating elements). The primary objects in the Hardware Layout are space heat nodes, although the Hardware Layout also organizes and captures information about the specific hardware getting used (i.e. a serial number for a component, as well as its make/model).

All of the actors in the Spaceheat Scada are associated with space heat nodes. The Space heat Node's *ActorClass* is an Actor enum used to determine the code used to run the actor. One node in the hardware layout will have an actor class of *Actor.Scada* (ActorClassEnumSymbol 6d37aa41): this is the SpaceHeat Node running the main SCADA code and supervising/managing all the other Actors.

TODO: discuss pro-actors

Back to Lexicon

1.1.3 Boolean Actuator

ADD

Back to Lexicon

1.1.4 Component

The actual physical devices monitored and actuated by a Spaceheat SCADA require

Back to Lexicon

1.1.5 Data Polling, Capturing and Reporting

Polling: A synchronous action taken by a local SCADA actor to query an external object for a raw physical state reading.

Capturing: The act of retaining a phsyical state read by a local SCADA actor for sending up in a message to its AtomicTNode.

Transmitting: The act of sending data up to the AtomicTNode.

Capturing can also occur both _synchronously_ and _asynchronously_.

Back to Lexicon

1.1.6 Egaugelo

When the component associated to a PowerMeter ShNode has MakeModel EGAUGE__4030, there is a significant amount of configuration required to specify both what is read from the eGauge (input) and what is then sent up to the SCADA (output). This type handles that information.

Back to Lexicon

1.1.7 Hardware Layout

The SCADA must be

able to deal with and process a wide variety of heating system arrangements. This is done in a Hardware Layout, which among other things organizes relevant physical devices in the heating system (thermal stores, circulator pumps, the heat pump, resistive heating elements, relays for heating elements). The primary objects in the Hardware Layout are space heat nodes, although the Hardware Layout also organizes and captures information about the specific hardware getting used (i.e. a serial number for a component, as well as its make/model).

Back to Lexicon

1.1.8 GridWorks TankModule1

TODO: explain what the GridWorks TankModule is, how we've built it using Fibaro SmartImplents, how we talk to it via a hubitat hub, how to do it yourself, and how the software drivers work.

hubitat.tank.settings.gt

1.1.9 IoT Hubs

Hubitat

TODO: How to use SCADA code with a Hubitat IoT hub.

1.1.10 Make Model

Determines Make/Model of device associated to a Spaceheat Node supervised by SCADA

Used to assign the correct driver code to various sensors. For example, if we are using an Adafruit 1-wire temp sensor identified by Adafruit as Product Id 642, the associated SimpleSensor actor in the SCADA code knows that the HardwareLayout will include the unique identifier for that 1-wire, and will know to use the appropriate driver code in the repository.

Back to Lexicon

1.1.11 Multipurpose Sensor

A sensor that either reads multiple kinds of readings from the same sensing device (for example reads current and voltage), reads multiple different objects (temperature from two different thermisters) or both.

Back to Lexicon

1.1.12 SpaceheatNode

A SpaceheatNode, or ShNode, ...

The Spaceheat SCADA maintains a set of SpaceHeat Nodes. These nodes are categorized using two different enums: SpaceheatActor, and SpaceheatRole.

Back to Lexicon

1.1.13 Relay State

A Relay State of *0* indicates the relay is OPEN (off). A Relay State of *1* indicates the relay is CLOSED (on). Note that *0* means the relay is open whether or not the relay is normally open or normally closed (For a normally open relay, the relay is ENERGIZED when it is in state *0* and DE-ENERGIZED when it is in state *1*.)

REPORTING THE RELAY STATE.

Boolean actuator actors report when they send an actuation command to its driver so that the SCADA can add this to information to be sent up to the AtomicTNode.

Boolean actuator actors read the state of their relay, when possible.

Note that a reading of the state of the actuator may not mean the relay is in the reported position. For example, the NCD relay requires two power sources - one from the Pi and one a lowish DC voltage from another plug (12 or 24V). If the second power source is off, the relay will still report being on when it is actually off.

Note also that the thing getting actuated (for example the boost element in the water tank) may not be getting any power because of another relay in series. For example, we can throw a large 240V breaker in the test garage and the NCD relay will actuate without the boost element turning on. Or the element could be burned out.

So measuring the current and/or power of the thing getting actuated is really the best test.

Back to Lexicon

1.1.14 Spaceheat Node Role

Categorizes SpaceheatNodes by their function within the heating system

Back to Lexicon

1.1.15 Spaceheat SCADA

A Spaceheat SCADA is a SCADA whose TerminalAsset represents a thermal storage heating system for heating a building and/or a room.

Scada as Spaceheat actor

The opensource GridWorks Spaceheat Scada software is organized internally as a set of actors, with the prime actor being the SCADA itself. The SCADA must be able to deal with and process a wide variety of heating system arrangements. This is done in a Hardware Layout, which among other things organizes relevant physical devices in the heating system (thermal stores, circulator pumps, the heat pump, resistive heating elements, relays for heating elements). The primary objects in the Hardware Layout are space heat nodes, although the Hardware Layout also organizes and captures information about the specific hardware getting used (i.e. a serial number for a component, as well as its make/model).

All of the actors in the Spaceheat Scada are associated with space heat nodes. The Space heat Node's *ActorClass* is an Actor enum used to determine the code used to run the actor. One node in the hardware layout will have an actor class of *Actor.Scada* (ActorClassEnumSymbol 6d37aa41): this is the SpaceHeat Node running the main SCADA code and supervising/managing all the other Actors.

TODO: discuss pro-actors

Back to Lexicon

1.1.16 Simple Sensor

A SimpleSensor is an ActorClass for a SpaceheatNode. An ShNode is a SimpleSensor if it is both the device taking a reading and the thing getting read. There can also only be one type of reading.

Here is an example.

A 1-wire temp sensor read by a SCADA has only one type of reading: temperature in degrees C times 1000. It also only reads one thing: say, water temp at the top of the tank where the sensor is located.

A SpaceHeat node for *a.tank.temp1* conflates the 1-wire reading that temperature with the location and temperature itself. this is a SimpelSensor

Back to Lexicon

1.1.17 TelemetryName

TelemetryName is a foundational Enum used by SpaceHeat SCADAs …

Back to Lexicon

1.2 Enum Application Shared Language (ASL) Specifications

1.2.1 spaceheat.make.model.001


```
{
  "gtr_asl": "001",
  "enum_name": "spaceheat.make.model",
  "enum_version": "001",
  "description": "Determines Make/Model of device associated to a Spaceheat Node␣
↪supervised by SCADA",
  "url": "https://gridworks-protocol.readthedocs.io/en/latest/make-model.html",
  "ssot": "https://gridworks-type-registry.readthedocs.io/en/latest/enums.html
↪#spaceheatmakemodel",
  "values": [
    "UnknownMake__UnknownModel",
    "Egauge__4030",
    "NCD__PR8-14-SPST",
    "Adafruit__642",
    "GridWorks__TSnap1",
    "GridWorks__WaterTempHighPrecision",
    "Gridworks__SimPm1",
    "SchneiderElectric__Iem3455",
    "GridWorks__SimBool30AmpRelay",
    "OpenEnergy__EmonPi",
    "GridWorks__SimTSnap1",
    "Atlas__EzFlo",
    "Hubitat__C7__LAN1",
    "GridWorks__Tank_Module_1",
    "Fibaro__Analog_Temp_Sensor",
    "Amphenol__NTC_10K_Thermistor_MA100GG103BN",
    "YHDC__SCT013-100",
    "Magnelab__SCT-0300-050",
    "GridWorks__MultiTemp1",
    "Krida__Emr16-I2c-V3"
  ],
  "value_to_gt_symbol": {
    "UNKNOWNMAKE__UNKNOWNMODEL": "00000000",
    "EGAUGE__4030": "beb6d3fb",
    "NCD__PR814SPST": "fabfa505",
    "ADAFRUIT__642": "acd93fb3",
    "GRIDWORKS__TSNAP1": "d0178dc3",
    "GRIDWORKS__WATERTEMPHIGHPRECISION": "f8b497e8",
    "GRIDWORKS__SIMPM1": "076da322",
    "SCHNEIDERELECTRIC__IEM3455": "d300635e",
    "GRIDWORKS__SIMBOOL30AMPRELAY": "e81d74a8",
```


(continues on next page)

(continued from previous page)


```
    "OPENENERGY__EMONPI": "c75d269f",
    "GRIDWORKS__SIMTSNAP1": "3042c432",
    "ATLAS__EZFLO": "d0b0e375",
    "HUBITAT__C7__LAN1": "4d649420",
    "GRIDWORKS__TANK_MODULE_1": "bd759051",
    "FIBARO__ANALOG_TEMP_SENSOR": "1f19839d",
    "AMPHENOL__NTC_10K_THERMISTOR_MA100GG103BN": "46f21cd5",
    "YHDC__SCT013100": "08da3f7d",
    "MAGNELAB__SCT0300050": "a8d9a70d",
    "GRIDWORKS__MULTITEMP1": "bb31d136",
    "KRIDA__EMR16I2CV3": "3353ce46"
  },
  "value_to_version": {
    "UNKNOWNMAKE__UNKNOWNMODEL": "000",
    "EGAUGE__4030": "000",
    "NCD__PR814SPST": "000",
    "ADAFRUIT__642": "000",
    "GRIDWORKS__TSNAP1": "000",
    "GRIDWORKS__WATERTEMPHIGHPRECISION": "000",
    "GRIDWORKS__SIMPM1": "000",
    "SCHNEIDERELECTRIC__IEM3455": "000",
    "GRIDWORKS__SIMBOOL30AMPRELAY": "000",
    "OPENENERGY__EMONPI": "000",
    "GRIDWORKS__SIMTSNAP1": "000",
    "ATLAS__EZFLO": "000",
    "HUBITAT__C7__LAN1": "001",
    "GRIDWORKS__TANK_MODULE_1": "001",
    "FIBARO__ANALOG_TEMP_SENSOR": "001",
    "AMPHENOL__NTC_10K_THERMISTOR_MA100GG103BN": "001",
    "YHDC__SCT013100": "001",
    "MAGNELAB__SCT0300050": "001",
    "GRIDWORKS__MULTITEMP1": "001",
    "KRIDA__EMR16I2CV3": "001"
  },
  "value_descriptions": {
    "UnknownMake__UnknownModel": "",
    "Egauge__4030": "A power meter in Egauge's 403x line. More Info: https://drive.
↪google.com/drive/u/0/folders/1abJ-o9tlTscsQpMvT6SHxIm5j5aODgfA",
    "NCD__PR8-14-SPST": "NCD's 4-channel high-power relay controller + 4 GPIO with I2C␣
↪interface. More Info: https://store.ncd.io/product/4-channel-high-power-relay-
↪controller-4-gpio-with-i2c-interface/?attribute_pa_choose-a-relay=20-amp-spdt",
    "Adafruit__642": "Adafruit's high-temp, water-proof 1-wire temp sensor. More Info:␣
↪https://www.adafruit.com/product/642",
    "GridWorks__TSnap1": "Actual GridWorks TSnap 1.0 SCADA Box.",
    "GridWorks__WaterTempHighPrecision": "Simulated temp sensor.",
    "Gridworks__SimPm1": "Simulated power meter.",
    "SchneiderElectric__Iem3455": "Schneider Electric IEM 344 utility meter.",
    "GridWorks__SimBool30AmpRelay": "Simulated relay.",
    "OpenEnergy__EmonPi": "Open Energy's open source multipurpose sensing device␣
↪(including internal power meter). More Info: https://docs.openenergymonitor.org/emonpi/
↪technical.html",
    "GridWorks__SimTSnap1": "Simulated SCADA Box.",
```


(continues on next page)

(continued from previous page)

```
    "Atlas__EzFlo": "Atlas Scientific EZO Embedded Flow Meter Totalizer, pulse to I2C.␣
↪More Info: https://drive.google.com/drive/u/0/folders/142bBV1pQIbMpyIR_0iRUr5gnzWgknOJp
↪",
    "Hubitat__C7__LAN1": "This refers to a Hubitat C7 that has been configured in a␣
↪specific way with respect to the APIs it presents on the Local Area Network. The␣
↪Hubitat C7 is a home automation hub that supports building ZigBee and ZWave meshes,␣
↪plugs into Ethernet, has a reasonable user interface and has an active community of␣
↪open-source developers who create drivers and package managers for devices, and␣
↪supports the creation of various types of APIs on the Local Area Network. More Info:␣
↪https://drive.google.com/drive/folders/1AqAU_lC2phzuI9XRYvogiIYA7GXNtlr6",
    "GridWorks__Tank_Module_1": "This refers to a small module designed and assembled by␣
↪GridWorks that is meant to be mounted to the side of a hot water tank. It requires 24V␣
↪DC and has 4 temperature sensors coming out of it labeled 1, 2, 3 and 4. It is meant␣
↪to provide temperature readings (taken within a half a second of each other) of all 4␣
↪of its sensors once a minute. More Info: https://drive.google.com/drive/folders/
↪1GSxDd8Naf1GKK_fSOgQU933M1UcJ4r8q",
    "Fibaro__Analog_Temp_Sensor": "This enum refers to a Fibaro FGBS-222 home automation␣
↪device that has been configured in a specific way. This includes (1) being attached to␣
↪two 10K NTC thermistors and a specific voltage divider circuit that specifies its␣
↪temperature as a function of voltage and (2) one of its potential free outputs being␣
↪in-line with the power of a partner Fibaro, so that it can power cycle its partner␣
↪(because there are reports of Fibaros no longer reporting temp change after weeks or␣
↪months until power cylced). The Fibaro itself is a tiny (29 X 18 X 13 mm) Z-Wave␣
↪device powered on 9-30V DC that can read up to 6 1-wire DS18B20 temp sensors, 2 0-10V␣
↪analog inputs and also has 2 potential free outputs. More Info: https://drive.google.
↪com/drive/u/0/folders/1Muhsvw00goppHIfGSEmreX4hM6V78b-m",
    "Amphenol__NTC_10K_Thermistor_MA100GG103BN": "A small gauge, low-cost, rapid␣
↪response NTC 10K Thermistor designed for medical applications. More Info: https://
↪drive.google.com/drive/u/0/folders/11HW4ov66UvxKAwqApW6IrtoXatZBLQkd",
    "YHDC__SCT013-100": "YHDC current transformer More Info: https://en.yhdc.com/product/
↪SCT013-401.html",
    "Magnelab__SCT-0300-050": "Magnelab 50A current transformer",
    "GridWorks__MultiTemp1": "GridWorks Analog temperature sensor that has 12 channels␣
↪(labeled 1-12) to read 12 10K NTC Thermistors. It is comprised of 3 NCD ADS 1115 I2C␣
↪temperature sensors with I2C Addresses 0x4b, 0x48, 0x49. More Info: https://drive.
↪google.com/drive/u/0/folders/1OuY0tunaad2Ie4Id3zFB7FcbEwHizWuL",
    "Krida__Emr16-I2c-V3": "16-Channel I2C Low Voltage Electromagnetic Relay Board More␣
↪Info: https://drive.google.com/drive/u/0/folders/1jL82MTRKEh9DDmxJFQ2yU2cjqnVD9Ik7"
  },
  "default_value": "UnknownMake__UnknownModel"
}
```

1.2.2 local.comm.interface.000


```
{
  "gtr_asl": "001",
  "enum_name": "local.comm.interface",
  "enum_version": "000",
  "description": "Categorization of in-house comm mechanisms for SCADA",
  "ssot": "https://gridworks-type-registry.readthedocs.io/en/latest/enums.html
↪#localcominterface",
  "values": [
    "Unknown",
    "I2C",
    "Ethernet",
    "OneWire",
    "RS485",
    "SimRabbit",
    "Wifi",
    "Analog_4_20_mA",
    "RS232"
  ],
  "value_to_gt_symbol": {
    "UNKNOWN": "00000000",
    "I2C": "9ec8bc49",
    "ETHERNET": "c1e7a955",
    "ONEWIRE": "ae2d4cd8",
    "RS485": "a6a4ac9f",
    "SIMRABBIT": "efc144cd",
    "WIFI": "46ac6589",
    "ANALOG_4_20_MA": "653c73b8",
    "RS232": "0843a726"
  },
  "value_to_version": {
    "UNKNOWN": "000",
    "I2C": "000",
    "ETHERNET": "000",
    "ONEWIRE": "000",
    "RS485": "000",
    "SIMRABBIT": "000",
    "WIFI": "000",
    "ANALOG_4_20_MA": "000",
    "RS232": "000"
  },
  "value_descriptions": {
    "Unknown": "",
    "I2C": "",
    "Ethernet": "",
    "OneWire": "",
    "RS485": "",
    "SimRabbit": "",
    "Wifi": "",
    "Analog_4_20_mA": "",
    "RS232": ""
  },
```


(continues on next page)

(continued from previous page)


```
  "default_value": "Unknown"
}
```


1.2.3 sh.node.role.000


```
{
  "gtr_asl": "001",
  "enum_name": "sh.node.role",
  "enum_version": "000",
  "description": "Categorizes SpaceheatNodes by their function within the heating system
↪",
  "url": "https://gridworks-protocol.readthedocs.io/en/latest/spaceheat-node-role.html",
  "ssot": "https://gridworks-type-registry.readthedocs.io/en/latest/enums.html#shnoderole
↪",
  "values": [
    "Unknown",
    "Scada",
    "HomeAlone",
    "Atn",
    "PowerMeter",
    "BoostElement",
    "BooleanActuator",
    "DedicatedThermalStore",
    "TankWaterTempSensor",
    "PipeTempSensor",
    "RoomTempSensor",
    "OutdoorTempSensor",
    "PipeFlowMeter",
    "HeatedSpace",
    "HydronicPipe",
    "BaseboardRadiator",
    "RadiatorFan",
    "CirculatorPump",
    "MultiChannelAnalogTempSensor",
    "Outdoors"
  ],
  "value_to_gt_symbol": {
    "Unknown": "00000000",
    "Scada": "d0afb424",
    "HomeAlone": "863e50d1",
    "Atn": "6ddff83b",
    "PowerMeter": "9ac68b6e",
    "BoostElement": "99c5f326",
    "BooleanActuator": "57b788ee",
    "DedicatedThermalStore": "3ecfe9b8",
    "TankWaterTempSensor": "73308a1f",
    "PipeTempSensor": "c480f612",
    "RoomTempSensor": "fec74958",
    "OutdoorTempSensor": "5938bf1f",
    "PipeFlowMeter": "ece3b600",
```


(continues on next page)

(continued from previous page)

```
    "HeatedSpace": "65725f44",
    "HydronicPipe": "fe3cbdd5",
    "BaseboardRadiator": "05fdd645",
    "RadiatorFan": "6896109b",
    "CirculatorPump": "b0eaf2ba",
    "MultiChannelAnalogTempSensor": "661d7e73",
    "Outdoors": "dd975b31"
  },
  "value_to_version": {
    "Unknown": "000",
    "Scada": "000",
    "HomeAlone": "000",
    "Atn": "000",
    "PowerMeter": "000",
    "BoostElement": "000",
    "BooleanActuator": "000",
    "DedicatedThermalStore": "000",
    "TankWaterTempSensor": "000",
    "PipeTempSensor": "000",
    "RoomTempSensor": "000",
    "OutdoorTempSensor": "000",
    "PipeFlowMeter": "000",
    "HeatedSpace": "000",
    "HydronicPipe": "000",
    "BaseboardRadiator": "000",
    "RadiatorFan": "000",
    "CirculatorPump": "000",
    "MultiChannelAnalogTempSensor": "000",
    "Outdoors": "000"
  },
  "value_descriptions": {
    "Unknown": "Unknown Role",
    "Scada": "Primary SCADA",
    "HomeAlone": "HomeAlone GNode",
    "Atn": "AtomicTNode",
    "PowerMeter": "A SpaceheatNode representing the power meter that is used to settle␣
↪financial transactions with the TerminalAsset. That is, this is the power meter whose␣
↪accuracy is certified in the creation of the TerminalAsset GNode via creation of the␣
↪TaDeed. More Info: https://gridworks.readthedocs.io/en/latest/terminal-asset.html",
    "BoostElement": "Resistive element used for providing heat to a thermal store.",
    "BooleanActuator": "A solid state or mechanical relay with two states (open, closed)
↪",
    "DedicatedThermalStore": "A dedicated thermal store within a thermal storage heating␣
↪system - could be one or more water tanks, phase change material, etc.",
    "TankWaterTempSensor": "A temperature sensor used for measuring temperature inside␣
↪or on the immediate outside of a water tank.",
    "PipeTempSensor": "A temperature sensor used for measuring the temperature of a tank.
↪ Typically curved metal thermistor with thermal grease for good contact.",
    "RoomTempSensor": "A temperature sensor used for measuring room temperature, or temp␣
↪in a heated space more generally.",
    "OutdoorTempSensor": "A temperature sensor used for measuring outdoor temperature.",
    "PipeFlowMeter": "A meter that measures flow of liquid through a pipe, in units of␣
```

(continues on next page)

(continued from previous page)

```
↪VOLUME/TIME",
    "HeatedSpace": "A Heated Space.",
    "HydronicPipe": "A pipe carrying techinical water or other fluid (e.g. glycol) in a␣
↪heating system.",
    "BaseboardRadiator": "A baseboard radiator - one kind of emitter in a hydronic␣
↪heating system.",
    "RadiatorFan": "A fan that can amplify the power out of a radiator.",
    "CirculatorPump": "Circulator pump for one or more of the hydronic pipe loops",
    "MultiChannelAnalogTempSensor": "An analog multi channel temperature sensor",
    "Outdoors": "The outdoors"
  },
  "default_value": "Unknown"
}
```

1.2.4 spaceheat.unit.000

```
{
  "gtr_asl": "001",
  "enum_name": "spaceheat.unit",
  "enum_version": "000",
  "description": "Specifies the physical unit of sensed data reported by SCADA",
  "ssot": "https://gridworks-type-registry.readthedocs.io/en/latest/enums.html
↪#spaceheatunit",
  "values": [
    "Unknown",
    "Unitless",
    "W",
    "Celcius",
    "Fahrenheit",
    "Gpm",
    "WattHours",
    "AmpsRms",
    "VoltsRms",
    "Gallons"
  ],
  "value_to_gt_symbol": {
    "Unknown": "00000000",
    "Unitless": "ec972387",
    "W": "f459a9c3",
    "Celcius": "ec14bd47",
    "Fahrenheit": "7d8832f8",
    "Gpm": "b4580361",
    "WattHours": "d66f1622",
    "AmpsRms": "a969ac7c",
    "VoltsRms": "e5d7555c",
    "Gallons": "8e123a26"
  },
  "value_to_version": {
    "Unknown": "000",
    "Unitless": "000",
```

(continues on next page)

(continued from previous page)

```
    "W": "000",
    "Celcius": "000",
    "Fahrenheit": "000",
    "Gpm": "000",
    "WattHours": "000",
    "AmpsRms": "000",
    "VoltsRms": "000",
    "Gallons": "000"
  },
  "value_descriptions": {
    "Unknown": "",
    "Unitless": "",
    "W": "",
    "Celcius": "",
    "Fahrenheit": "",
    "Gpm": "",
    "WattHours": "",
    "AmpsRms": "",
    "VoltsRms": "",
    "Gallons": ""
  },
  "default_value": "Unknown"
}
```

1.2.5 sh.actor.class.001

```
{
  "gtr_asl": "001",
  "enum_name": "sh.actor.class",
  "enum_version": "001",
  "description": "Determines the code running Spaceheat Nodes supervised by Spaceheat␣
↪SCADA software",
  "url": "https://gridworks-protocol.readthedocs.io/en/latest/actor-class.html",
  "ssot": "https://gridworks-type-registry.readthedocs.io/en/latest/enums.html
↪#shactorclass",
  "values": [
    "NoActor",
    "Scada",
    "HomeAlone",
    "BooleanActuator",
    "PowerMeter",
    "Atn",
    "SimpleSensor",
    "MultipurposeSensor",
    "Thermostat",
    "HubitatTelemetryReader",
    "HubitatTankModule",
    "HubitatPoller"
  ],
  "value_to_gt_symbol": {
```

(continues on next page)

(continued from previous page)


```
    "NoActor": "00000000",
    "Scada": "6d37aa41",
    "HomeAlone": "32d3d19f",
    "BooleanActuator": "fddd0064",
    "PowerMeter": "2ea112b9",
    "Atn": "b103058f",
    "SimpleSensor": "dae4b2f0",
    "MultipurposeSensor": "7c483ad0",
    "Thermostat": "4a9c1785",
    "HubitatTelemetryReader": "0401b27e",
    "HubitatTankModule": "e2877329",
    "HubitatPoller": "00000100"
  },
  "value_to_version": {
    "NoActor": "000",
    "Scada": "000",
    "HomeAlone": "000",
    "BooleanActuator": "000",
    "PowerMeter": "000",
    "Atn": "000",
    "SimpleSensor": "000",
    "MultipurposeSensor": "000",
    "Thermostat": "000",
    "HubitatTelemetryReader": "001",
    "HubitatTankModule": "001",
    "HubitatPoller": "001"
  },
  "value_descriptions": {
    "NoActor": "A SpaceheatNode that does not have any code running on its behalf within␣
↪the SCADA, but is instead only a reference object (for example, a tank of hot water or␣
↪a resistive element) that can be discussed (for example, the power drawn by the␣
↪resistive element can be measured) or evaluated (for example, a set of 5 different␣
↪temperatures in different places on the tank can be used to estimate total thermal␣
↪energy in the tank).",
    "Scada": "The SCADA actor is the prime piece of code running and supervising other␣
↪ProActors within the SCADA code. It is also responsible for managing the state of␣
↪TalkingWith the AtomicTNode, as well maintaining and reporting a boolean state␣
↪variable that indicates whether it is following dispatch commands from the AtomicTNode␣
↪XOR following dispatch commands from its own HomeAlone actor.",
    "HomeAlone": "HomeAlone is an abstract Spaceheat Actor responsible for dispatching␣
↪the SCADA when it is not talking with the AtomicTNode.",
    "BooleanActuator": "A SpaceheatNode representing a generic boolean actuator capable␣
↪of turning on (closing a circuit) or turning off (opening a circuit).",
    "PowerMeter": "A SpaceheatNode representing the power meter that is used to settle␣
↪financial transactions with the TerminalAsset. That is, this is the power meter whose␣
↪accuracy is certified in the creation of the TerminalAsset GNode via creation of the␣
↪TaDeed. More Info: https://gridworks.readthedocs.io/en/latest/terminal-asset.html",
    "Atn": "A SpaceheatNode representing the AtomicTNode. Note that the code running the␣
↪AtomicTNode is not local within the SCADA code, except for a stub used for testing␣
↪purposes. More Info: https://gridworks.readthedocs.io/en/latest/atomic-t-node.html",
    "SimpleSensor": "A SpaceheatNode representing a sensor that measures a single␣
↪category of quantity (for example, temperature) for a single object (for example, on a␣
```


(continues on next page)

(continued from previous page)

```
↪pipe). More Info: https://gridworks-protocol.readthedocs.io/en/latest/simple-sensor.
↪html",
    "MultipurposeSensor": "A sensor that either reads multiple kinds of readings from␣
↪the same sensing device (for example reads current and voltage), reads multiple␣
↪different objects (temperature from two different thermisters) or both. More Info:␣
↪https://gridworks-protocol.readthedocs.io/en/latest/multipurpose-sensor.html",
    "Thermostat": "A SpaceheatNode representing a thermostat.",
    "HubitatTelemetryReader": "A generic actor for reading telemetry data from a Hubitat␣
↪Home Automation Hub LAN API. More Info: https://drive.google.com/drive/u/0/folders/
↪1AqAU_lC2phzuI9XRYvogiIYA7GXNtlr6",
    "HubitatTankModule": "The actor for running a GridWorks TankModule, comprised of two␣
↪Z-Wave Fibaro temp sensors built together inside a small container that has 4␣
↪thermistors attached. These are designed to be installed from top (1) to bottom (4) on␣
↪a stratified thermal storage tank. More Info: https://drive.google.com/drive/u/0/
↪folders/1GSxDd8Naf1GKK_fSOgQU933M1UcJ4r8q",
    "HubitatPoller": "An actor for representing a somewhat generic ShNode (like a␣
↪thermostat) that can be polled through the Hubitat."
  },
  "default_value": "NoActor"
}
```

1.2.6 spaceheat.telemetry.name.001

```
{
  "gtr_asl": "001",
  "enum_name": "spaceheat.telemetry.name",
  "enum_version": "001",
  "description": "Specifies the name of sensed data reported by a Spaceheat SCADA",
  "url": "https://gridworks-protocol.readthedocs.io/en/latest/telemetry-name.html",
  "ssot": "https://gridworks-type-registry.readthedocs.io/en/latest/enums.html
↪#spaceheattelemetryname",
  "values": [
    "Unknown",
    "PowerW",
    "RelayState",
    "WaterTempCTimes1000",
    "WaterTempFTimes1000",
    "GpmTimes100",
    "CurrentRmsMicroAmps",
    "GallonsTimes100",
    "VoltageRmsMilliVolts",
    "MilliWattHours",
    "FrequencyMicroHz",
    "AirTempCTimes1000",
    "AirTempFTimes1000"
  ],
  "value_to_gt_symbol": {
    "Unknown": "00000000",
    "PowerW": "af39eec9",
    "RelayState": "5a71d4b3",
```

(continues on next page)

(continued from previous page)


```
    "WaterTempCTimes1000": "c89d0ba1",
    "WaterTempFTimes1000": "793505aa",
    "GpmTimes100": "d70cce28",
    "CurrentRmsMicroAmps": "ad19e79c",
    "GallonsTimes100": "329a68c0",
    "VoltageRmsMilliVolts": "bb6fdd59",
    "MilliWattHours": "e0bb014b",
    "FrequencyMicroHz": "337b8659",
    "AirTempCTimes1000": "0f627faa",
    "AirTempFTimes1000": "4c3f8c78"
  },
  "value_to_version": {
    "Unknown": "000",
    "PowerW": "000",
    "RelayState": "000",
    "WaterTempCTimes1000": "000",
    "WaterTempFTimes1000": "000",
    "GpmTimes100": "000",
    "CurrentRmsMicroAmps": "000",
    "GallonsTimes100": "000",
    "VoltageRmsMilliVolts": "001",
    "MilliWattHours": "001",
    "FrequencyMicroHz": "001",
    "AirTempCTimes1000": "001",
    "AirTempFTimes1000": "001"
  },
  "value_descriptions": {
    "Unknown": "Default Value - unknown telemetry name.",
    "PowerW": "Power in Watts.",
    "RelayState": "An associated read must be either 0 or 1, with 0 meaning that the␣
↪relay is open and current CANNOT flow and 1 meaning that the relay is closed and␣
↪current CAN flow. Note in particular that this TelemetryName is NOT meant to be used␣
↪to reflect whether a relay is energized or de-energized and in particular '1' means␣
↪the same thing for both Normally Open and Normally Closed relays. Also, it is not␣
↪meant to be used for a double-throw relay.",
    "WaterTempCTimes1000": "Water temperature, in Degrees Celcius multiplied by 1000.␣
↪Example: 43200 means 43.2 deg Celcius.",
    "WaterTempFTimes1000": "Water temperature, in Degrees F multiplied by 1000. Example:␣
↪142100 means 142.1 deg Fahrenheit.",
    "GpmTimes100": "Gallons Per Minute multiplied by 100. Example: 433 means 4.33␣
↪gallons per minute.",
    "CurrentRmsMicroAmps": "Current measurement in Root Mean Square MicroAmps.",
    "GallonsTimes100": "Gallons multipled by 100. This is useful for flow meters that␣
↪report cumulative gallons as their raw output. Example: 55300 means 55.3 gallons.",
    "VoltageRmsMilliVolts": "Voltage in Root Mean Square MilliVolts.",
    "MilliWattHours": "Energy in MilliWattHours.",
    "FrequencyMicroHz": "Frequency in MicroHz. Example: 59,965,332 means 59.965332 Hz.",
    "AirTempCTimes1000": "Air temperature, in Degrees Celsius multiplied by 1000.␣
↪Example: 6234 means 6.234 deg Celcius.",
    "AirTempFTimes1000": "Air temperature, in Degrees F multiplied by 1000. Example:␣
↪69329 means 69.329 deg Fahrenheit."
  },
```


(continues on next page)

(continued from previous page)

```
  "default_value": "Unknown"
}
```

1.3 Type Application Shared Language (ASL) Specifications

1.3.1 component.attribute.class.gt.000

```
{
  "gtr_asl": "001",
  "type_name": "component.attribute.class.gt",
  "version": "000",
  "owner": "gridworks@gridworks-consulting.com",
  "description": "Component Attribute Class Gt. Authority for the attributes of the␣
↪component.attribute.class.gt.000 belongs to the WorldRegistry. The WorldRegistry is␣
↪part of the GridWorks 'BackOffice' structure for managing relational device data.␣
↪Generally speaking, a component attribute class is meant to specify WHAT you might␣
↪order from a plumbing supply store to 'get the same part.' The Component refers to␣
↪something that will have a specific serial number.",
  "url": "https://g-node-registry.readthedocs.io/en/latest/component-attribute-class.html
↪",
  "properties": {
    "ComponentAttributeClassId": {
      "type": "string",
      "format": "UuidCanonicalTextual",
      "title": "ComponentAttributeClassId",
      "description": "Unique identifier for the device class (aka 'cac' or Component␣
↪Attribute Class). This identifier is used to associate a make/model with a specific␣
↪component (i.e. the component will point to its ComponentAttributeClassId).",
      "required": true
    },
    "DisplayName": {
      "type": "string",
      "description": "Optional Mutable field to include manufacturer's model name. Note␣
↪that several different models may be given the same spaceheat.make.model enum name.",
      "required": false
    },
    "TypeName": {
      "type": "string",
      "value": "component.attribute.class.gt",
      "title": "The type name"
    },
    "Version": {
      "type": "string",
      "title": "The type version",
      "default": "000",
      "required": true
    }
  },
  "example": {
```

(continues on next page)

(continued from previous page)

```
    "ComponentAttributeClassId": "683c193a-bf83-4491-a294-c0e32865a407",
    "DisplayName": "Axeman Pressurized 150 Gallon Water Tank",
    "TypeName": "component.attribute.class.gt",
    "Version": "000"
  },
  "formats": {
    "UuidCanonicalTextual": {
      "type": "string",
      "description": "A string of hex words separated by hyphens of length 8-4-4-4-12.",
      "example": "652ba6b0-c3bf-4f06-8a80-6b9832d60a25"
    }
  }
}
```

1.3.2 component.gt.000

```
{
  "gtr_asl": "001",
  "type_name": "component.gt",
  "version": "000",
  "owner": "gridworks@gridworks-consulting.com",
  "description": "Component Gt. Authority for the attributes of the component.gt.000␣
↪(ComponentId, ComponentAttributeClassId, DisplayName, HwUid) belongs to the␣
↪WorldRegistry. The WorldRegistry is part of the GridWorks 'BackOffice' structure for␣
↪managing relational device data . Notably, ComponentId and ComponentAttributeClass are␣
↪both required and immutable. HwUid is optional but once it is set to a non-null value␣
↪that is also immutable - it is meant to be an immutable identifier associated to a␣
↪specific physical device, ideally one that can be read remotely by the SCADA and also␣
↪by the naked eye. The DisplayName is mutable, with its current value in time governed␣
↪by the WorldRegistry.",
  "url": "https://g-node-registry.readthedocs.io/en/latest/component.html",
  "properties": {
    "ComponentId": {
      "type": "string",
      "format": "UuidCanonicalTextual",
      "title": "Component Id",
      "description": "Primary identifier for components in all GridWorks registries.",
      "required": true
    },
    "ComponentAttributeClassId": {
      "type": "string",
      "format": "UuidCanonicalTextual",
      "title": "Unique identifier for ComponentAttributeClass object articulated by the␣
↪component.attribute.class.gt.000 type.",
      "description": "Unique identifier for the device class. Authority for these, as␣
↪well as the relationship between Components and ComponentAttributeClasses (Cacs) is␣
↪maintained by the World Registry.",
      "required": true
    },
    "DisplayName": {
```

(continues on next page)

(continued from previous page)

```
      "type": "string",
      "description": "This is an optional, mutable field whose use is strongly␣
↪encouraged. It may include information about HOW the component is used in a hardware␣
↪layout. It may also include the HwUid for the component.",
      "required": false
    },
    "HwUid": {
      "type": "string",
      "description": "Usually this is determined by the inheriting class.",
      "required": false
    },
    "TypeName": {
      "type": "string",
      "value": "component.gt",
      "title": "The type name"
    },
    "Version": {
      "type": "string",
      "title": "The type version",
      "default": "000",
      "required": true
    }
  },
  "example": {
    "ComponentId": "780788df-9706-4299-b116-304a48838338",
    "DisplayName": "Little Orange house Axeman Tank",
    "ComponentAttributeClassId": "683c193a-bf83-4491-a294-c0e32865a407",
    "TypeName": "component.gt",
    "Version": "000"
  },
  "formats": {
    "UuidCanonicalTextual": {
      "type": "string",
      "description": "A string of hex words separated by hyphens of length 8-4-4-4-12.",
      "example": "652ba6b0-c3bf-4f06-8a80-6b9832d60a25"
    }
  }
}
```

1.3.3 data.channel.000

```
{
  "gtr_asl": "001",
  "type_name": "data.channel",
  "version": "000",
  "owner": "gridworks@gridworks-consulting.com",
  "description": "Data Channel. A data channel is a concept of some collection of␣
↪readings that share all characteristics other than time.",
  "properties": {
    "DisplayName": {
```

(continues on next page)

(continued from previous page)

```
      "type": "string",
      "description": "This display name is the handle for the data channel. It is meant␣
↪to be set by the person/people who will be analyzing time series data. It is only␣
↪expected to be unique within the data channels associated to a specific Terminal Asset.
↪",
      "required": true
    },
    "AboutName": {
      "type": "string",
      "format": "SpaceheatName",
      "title": "About Name",
      "description": "The name of the SpaceheatNode whose physical quantities are␣
↪getting captured.",
      "required": true
    },
    "CapturedByName": {
      "type": "string",
      "format": "SpaceheatName",
      "title": "",
      "description": "The name of the SpaceheatNode that is capturing the physical␣
↪quantities (which can be AboutName but does not have to be).",
      "required": true
    },
    "TelemetryName": {
      "type": "string",
      "format": "spaceheat.telemetry.name",
      "title": "",
      "description": "The name of the physical quantity getting measured.",
      "required": true
    },
    "TypeName": {
      "type": "string",
      "value": "data.channel",
      "title": "The type name"
    },
    "Version": {
      "type": "string",
      "title": "The type version",
      "default": "000",
      "required": true
    }
  },
  "formats": {
    "SpaceheatName": {
      "type": "string",
      "description": "Lowercase words separated by periods, where the word characters␣
↪can be alphanumeric or a hyphen and the first word starts with an alphabet character.",
      "example": "store-hot-pipe"
    }
  }
}
```

1.3.4 egauge.io.000


```
{
  "gtr_asl": "001",
  "type_name": "egauge.io",
  "version": "000",
  "owner": "gridworks@gridworks-consulting.com",
  "description": "Used for an eGauge meter's component information in a hardware layout. When the component associated to a PowerMeter ShNode has MakeModel EGAUGE__4030, there is a significant amount of configuration required to specify both what is read from the eGauge (input) and what is then sent up to the SCADA (output). This type handles that information.",
  "url": "https://gridworks-protocol.readthedocs.io/en/latest/egauge-io.html",
  "properties": {
    "InputConfig": {
      "type": "egauge.register.config.000",
      "description": "This is the data available from the modbus csv map provided by eGauge for this component, for example http://egauge14875.egaug.es/6001C/settings.html for a eGauge device with ID 14875",
      "required": true
    },
    "OutputConfig": {
      "type": "telemetry.reporting.config.000",
      "description": "This is the data as the Scada proactor expects to consume it from the power meter driver proactor.",
      "required": true
    },
    "TypeName": {
      "type": "string",
      "value": "egauge.io",
      "title": "The type name"
    },
    "Version": {
      "type": "string",
      "title": "The type version",
      "default": "000",
      "required": true
    }
  },
  "example": {
    "InputConfig": {
      "Address": 9016,
      "Name": "house-panel-power",
      "Description": "change in value",
      "Type": "f32",
      "Denominator": 1,
      "Unit": "W",
      "TypeName": "egauge.register.config",
      "Version": "000"
    },
    "OutputConfig": {
      "TelemetryNameGtEnumSymbol": "af39eec9",
      "AboutNodeName": "house-panel-power",
```


(continues on next page)

(continued from previous page)

```
      "ReportOnChange": true,
      "SamplePeriodS": 300,
      "Exponent": 0,
      "UnitGtEnumSymbol": "f459a9c3",
      "AsyncReportThreshold": 0.02,
      "NameplateMaxValue": 3500,
      "TypeName": "telemetry.reporting.config",
      "Version": "000"
    },
    "TypeName": "egauge.io",
    "Version": "000"
  }
}
```

1.3.5 egauge.register.config.000

```
{
  "gtr_asl": "001",
  "type_name": "egauge.register.config",
  "version": "000",
  "owner": "gridworks@gridworks-consulting.com",
  "description": "Used to translate eGauge's Modbus Map. This type captures the␣
↪information provided by eGauge in its modbus csv map, when reading current, power,␣
↪energy, voltage, frequency etc from an eGauge 4030.",
  "properties": {
    "Address": {
      "type": "integer",
      "description": "EGauge's modbus holding address. Note that the EGauge modbus map␣
↪for holding address 100 will be 30100 - the '+30000' indicates it is a holding address.
↪ We use the 4-digit address after the '3'.",
      "required": true
    },
    "Name": {
      "type": "string",
      "description": "The name assigned in the EGauge's modbus map. This is configured␣
↪by the user (see URL)",
      "required": true
    },
    "Description": {
      "type": "string",
      "description": "Again, assigned by the EGauge modbus map. Is usually 'change in␣
↪value'",
      "required": true
    },
    "Type": {
      "type": "string",
      "description": "EGauge's numerical data type. Typically our power measurements are␣
↪f32 ( 32-bit floating-point number). The serial number & firmware are t16 (which work␣
↪to treat as 16-bit unsigned integer) and timestamps are u32 (32-bit unsigned integer).
↪",
```

(continues on next page)

(continued from previous page)

```
      "required": true
    },
    "Denominator": {
      "type": "integer",
      "description": "Some of the modbus registers divide by 3.60E+06 (cumulative energy␣
↪registers typically). For the power, current, voltage and phase angle the denominator␣
↪is 1.",
      "required": true
    },
    "Unit": {
      "type": "string",
      "description": "The EGauge unit - typically A, Hz, or W.",
      "required": true
    },
    "TypeName": {
      "type": "string",
      "value": "egauge.register.config",
      "title": "The type name"
    },
    "Version": {
      "type": "string",
      "title": "The type version",
      "default": "000",
      "required": true
    }
  },
  "example": {
    "Address": 9016,
    "Name": "house-panel-power",
    "Description": "change in value",
    "Type": "f32",
    "Denominator": 1,
    "Unit": "W",
    "TypeName": "egauge.register.config",
    "Version": "000"
  }
}
```

1.3.6 electric.meter.cac.gt.000

```
{
  "gtr_asl": "001",
  "type_name": "electric.meter.cac.gt",
  "version": "000",
  "owner": "gridworks@gridworks-consulting.com",
  "description": "Type for tracking  Electric Meter ComponentAttributeClasses. GridWorks␣
↪Spaceheat SCADA uses the GridWorks GNodeRegistry structures and abstractions for␣
↪managing relational device data. The Cac, or ComponentAttributeClass, is part of this␣
↪structure.",
  "url": "https://g-node-registry.readthedocs.io/en/latest/component-attribute-class.html
```

(continues on next page)

(continued from previous page)

```
↪",
  "properties": {
    "ComponentAttributeClassId": {
      "type": "string",
      "format": "UuidCanonicalTextual",
      "title": "ComponentAttributeClassId",
      "description": "Unique identifier for the device class (aka 'cac' or Component␣
↪Attribute Class). Authority is maintained by the World Registry.",
      "required": true
    },
    "MakeModel": {
      "type": "string",
      "format": "spaceheat.make.model",
      "title": "MakeModel",
      "description": "The brand name identifier for the electric meter (what you would␣
↪specify in order to buy one).",
      "required": true
    },
    "DisplayName": {
      "type": "string",
      "description": "Sample: EGauge 4030",
      "required": false
    },
    "TelemetryNameList": {
      "type": "array",
      "items": {
        "type": "string"
      },
      "format": "spaceheat.telemetry.name",
      "title": "TelemetryNames read by this power meter",
      "required": true
    },
    "PollPeriodMs": {
      "type": "integer",
      "description": "Poll Period refers to the period of time between two readings by␣
↪the local actor. This is in contrast to Capture Period, which refers to the period␣
↪between readings that are sent up to the cloud (or otherwise saved for the long-term).
↪",
      "required": true
    },
    "Interface": {
      "type": "string",
      "format": "local.comm.interface",
      "title": "",
      "required": true
    },
    "DefaultBaud": {
      "type": "integer",
      "required": false
    },
    "TypeName": {
      "type": "string",
```

(continues on next page)

(continued from previous page)

```
      "value": "electric.meter.cac.gt",
      "title": "The type name"
    },
    "Version": {
      "type": "string",
      "title": "The type version",
      "default": "000",
      "required": true
    }
  },
  "example": {
    "ComponentAttributeClassId": "739a6e32-bb9c-43bc-a28d-fb61be665522",
    "DisplayName": "EGauge 4030",
    "InterfaceGtEnumSymbol": "c1e7a955",
    "MakeModelGtEnumSymbol": "beb6d3fb",
    "PollPeriodMs": 1000,
    "TelemetryNameList": ["af39eec9"],
    "TypeName": "electric.meter.cac.gt",
    "Version": "000"
  },
  "formats": {
    "UuidCanonicalTextual": {
      "type": "string",
      "description": "A string of hex words separated by hyphens of length 8-4-4-4-12.",
      "example": "652ba6b0-c3bf-4f06-8a80-6b9832d60a25"
    },
    "PositiveInteger": {
      "type": "string",
      "description": "Must be positive when interpreted as an integer. Interpretation as␣
↪an integer follows the pydantic rules for this - which will round down rational␣
↪numbers. So 1.7 will be interpreted as 1 and is also fine, while 0.5 is interpreted as␣
↪0 and will raise an exception.",
      "example": ""
    }
  }
}
```

1.3.7 electric.meter.component.gt.000

```
{
  "gtr_asl": "001",
  "type_name": "electric.meter.component.gt",
  "version": "000",
  "owner": "gridworks@gridworks-consulting.com",
  "description": "Type for tracking Electric Meter Components. Designed for specific␣
↪instances of Electric Meters. It extends the component.gt.000 type. Authority for the␣
↪attributes of the component.gt.000 (ComponentId, ComponentAttributeClassId,␣
↪DisplayName, HwUid) belongs to the WorldRegistry. The WorldRegistry is part of the␣
↪GridWorks 'BackOffice' structure for managing relational device data . Notably,␣
↪ComponentId and ComponentAttributeClass are both required and immutable. HwUid is␣
```

(continues on next page)

(continued from previous page)

```
↪optional but once it is set to a non-null value that is also immutable - it is meant␣
↪to be an immutable identifier associated to a specific physical device, ideally one␣
↪that can be read remotely by the SCADA and also by the naked eye. The DisplayName is␣
↪mutable, with its current value in time governed by the WorldRegistry.",
  "url": "https://g-node-registry.readthedocs.io/en/latest/electric-meters.html",
  "properties": {
    "ComponentId": {
      "type": "string",
      "format": "UuidCanonicalTextual",
      "title": "Component Id",
      "description": "Primary GridWorks identifier for a specific physical instance of␣
↪an ElectricMeter, and also as a more generic Component.",
      "required": true
    },
    "ComponentAttributeClassId": {
      "type": "string",
      "format": "UuidCanonicalTextual",
      "title": "Unique identifier for ElectricMeterCac object articulated by the␣
↪electric.meter.cac.gt.000 type.",
      "description": "Unique identifier for the device class. Authority for these, as␣
↪well as the relationship between Components and ComponentAttributeClasses (Cacs) is␣
↪maintained by the World Registry.",
      "required": true
    },
    "DisplayName": {
      "type": "string",
      "description": "Sample: Oak EGauge6074",
      "required": false
    },
    "ConfigList": {
      "type": "array",
      "items": {
        "type": "telemetry.reporting.config.000"
      },
      "description": "This power meter will produce multiple data channels. Each data␣
↪channel measures a certain quantities (like power, current) for certain ShNodes (like␣
↪a boost element or heat pump).",
      "required": true
    },
    "HwUid": {
      "type": "string",
      "description": "For eGauge, use what comes back over modbus address 100.",
      "required": false
    },
    "ModbusHost": {
      "type": "string",
      "required": false
    },
    "ModbusPort": {
      "type": "integer",
      "format": "NonNegativeInteger",
      "title": "",
```

(continues on next page)

(continued from previous page)

```
      "required": false
    },
    "EgaugeIoList": {
      "type": "array",
      "items": {
        "type": "egauge.io.000"
      },
      "description": "This should be empty unless the MakeModel of the corresponding␣
↪component attribute class is EGauge 4030. The channels that can be read from an EGauge␣
↪4030 are configurable by the person who installs the device. The information is␣
↪encapsulated in a modbus map provided by eGauge as a csv from a device-specific API.␣
↪The EGaugeIoList maps the data from this map to the data that the SCADA expects to see.
↪",
      "required": true
    },
    "TypeName": {
      "type": "string",
      "value": "electric.meter.component.gt",
      "title": "The type name"
    },
    "Version": {
      "type": "string",
      "title": "The type version",
      "default": "000",
      "required": true
    }
  },
  "axioms": {
    "Axiom1": {
      "title": "Modbus consistency",
      "description": "ModbusHost is None if and only if ModbusPort is None"
    },
    "Axiom2": {
      "title": "Egauge4030 consistency",
      "description": "If the EgaugeIoList has non-zero length, then the ModbusHost is␣
↪not None and the set of output configs is equal to ConfigList as a set"
    }
  },
  "example": {
    "ComponentAttributeClassId": "739a6e32-bb9c-43bc-a28d-fb61be665522",
    "ComponentId": "36a31af8-5ff6-4105-a751-fb858889bc60",
    "DisplayName": "Oak EGauge6074",
    "HwUid": "BP01954",
    "ModbusHost": "eGauge6074.local",
    "ModbusPort": 502,
    "ConfigList": [
      {
        "AboutNodeName": "a.m.house.panel.power",
        "AsyncReportThreshold": 0.02,
        "Exponent": 0,
        "NameplateMaxValue": 3500,
        "ReportOnChange": true,
```

(continues on next page)

(continued from previous page)


```
      "SamplePeriodS": 300,
      "TelemetryNameGtEnumSymbol": "af39eec9",
      "TypeName": "telemetry.reporting.config",
      "UnitGtEnumSymbol": "f459a9c3",
      "Version": "000"
    },
    {
      "AboutNodeName": "oilpluspumpspower",
      "AsyncReportThreshold": 0.02,
      "Exponent": 0,
      "NameplateMaxValue": 1000,
      "ReportOnChange": true,
      "SamplePeriodS": 300,
      "TelemetryNameGtEnumSymbol": "af39eec9",
      "TypeName": "telemetry.reporting.config",
      "UnitGtEnumSymbol": "f459a9c3",
      "Version": "000"
    }
  ],
  "EgaugeIoList": [
    {
      "InputConfig": {
        "Address": 9016,
        "Denominator": 1,
        "Description": "change in value",
        "Name": "house-panel-power",
        "Type": "f32",
        "TypeName": "egauge.register.config",
        "Unit": "W",
        "Version": "000"
      },
      "OutputConfig": {
        "AboutNodeName": "a.m.house.panel.power",
        "AsyncReportThreshold": 0.02,
        "Exponent": 0,
        "NameplateMaxValue": 3500,
        "ReportOnChange": true,
        "SamplePeriodS": 300,
        "TelemetryNameGtEnumSymbol": "af39eec9",
        "TypeName": "telemetry.reporting.config",
        "UnitGtEnumSymbol": "f459a9c3",
        "Version": "000"
      },
      "TypeName": "egauge.io",
      "Version": "000"
    },
    {
      "InputConfig": {
        "Address": 9018,
        "Denominator": 1,
        "Description": "change in value",
        "Name": "oil-boiler-plus-pumps",
```


(continues on next page)

(continued from previous page)


```
          "Type": "f32",
          "TypeName": "egauge.register.config",
          "Unit": "W",
          "Version": "000"
        },
        "OutputConfig": {
          "AboutNodeName": "oilpluspumpspower",
          "AsyncReportThreshold": 0.02,
          "Exponent": 0,
          "NameplateMaxValue": 1000,
          "ReportOnChange": true,
          "SamplePeriodS": 300,
          "TelemetryNameGtEnumSymbol": "af39eec9",
          "TypeName": "telemetry.reporting.config",
          "UnitGtEnumSymbol": "f459a9c3",
          "Version": "000"
        },
        "TypeName": "egauge.io",
        "Version": "000"
      }
    ],
    "TypeName": "electric.meter.component.gt",
    "Version": "000"
  },
  "formats": {
    "UuidCanonicalTextual": {
      "type": "string",
      "description": "A string of hex words separated by hyphens of length 8-4-4-4-12.",
      "example": "652ba6b0-c3bf-4f06-8a80-6b9832d60a25"
    },
    "PositiveInteger": {
      "type": "string",
      "description": "Must be positive when interpreted as an integer. Interpretation as␣
↪an integer follows the pydantic rules for this - which will round down rational␣
↪numbers. So 1.7 will be interpreted as 1 and is also fine, while 0.5 is interpreted as␣
↪0 and will raise an exception.",
      "example": ""
    },
    "NonNegativeInteger": {
      "type": "string",
      "description": "Must be non-negative when interpreted as an integer.␣
↪Interpretation as an integer follows the pydantic rules for this - which will round␣
↪down rational numbers. So 0 is fine, and 1.7 will be interpreted as 1 and is also fine.
↪",
      "example": ""
    }
  }
}
```

1.3.8 fibaro.smart.implant.cac.gt.000


```
{
  "gtr_asl": "001",
  "type_name": "fibaro.smart.implant.cac.gt",
  "version": "000",
  "owner": "gridworks@gridworks-consulting.com",
  "description": "Fibaro Make/Model. A small IoT Z-Wave device with two analog sensors,␣
↪two digital outputs and a 1-wire temp sensor.",
  "url": "https://www.fibaro.com/us/products/smart-implant/",
  "properties": {
    "ComponentAttributeClassId": {
      "type": "string",
      "format": "UuidCanonicalTextual",
      "title": "ComponentAttributeClassId",
      "description": "Unique identifier for the device class (aka 'cac' or Component␣
↪Attribute Class). Authority is maintained by the World Registry.",
      "required": true
    },
    "Model": {
      "type": "string",
      "required": true
    },
    "DisplayName": {
      "type": "string",
      "description": "Sample: FGBS-222 v5.2",
      "required": false
    },
    "TypeName": {
      "type": "string",
      "value": "fibaro.smart.implant.cac.gt",
      "title": "The type name"
    },
    "Version": {
      "type": "string",
      "title": "The type version",
      "default": "000",
      "required": true
    }
  },
  "example": {
    "ComponentAttributeClassId": "7ce0ce69-14c6-4cb7-a33f-2aeca91e0680",
    "DisplayName": "Fibaro SmartImplant FGBS-222",
    "Model": "FGBS-222 v5.2",
    "TypeName": "fibaro.smart.implant.cac.gt",
    "Version": "000"
  },
  "formats": {
    "UuidCanonicalTextual": {
      "type": "string",
      "description": "A string of hex words separated by hyphens of length 8-4-4-4-12.",
      "example": "652ba6b0-c3bf-4f06-8a80-6b9832d60a25"
    }
```


(continues on next page)

(continued from previous page)

```
  }
}
```

1.3.9 fibaro.smart.implant.component.gt.000

```
{
  "gtr_asl": "001",
  "type_name": "fibaro.smart.implant.component.gt",
  "version": "000",
  "owner": "gridworks@gridworks-consulting.com",
  "description": "Fibaro Smart Implant. Designed for specific Fibaro Smart Implants. It␣
↪extends the component.gt.000 type. Authority for the attributes of the component.gt.
↪000 (ComponentId, ComponentAttributeClassId, DisplayName, HwUid) belongs to the␣
↪WorldRegistry. The WorldRegistry is part of the GridWorks 'BackOffice' structure for␣
↪managing relational device data . Notably, ComponentId and ComponentAttributeClass are␣
↪both required and immutable. HwUid is optional but once it is set to a non-null value␣
↪that is also immutable - it is meant to be an immutable identifier associated to a␣
↪specific physical device, ideally one that can be read remotely by the SCADA and also␣
↪by the naked eye. The DisplayName is mutable, with its current value in time governed␣
↪by the WorldRegistry.",
  "url": "https://www.fibaro.com/us/products/smart-implant/",
  "properties": {
    "ComponentId": {
      "type": "string",
      "format": "UuidCanonicalTextual",
      "title": "Component Id",
      "description": "Primary GridWorks identifier for a specific physical instance of␣
↪an Fibaro, and also as a more generic Component.",
      "required": true
    },
    "ComponentAttributeClassId": {
      "type": "string",
      "format": "UuidCanonicalTextual",
      "title": "Unique identifier for FibaroSmartImplantCac object articulated by the␣
↪fibaro.smart.implant.cac.gt.000 type.",
      "description": "Unique identifier for the device class. Authority for these, as␣
↪well as the relationship between Components and ComponentAttributeClasses (Cacs) is␣
↪maintained by the World Registry.",
      "required": true
    },
    "ZWaveDSK": {
      "type": "string",
      "description": "The Z-Wave DSK (Device Specific Key) is a unique identifier␣
↪associated with a Z-Wave device, used during the process of securely including the␣
↪device into a Z-Wave network. It helps establish secure communication between the Z-
↪Wave controller and the device, ensuring that only authorized devices can join the␣
↪network. Unfortunately Hubitat does not currently provide a way to view the ZWave DSK␣
↪of a Fibaro.",
      "required": true
    },
```

(continues on next page)

(continued from previous page)

```
    "DisplayName": {
      "type": "string",
      "description": "Sample: Fibaro Smart Implant 1010 A (For Fibaro A as opposed to B␣
↪for GridWorks TankModule1 with Serial Number 1010).",
      "required": false
    },
    "HwUid": {
      "type": "string",
      "description": "Use the Fibaro S2 PIN Code, which is printed on the back of each␣
↪Fibaro Implant.",
      "required": false
    },
    "TypeName": {
      "type": "string",
      "value": "fibaro.smart.implant.component.gt",
      "title": "The type name"
    },
    "Version": {
      "type": "string",
      "title": "The type version",
      "default": "000",
      "required": true
    }
  },
  "example": {
    "ComponentId": "1fdd40dd-14d7-4da2-8cf8-7cf66484e385",
    "ComponentAttributeClassId": "7ce0ce69-14c6-4cb7-a33f-2aeca91e0680",
    "DisplayName": "Fibaro 1010 A",
    "ZWaveDSK": "",
    "HwUid": "20134",
    "TypeName": "fibaro.smart.implant.component.gt",
    "Version": "000"
  },
  "formats": {
    "UuidCanonicalTextual": {
      "type": "string",
      "description": "A string of hex words separated by hyphens of length 8-4-4-4-12.",
      "example": "652ba6b0-c3bf-4f06-8a80-6b9832d60a25"
    }
  }
}
```

1.3.10 gt.dispatch.boolean.110


```
{
  "gtr_asl": "001",
  "type_name": "gt.dispatch.boolean",
  "version": "110",
  "owner": "gridworks@gridworks-consulting.com",
  "description": "GridWorks Type Boolean Dispatch. Boolean dispatch command designed to␣
↪be sent from an AtomicTNode to a SCADA.",
  "properties": {
    "AboutNodeName": {
      "type": "string",
      "format": "LeftRightDot",
      "title": "The Spaceheat Node getting dispatched",
      "required": true
    },
    "ToGNodeAlias": {
      "type": "string",
      "format": "LeftRightDot",
      "title": "GNodeAlias of the SCADA",
      "required": true
    },
    "FromGNodeAlias": {
      "type": "string",
      "format": "LeftRightDot",
      "title": "GNodeAlias of AtomicTNode",
      "required": true
    },
    "FromGNodeInstanceId": {
      "type": "string",
      "format": "UuidCanonicalTextual",
      "title": "GNodeInstance of the AtomicTNode",
      "required": true
    },
    "RelayState": {
      "type": "integer",
      "format": "Bit",
      "title": "Relay State (0 or 1)",
      "description": "A Relay State of `0` indicates the relay is OPEN (off). A Relay␣
↪State of `1` indicates the relay is CLOSED (on). Note that `0` means the relay is open␣
↪whether or not the relay is normally open or normally closed (For a normally open␣
↪relay, the relay is ENERGIZED when it is in state `0` and DE-ENERGIZED when it is in␣
↪state `1`.)",
      "required": true
    },
    "SendTimeUnixMs": {
      "type": "integer",
      "format": "ReasonableUnixTimeMs",
      "title": "Time the AtomicTNode sends the dispatch, by its clock",
      "required": true
    },
    "TypeName": {
      "type": "string",
```


(continues on next page)

(continued from previous page)

```
      "value": "gt.dispatch.boolean",
      "title": "The type name"
    },
    "Version": {
      "type": "string",
      "title": "The type version",
      "default": "110",
      "required": true
    }
  },
  "formats": {
    "UuidCanonicalTextual": {
      "type": "string",
      "description": "A string of hex words separated by hyphens of length 8-4-4-4-12.",
      "example": "652ba6b0-c3bf-4f06-8a80-6b9832d60a25"
    },
    "Bit": {
      "type": "string",
      "description": "The value must be the integer 0 or the integer 1. Will not attempt␣
↪to first interpret as an integer. For example, 1.3 will not be interpreted as 1 but␣
↪will raise an error.",
      "example": ""
    },
    "LeftRightDot": {
      "type": "string",
      "description": "Lowercase alphanumeric words separated by periods, with the most␣
↪significant word (on the left) starting with an alphabet character.",
      "example": "dw1.isone.me.freedom.apple"
    },
    "ReasonableUnixTimeMs": {
      "type": "string",
      "description": "An integer reflecting unix time in MILLISECONDS between midnight␣
↪Jan 1 2000 and midnight Jan 1 3000 UTC",
      "example": "1702327940710"
    }
  }
}
```

1.3.11 gt.dispatch.boolean.local.110

```
{
  "gtr_asl": "001",
  "type_name": "gt.dispatch.boolean.local",
  "version": "110",
  "owner": "gridworks@gridworks-consulting.com",
  "description": "Dispatch message sent locally by SCADA HomeAlone actor. By Locally,␣
↪this means sent without access to Internet. The HomeAlone actor must reside within the␣
↪Local Area Network of the SCADA - typically it should reside on the same hardware.",
  "properties": {
    "RelayState": {
```

(continues on next page)

(continued from previous page)

```
      "type": "integer",
      "format": "Bit",
      "title": "Relay State (0 or 1)",
      "description": "A Relay State of `0` indicates the relay is OPEN (off). A Relay␣
↪State of `1` indicates the relay is CLOSED (on). Note that `0` means the relay is open␣
↪whether or not the relay is normally open or normally closed (For a normally open␣
↪relay, the relay is ENERGIZED when it is in state `0` and DE-ENERGIZED when it is in␣
↪state `1`.)",
      "required": true
    },
    "AboutNodeName": {
      "type": "string",
      "format": "LeftRightDot",
      "title": "About Node Name",
      "description": "The boolean actuator Spaceheat Node getting turned on or off.",
      "required": true
    },
    "FromNodeName": {
      "type": "string",
      "format": "LeftRightDot",
      "title": "From Node Name",
      "description": "The Spaceheat Node sending the command.",
      "required": true
    },
    "SendTimeUnixMs": {
      "type": "integer",
      "format": "ReasonableUnixTimeMs",
      "title": "Send Time in Unix Milliseconds",
      "required": true
    },
    "TypeName": {
      "type": "string",
      "value": "gt.dispatch.boolean.local",
      "title": "The type name"
    },
    "Version": {
      "type": "string",
      "title": "The type version",
      "default": "110",
      "required": true
    }
  },
  "formats": {
    "Bit": {
      "type": "string",
      "description": "The value must be the integer 0 or the integer 1. Will not attempt␣
↪to first interpret as an integer. For example, 1.3 will not be interpreted as 1 but␣
↪will raise an error.",
      "example": ""
    },
    "LeftRightDot": {
      "type": "string",
```

(continues on next page)

(continued from previous page)

```
      "description": "Lowercase alphanumeric words separated by periods, with the most␣
↪significant word (on the left) starting with an alphabet character.",
      "example": "dw1.isone.me.freedom.apple"
    },
    "ReasonableUnixTimeMs": {
      "type": "string",
      "description": "An integer reflecting unix time in MILLISECONDS between midnight␣
↪Jan 1 2000 and midnight Jan 1 3000 UTC",
      "example": "1702327940710"
    }
  }
}
```

1.3.12 gt.driver.booleanactuator.cmd.100

```
{
  "gtr_asl": "001",
  "type_name": "gt.driver.booleanactuator.cmd",
  "version": "100",
  "owner": "gridworks@gridworks-consulting.com",
  "description": "Boolean Actuator Driver Command. The boolean actuator actor reports␣
↪when it has sent an actuation command to its driver so that the SCADA can add this to␣
↪information to be sent up to the AtomicTNode.",
  "url": "https://gridworks.readthedocs.io/en/latest/relay-state.html",
  "properties": {
    "RelayState": {
      "type": "integer",
      "format": "Bit",
      "title": "",
      "required": true
    },
    "ShNodeAlias": {
      "type": "string",
      "format": "LeftRightDot",
      "title": "",
      "required": true
    },
    "CommandTimeUnixMs": {
      "type": "integer",
      "format": "ReasonableUnixTimeMs",
      "title": "",
      "required": true
    },
    "TypeName": {
      "type": "string",
      "value": "gt.driver.booleanactuator.cmd",
      "title": "The type name"
    },
    "Version": {
      "type": "string",
```

(continues on next page)

(continued from previous page)

```
      "title": "The type version",
      "default": "100",
      "required": true
    }
  },
  "formats": {
    "Bit": {
      "type": "string",
      "description": "The value must be the integer 0 or the integer 1. Will not attempt␣
↪to first interpret as an integer. For example, 1.3 will not be interpreted as 1 but␣
↪will raise an error.",
      "example": ""
    },
    "LeftRightDot": {
      "type": "string",
      "description": "Lowercase alphanumeric words separated by periods, with the most␣
↪significant word (on the left) starting with an alphabet character.",
      "example": "dw1.isone.me.freedom.apple"
    },
    "ReasonableUnixTimeMs": {
      "type": "string",
      "description": "An integer reflecting unix time in MILLISECONDS between midnight␣
↪Jan 1 2000 and midnight Jan 1 3000 UTC",
      "example": "1702327940710"
    }
  }
}
```

1.3.13 gt.sh.booleanactuator.cmd.status.100

```
{
  "gtr_asl": "001",
  "type_name": "gt.sh.booleanactuator.cmd.status",
  "version": "100",
  "owner": "gridworks@gridworks-consulting.com",
  "description": "Boolean  Actuator Driver Command Status Package. This is a subtype of␣
↪the status message sent from a SCADA to its AtomicTNode. It contains a list of all the␣
↪commands that a particular boolean actuator actor has reported as sending as actuation␣
↪commands to its driver in the last transmission period (typically 5 minutes).",
  "url": "https://gridworks.readthedocs.io/en/latest/relay-state.html",
  "properties": {
    "ShNodeAlias": {
      "type": "string",
      "format": "LeftRightDot",
      "title": "SpaceheatNodeAlias",
      "description": "The alias of the spaceheat node that is getting actuated. For␣
↪example, `a.elt1.relay` would likely indicate the relay for a resistive element.",
      "required": true
    },
    "RelayStateCommandList": {
```

(continues on next page)

(continued from previous page)

```
      "type": "array",
      "items": {
        "type": "integer"
      },
      "required": true
    },
    "CommandTimeUnixMsList": {
      "type": "array",
      "items": {
        "type": "integer"
      },
      "format": "ReasonableUnixTimeMs",
      "title": "List of Command Times",
      "required": true
    },
    "TypeName": {
      "type": "string",
      "value": "gt.sh.booleanactuator.cmd.status",
      "title": "The type name"
    },
    "Version": {
      "type": "string",
      "title": "The type version",
      "default": "100",
      "required": true
    }
  },
  "formats": {
    "LeftRightDot": {
      "type": "string",
      "description": "Lowercase alphanumeric words separated by periods, with the most␣
↪significant word (on the left) starting with an alphabet character.",
      "example": "dw1.isone.me.freedom.apple"
    },
    "ReasonableUnixTimeMs": {
      "type": "string",
      "description": "An integer reflecting unix time in MILLISECONDS between midnight␣
↪Jan 1 2000 and midnight Jan 1 3000 UTC",
      "example": "1702327940710"
    }
  }
}
```

1.3.14 gt.sh.cli.atn.cmd.110


```
{
  "gtr_asl": "001",
  "type_name": "gt.sh.cli.atn.cmd",
  "version": "110",
  "owner": "gridworks@gridworks-consulting.com",
  "description": "AtomicTNode CLI Command. This is a generic type mechanism for a crude␣
↪command line interface on a SCADA, brokered by the AtomicTNode.",
  "properties": {
    "FromGNodeAlias": {
      "type": "string",
      "format": "LeftRightDot",
      "title": "GNodeAlias",
      "description": "Must be the SCADA's AtomicTNode.",
      "required": true
    },
    "SendSnapshot": {
      "type": "boolean",
      "description": "Asks SCADA to send back a snapshot. For this version of the type,␣
↪nothing would happen if SendSnapshot were set to False. However, we include this in␣
↪case additional variations are added later.",
      "required": true
    },
    "FromGNodeId": {
      "type": "string",
      "format": "UuidCanonicalTextual",
      "title": "GNodeId",
      "required": true
    },
    "TypeName": {
      "type": "string",
      "value": "gt.sh.cli.atn.cmd",
      "title": "The type name"
    },
    "Version": {
      "type": "string",
      "title": "The type version",
      "default": "110",
      "required": true
    }
  },
  "formats": {
    "UuidCanonicalTextual": {
      "type": "string",
      "description": "A string of hex words separated by hyphens of length 8-4-4-4-12.",
      "example": "652ba6b0-c3bf-4f06-8a80-6b9832d60a25"
    },
    "LeftRightDot": {
      "type": "string",
      "description": "Lowercase alphanumeric words separated by periods, with the most␣
↪significant word (on the left) starting with an alphabet character.",
      "example": "dw1.isone.me.freedom.apple"
```


(continues on next page)

(continued from previous page)

```
    }
  }
}
```

1.3.15 gt.sh.multipurpose.telemetry.status.100

```
{
  "gtr_asl": "001",
  "type_name": "gt.sh.multipurpose.telemetry.status",
  "version": "100",
  "owner": "gridworks@gridworks-consulting.com",
  "description": "Data read from a MultipurposeSensor run by a Spaceheat SCADA. A list␣
↪of readings about a specific SpaceheatNode made by a MultipurposeSensor node, for a␣
↪Spaceheat SCADA. Designed as part of a status message sent from the SCADA to its␣
↪AtomicTNode typically once every 5 minutes. The nth element of each of its two lists␣
↪refer to the same reading (i.e. what the value is, when it was read).",
  "url": "https://gridworks-protocol.readthedocs.io/en/latest/multipurpose-sensor.html",
  "properties": {
    "AboutNodeAlias": {
      "type": "string",
      "format": "LeftRightDot",
      "title": "AboutNodeAlias",
      "description": "The SpaceheatNode representing the physical object that the sensor␣
↪reading is collecting data about. For example, a multipurpose temp sensor that reads␣
↪12 temperatures would have data for 12 different AboutNodeAliases, including say `a.
↪tank1.temp1` for a temp sensor at the top of a water tank.",
      "required": true
    },
    "SensorNodeAlias": {
      "type": "string",
      "description": "The alias of the SpaceheatNode representing the telemetry device",
      "required": true
    },
    "TelemetryName": {
      "type": "string",
      "format": "spaceheat.telemetry.name",
      "title": "TelemetryName",
      "description": "The TelemetryName of the readings. This is used to interpet the␣
↪meaning of the reading values. For example, WaterTempCTimes1000 means the reading is␣
↪measuring the a reading of 37 deg C.",
      "required": true
    },
    "ValueList": {
      "type": "array",
      "items": {
        "type": "integer"
      },
      "description": "The values of the readings.",
      "required": true
    },
```

(continues on next page)

(continued from previous page)

```
    "ReadTimeUnixMsList": {
      "type": "array",
      "items": {
        "type": "integer"
      },
      "format": "ReasonableUnixTimeMs",
      "title": "List of Read Times",
      "description": "The times that the MultipurposeSensor took the readings, in unix␣
↪milliseconds",
      "required": true
    },
    "TypeName": {
      "type": "string",
      "value": "gt.sh.multipurpose.telemetry.status",
      "title": "The type name"
    },
    "Version": {
      "type": "string",
      "title": "The type version",
      "default": "100",
      "required": true
    }
  },
  "axioms": {
    "Axiom1": {
      "title": "ListLengthConsistency",
      "description": "ValueList and ReadTimeUnixMsList must have the same length."
    }
  },
  "formats": {
    "LeftRightDot": {
      "type": "string",
      "description": "Lowercase alphanumeric words separated by periods, with the most␣
↪significant word (on the left) starting with an alphabet character.",
      "example": "dw1.isone.me.freedom.apple"
    },
    "ReasonableUnixTimeMs": {
      "type": "string",
      "description": "An integer reflecting unix time in MILLISECONDS between midnight␣
↪Jan 1 2000 and midnight Jan 1 3000 UTC",
      "example": "1702327940710"
    }
  }
}
```

1.3.16 gt.sh.simple.telemetry.status.100


```
{
  "gtr_asl": "001",
  "type_name": "gt.sh.simple.telemetry.status",
  "version": "100",
  "owner": "gridworks@gridworks-consulting.com",
  "description": "Data read from a SimpleSensor run by a SpaceHeat SCADA. A list of␣
↪readings from a simple sensor for a Spaceheat SCADA. Designed as part of a status␣
↪message sent from the SCADA to its AtomicTNode typically once every 5 minutes. The nth␣
↪element of each of its two lists refer to the same reading (i.e. what the value is,␣
↪when it was read).",
  "url": "https://gridworks-protocol.readthedocs.io/en/latest/simple-sensor.html",
  "properties": {
    "ShNodeAlias": {
      "type": "string",
      "format": "LeftRightDot",
      "title": "SpaceheatNodeAlias",
      "description": "The Alias of the SimpleSensor associated to the readings",
      "required": true
    },
    "TelemetryName": {
      "type": "string",
      "format": "spaceheat.telemetry.name",
      "title": "TelemetryName",
      "description": "The TelemetryName of the readings. This is used to interpet the␣
↪meaning of the reading values. For example, WaterTempCTimes1000 means the reading is␣
↪measuring the temperature of water, in Celsius multiplied by 1000. So a value of 37000␣
↪would be a reading of 37 deg C.",
      "required": true
    },
    "ValueList": {
      "type": "array",
      "items": {
        "type": "integer"
      },
      "description": "The values of the readings.",
      "required": true
    },
    "ReadTimeUnixMsList": {
      "type": "array",
      "items": {
        "type": "integer"
      },
      "format": "ReasonableUnixTimeMs",
      "title": "List of Read Times",
      "description": "The times that the SImpleSensor took the readings, in unix␣
↪milliseconds",
      "required": true
    },
    "TypeName": {
      "type": "string",
      "value": "gt.sh.simple.telemetry.status",
```


(continues on next page)

(continued from previous page)

```
      "title": "The type name"
    },
    "Version": {
      "type": "string",
      "title": "The type version",
      "default": "100",
      "required": true
    }
  },
  "axioms": {
    "Axiom1": {
      "title": "ListLengthConsistency",
      "description": "ValueList and ReadTimeUnixMsList must have the same length."
    }
  },
  "formats": {
    "LeftRightDot": {
      "type": "string",
      "description": "Lowercase alphanumeric words separated by periods, with the most␣
↪significant word (on the left) starting with an alphabet character.",
      "example": "dw1.isone.me.freedom.apple"
    },
    "ReasonableUnixTimeMs": {
      "type": "string",
      "description": "An integer reflecting unix time in MILLISECONDS between midnight␣
↪Jan 1 2000 and midnight Jan 1 3000 UTC",
      "example": "1702327940710"
    }
  }
}
```

1.3.17 gt.sh.status.110

```
{
  "gtr_asl": "001",
  "type_name": "gt.sh.status",
  "version": "110",
  "owner": "gridworks@gridworks-consulting.com",
  "description": ". Status message sent by a Spaceheat SCADA every 5 minutes",
  "properties": {
    "FromGNodeAlias": {
      "type": "string",
      "format": "LeftRightDot",
      "title": "",
      "required": true
    },
    "FromGNodeId": {
      "type": "string",
      "format": "UuidCanonicalTextual",
      "title": "",
```

(continues on next page)

(continued from previous page)

```
      "required": true
    },
    "AboutGNodeAlias": {
      "type": "string",
      "format": "LeftRightDot",
      "title": "",
      "required": true
    },
    "SlotStartUnixS": {
      "type": "integer",
      "format": "ReasonableUnixTimeS",
      "title": "",
      "required": true
    },
    "ReportingPeriodS": {
      "type": "integer",
      "required": true
    },
    "SimpleTelemetryList": {
      "type": "array",
      "items": {
        "type": "gt.sh.simple.telemetry.status.100"
      },
      "required": true
    },
    "MultipurposeTelemetryList": {
      "type": "array",
      "items": {
        "type": "gt.sh.multipurpose.telemetry.status.100"
      },
      "required": true
    },
    "BooleanactuatorCmdList": {
      "type": "array",
      "items": {
        "type": "gt.sh.booleanactuator.cmd.status.100"
      },
      "required": true
    },
    "StatusUid": {
      "type": "string",
      "format": "UuidCanonicalTextual",
      "title": "",
      "required": true
    },
    "TypeName": {
      "type": "string",
      "value": "gt.sh.status",
      "title": "The type name"
    },
    "Version": {
      "type": "string",
```

(continues on next page)

(continued from previous page)

```
      "title": "The type version",
      "default": "110",
      "required": true
    }
  },
  "formats": {
    "ReasonableUnixTimeS": {
      "type": "string",
      "description": "Integer reflecting unix time seconds between 1970 and 3000",
      "example": ""
    },
    "UuidCanonicalTextual": {
      "type": "string",
      "description": "A string of hex words separated by hyphens of length 8-4-4-4-12.",
      "example": "652ba6b0-c3bf-4f06-8a80-6b9832d60a25"
    },
    "LeftRightDot": {
      "type": "string",
      "description": "Lowercase alphanumeric words separated by periods, with the most␣
↪significant word (on the left) starting with an alphabet character.",
      "example": "dw1.isone.me.freedom.apple"
    }
  }
}
```

1.3.18 gt.sh.telemetry.from.multipurpose.sensor.100

```
{
  "gtr_asl": "001",
  "type_name": "gt.sh.telemetry.from.multipurpose.sensor",
  "version": "100",
  "owner": "gridworks@gridworks-consulting.com",
  "description": "Data sent from a MultipurposeSensor to a Spaceheat SCADA. A set of␣
↪readings made at the same time by a multipurpose sensor, sent by the␣
↪MultipurposeSensor SpaceheatNode actor to its SCADA. The nth element of each of its␣
↪three readings (what is getting read, what the value is, what the TelemetryNames are).
↪",
  "url": "https://gridworks-protocol.readthedocs.io/en/latest/multipurpose-sensor.html",
  "properties": {
    "ScadaReadTimeUnixMs": {
      "type": "integer",
      "format": "ReasonableUnixTimeMs",
      "title": "ScadaReadTime in Unix MilliSeconds",
      "required": true
    },
    "AboutNodeAliasList": {
      "type": "array",
      "items": {
        "type": "string"
      },
```

(continues on next page)

(continued from previous page)

```
      "format": "LeftRightDot",
      "title": "AboutNodeAliasList",
      "description": "List of aliases of the SpaceHeat Nodes getting measured",
      "required": true
    },
    "TelemetryNameList": {
      "type": "array",
      "items": {
        "type": "string"
      },
      "format": "spaceheat.telemetry.name",
      "title": "TelemetryNameList",
      "description": "List of the TelemetryNames. The nth name in this list indicates␣
↪the TelemetryName of the nth alias in the AboutNodeAliasList.",
      "required": true
    },
    "ValueList": {
      "type": "array",
      "items": {
        "type": "integer"
      },
      "required": true
    },
    "TypeName": {
      "type": "string",
      "value": "gt.sh.telemetry.from.multipurpose.sensor",
      "title": "The type name"
    },
    "Version": {
      "type": "string",
      "title": "The type version",
      "default": "100",
      "required": true
    }
  },
  "axioms": {
    "Axiom1": {
      "title": "ListLengthConsistency",
      "description": "AboutNodeAliasList, ValueList and TelemetryNameList must all have␣
↪the same length."
    }
  },
  "formats": {
    "LeftRightDot": {
      "type": "string",
      "description": "Lowercase alphanumeric words separated by periods, with the most␣
↪significant word (on the left) starting with an alphabet character.",
      "example": "dw1.isone.me.freedom.apple"
    },
    "ReasonableUnixTimeMs": {
      "type": "string",
      "description": "An integer reflecting unix time in MILLISECONDS between midnight␣
```

(continues on next page)

(continued from previous page)

```
↪Jan 1 2000 and midnight Jan 1 3000 UTC",
      "example": "1702327940710"
    }
  }
}
```

1.3.19 gt.telemetry.110

```
{
  "gtr_asl": "001",
  "type_name": "gt.telemetry",
  "version": "110",
  "owner": "gridworks@gridworks-consulting.com",
  "description": "Data sent from a SimpleSensor to a SCADA. This type is meant to be␣
↪used by a SimpleSensor, where _what_ is doing the reading can be conflated with _what_␣
↪is being read.",
  "properties": {
    "ScadaReadTimeUnixMs": {
      "type": "integer",
      "format": "ReasonableUnixTimeMs",
      "title": "Scada Read Time in Unix Milliseconds",
      "required": true
    },
    "Value": {
      "type": "integer",
      "description": "The value of the reading.",
      "required": true
    },
    "Name": {
      "type": "string",
      "format": "spaceheat.telemetry.name",
      "title": "Name",
      "description": "The name of the Simple Sensing Spaceheat Node. This is both the␣
↪AboutNodeName and FromNodeName for a data channel. The TelemetryName (and thus Units)␣
↪are expected to be inferred by the Spaceheat Node. For example this is done initially␣
↪in SCADA code according to whether the component of the Node is a␣
↪PipeFlowSensorComponent, SimpleTempSensorComponent etc.",
      "required": true
    },
    "Exponent": {
      "type": "integer",
      "description": "Say the TelemetryName is WaterTempCTimes1000; this corresponds to␣
↪units of Celsius. To match the implication in the name, the Exponent should be 3, and␣
↪a Value of 65300 would indicate 65.3 deg C",
      "required": true
    },
    "TypeName": {
      "type": "string",
      "value": "gt.telemetry",
      "title": "The type name"
```

(continues on next page)

(continued from previous page)

```
    },
    "Version": {
      "type": "string",
      "title": "The type version",
      "default": "110",
      "required": true
    }
  },
  "formats": {
    "ReasonableUnixTimeMs": {
      "type": "string",
      "description": "An integer reflecting unix time in MILLISECONDS between midnight␣
↪Jan 1 2000 and midnight Jan 1 3000 UTC",
      "example": "1702327940710"
    }
  }
}
```

1.3.20 heartbeat.b.001

```
{
  "gtr_asl": "001",
  "type_name": "heartbeat.b",
  "version": "001",
  "owner": "gridworks@gridworks-consulting.com",
  "description": "Heartbeat B. This is the Heartbeat intended to be sent between the␣
↪Scada and the AtomicTNode to allow for block-chain validation of the status of their␣
↪communication.",
  "url": "https://gridworks.readthedocs.io/en/latest/dispatch-contract.html",
  "properties": {
    "FromGNodeAlias": {
      "type": "string",
      "format": "LeftRightDot",
      "title": "My GNodeAlias",
      "required": true
    },
    "FromGNodeInstanceId": {
      "type": "string",
      "format": "UuidCanonicalTextual",
      "title": "My GNodeInstanceId",
      "required": true
    },
    "MyHex": {
      "type": "string",
      "format": "HexChar",
      "title": "Hex character getting sent",
      "required": true
    },
    "YourLastHex": {
      "type": "string",
```

(continues on next page)

(continued from previous page)

```
      "format": "HexChar",
      "title": "Last hex character received from heartbeat partner.",
      "required": true
    },
    "LastReceivedTimeUnixMs": {
      "type": "integer",
      "format": "ReasonableUnixTimeMs",
      "title": "Time YourLastHex was received on my clock",
      "required": true
    },
    "SendTimeUnixMs": {
      "type": "integer",
      "format": "ReasonableUnixTimeMs",
      "title": "Time this message is made and sent on my clock",
      "required": true
    },
    "StartingOver": {
      "type": "boolean",
      "description": "(typically the AtomicTNode in an AtomicTNode / SCADA pair) wants␣
↪to start the heartbeating volley over. The result is that its partner will not expect␣
↪the initiator to know its last Hex.",
      "required": true
    },
    "TypeName": {
      "type": "string",
      "value": "heartbeat.b",
      "title": "The type name"
    },
    "Version": {
      "type": "string",
      "title": "The type version",
      "default": "001",
      "required": true
    }
  },
  "formats": {
    "UuidCanonicalTextual": {
      "type": "string",
      "description": "A string of hex words separated by hyphens of length 8-4-4-4-12.",
      "example": "652ba6b0-c3bf-4f06-8a80-6b9832d60a25"
    },
    "HexChar": {
      "type": "string",
      "description": "single-char string in '0123456789abcdefABCDEF'",
      "example": "d"
    },
    "LeftRightDot": {
      "type": "string",
      "description": "Lowercase alphanumeric words separated by periods, with the most␣
↪significant word (on the left) starting with an alphabet character.",
      "example": "dw1.isone.me.freedom.apple"
    },
```

(continues on next page)

(continued from previous page)

```
    "ReasonableUnixTimeMs": {
      "type": "string",
      "description": "An integer reflecting unix time in MILLISECONDS between midnight␣
↪Jan 1 2000 and midnight Jan 1 3000 UTC",
      "example": "1702327940710"
    }
  }
}
```

1.3.21 hubitat.cac.gt.000

```
{
  "gtr_asl": "001",
  "type_name": "hubitat.cac.gt",
  "version": "000",
  "owner": "gridworks@gridworks-consulting.com",
  "description": "Hubitat Component Attribute Class (GridWorks Type). Hubitat is a␣
↪company that makes IoT hubs. This is a type for MakeModels made by Hubitat.",
  "url": "https://gridworks-protocol.readthedocs.io/en/latest/iot-hubs.html",
  "properties": {
    "ComponentAttributeClassId": {
      "type": "string",
      "format": "UuidCanonicalTextual",
      "title": "ComponentAttributeClassId",
      "description": "Unique identifier for the device class (aka 'cac' or Component␣
↪Attribute Class). Authority is maintained by the World Registry.",
      "required": true
    },
    "DisplayName": {
      "type": "string",
      "description": "Sample: Hubitat Elevation C-7",
      "required": false
    },
    "TypeName": {
      "type": "string",
      "value": "hubitat.cac.gt",
      "title": "The type name"
    },
    "Version": {
      "type": "string",
      "title": "The type version",
      "default": "000",
      "required": true
    }
  },
  "example": {
    "ComponentAttributeClassId": "62528da5-b510-4ac2-82c1-3782842eae07",
    "DisplayName": "Hubitat Elevation C-7",
    "TypeName": "hubitat.cac.gt",
    "Version": "000"
```

(continues on next page)

(continued from previous page)

```
  }
}
```

1.3.22 hubitat.component.gt.000

```
{
  "gtr_asl": "001",
  "type_name": "hubitat.component.gt",
  "version": "000",
  "owner": "gridworks@gridworks-consulting.com",
  "description": "Hubitat Component (GridWorks Type). Designed for specific Hubitat hubs.
↪ It extends the component.gt.000 type. Authority for the attributes of the component.
↪gt.000 (ComponentId, ComponentAttributeClassId, DisplayName, HwUid) belongs to the␣
↪WorldRegistry. The WorldRegistry is part of the GridWorks 'BackOffice' structure for␣
↪managing relational device data . Notably, ComponentId and ComponentAttributeClass are␣
↪both required and immutable. HwUid is optional but once it is set to a non-null value␣
↪that is also immutable - it is meant to be an immutable identifier associated to a␣
↪specific physical device, ideally one that can be read remotely by the SCADA and also␣
↪by the naked eye. The DisplayName is mutable, with its current value in time governed␣
↪by the WorldRegistry.",
  "url": "https://gridworks-protocol.readthedocs.io/en/latest/iot-hubs.html",
  "properties": {
    "ComponentId": {
      "type": "string",
      "format": "UuidCanonicalTextual",
      "title": "Component Id",
      "description": "Primary GridWorks identifier for a specific physical instance of a␣
↪Hubitat, and also as a more generic Component.",
      "required": true
    },
    "ComponentAttributeClassId": {
      "type": "string",
      "format": "UuidCanonicalTextual",
      "title": "Unique identifier for HubitatCac object articulated by the hubitat.cac.
↪gt.000 type.",
      "description": "Unique identifier for the device class. Authority for these, as␣
↪well as the relationship between Components and ComponentAttributeClasses (Cacs) is␣
↪maintained by the World Registry.",
      "required": true
    },
    "Hubitat": {
      "type": "dictDict",
      "description": "Includes the information needed to access the MakerAPI of a␣
↪Hubitat on the Local area network: Host, MakerApiID, AccessToken and MacAddress for␣
↪the Hubitat.",
      "required": true
    },
    "DisplayName": {
      "type": "string",
      "description": "Sample: Oak Hubitat 81:37:82 (using the last 6 digits of the␣
```

(continues on next page)

(continued from previous page)

```
↪Hubitat MacId in the display name, as well as the short alias for the associated g␣
↪node.)",
      "required": false
    },
    "HwUid": {
      "type": "string",
      "description": "Use the final 6 characters of the Hubitat mac address.",
      "required": false
    },
    "TypeName": {
      "type": "string",
      "value": "hubitat.component.gt",
      "title": "The type name"
    },
    "Version": {
      "type": "string",
      "title": "The type version",
      "default": "000",
      "required": true
    }
  },
  "axioms": {
    "Axiom1": {
      "title": "Hubitat.MacAddressId must have MacAddress format",
      "description": "Mac Address format is 6 pairs of double hex digits separated by␣
↪colons, agnostic to caps. - e.g. '34:E1:D1:81:37:82' and '34:e1:d1:81:37:82' both␣
↪satisfy this property. This works in python: MAC_REGEX = re.compile('[0-9a-f]{2}([-:]?
↪)[0-9a-f]{2}(\\1[0-9a-f]{2}){4}$') and then bool(MAC_REGEX.match(mac_str.lower()))"
    },
    "Axiom2": {
      "title": "HwUid matches Hubitat MacAddress",
      "description": "The HwUid must exist, and it must be lower alphanumeric versions␣
↪of the last 6 digits of the Hubitat.MacAddressId with the colons taken out. For␣
↪example, if the HubitatMacAddressId is '34:E1:D1:81:37:8A' then the HwUid must be
↪'81378a'."
    }
  },
  "example": {
    "ComponentAttributeClassId": "62528da5-b510-4ac2-82c1-3782842eae07",
    "ComponentId": "48039704-7d45-4937-adda-0e362d13cef6",
    "DisplayName": "Oak Hubitat 81:37:82",
    "Hubitat": {
      "AccessToken": "a8232144-abe9-4eed-bcfd-8f182600b8e7",
      "Host": "hubitat-keene-oak.local",
      "MacAddress": "34:E1:D1:81:37:82",
      "MakerApiId": 2
    },
    "HwUid": "813782",
    "TypeName": "hubitat.component.gt",
    "Version": "000"
  }
}
```

1.3.23 hubitat.poller.cac.gt.000


```
{
  "gtr_asl": "001",
  "type_name": "hubitat.poller.cac.gt",
  "version": "000",
  "owner": "gridworks@gridworks-consulting.com",
  "description": "Hubitat Poller Cac (GridWorks Type). A class of devices - like a␣
↪Honeywell Z-Wave T6 Thermostat - that can be polled through a Hubitat IoT hub.",
  "properties": {
    "ComponentAttributeClassId": {
      "type": "string",
      "description": "Unique identifier for the device class (aka 'cac' or Component␣
↪Attribute Class). Authority is maintained by the World Registry.",
      "required": true
    },
    "DisplayName": {
      "type": "string",
      "description": "Sample: Honeywell T6 ZWave Thermostat",
      "required": false
    },
    "TypeName": {
      "type": "string",
      "value": "hubitat.poller.cac.gt",
      "title": "The type name"
    },
    "Version": {
      "type": "string",
      "title": "The type version",
      "default": "000",
      "required": true
    }
  },
  "formats": {
    "UuidCanonicalTextual": {
      "type": "string",
      "description": "A string of hex words separated by hyphens of length 8-4-4-4-12.",
      "example": "652ba6b0-c3bf-4f06-8a80-6b9832d60a25"
    }
  }
}
```


1.3.24 hubitat.poller.component.gt.000


```
{
  "gtr_asl": "001",
  "type_name": "hubitat.poller.component.gt",
  "version": "000",
  "owner": "gridworks@gridworks-consulting.com",
  "description": "Hubitat Poller Component (GridWorks Types). A specific instance of a␣
↪Hubitat Poller Cac (like a Honeywell T6 Thermostat) - a device that can be polled␣
```


(continues on next page)

(continued from previous page)

```
↪through a Hubitat IoT hub.",
  "properties": {
    "ComponentId": {
      "type": "string",
      "required": true
    },
    "ComponentAttributeClassId": {
      "type": "string",
      "format": "UuidCanonicalTextual",
      "title": "Unique identifier for HubitatPollerCac object articulated by the hubitat.
↪poller.cac.gt.000 type.",
      "required": true
    },
    "DisplayName": {
      "type": "string",
      "description": "Sample: Downstairs Thermostat",
      "required": false
    },
    "HwUid": {
      "type": "string",
      "description": "Unique Hardware Identifier",
      "required": false
    },
    "Poller": {
      "type": "dictDict",
      "description": "Includes hubitat_component_id (str), device_id (int), enabled␣
↪(bool), poll_period_s (int) and attributes.",
      "required": true
    },
    "TypeName": {
      "type": "string",
      "value": "hubitat.poller.component.gt",
      "title": "The type name"
    },
    "Version": {
      "type": "string",
      "title": "The type version",
      "default": "000",
      "required": true
    }
  },
  "formats": {
    "UuidCanonicalTextual": {
      "type": "string",
      "description": "A string of hex words separated by hyphens of length 8-4-4-4-12.",
      "example": "652ba6b0-c3bf-4f06-8a80-6b9832d60a25"
    }
  }
}
```

1.3.25 hubitat.tank.cac.gt.000


```
{
  "gtr_asl": "001",
  "type_name": "hubitat.tank.cac.gt",
  "version": "000",
  "owner": "gridworks@gridworks-consulting.com",
  "description": "Hubitat Tank Component Attribute Class (GridWorks Type). A class of␣
↪MakeModels for GridWorks tank temp sensor modules that use Hubitat hubs as part of the␣
↪data collection.",
  "url": "https://gridworks-protocol.readthedocs.io/en/latest/gridworks-tank-module-1.
↪html",
  "properties": {
    "ComponentAttributeClassId": {
      "type": "string",
      "format": "UuidCanonicalTextual",
      "title": "ComponentAttributeClassId",
      "description": "Unique identifier for the device class (aka 'cac' or Component␣
↪Attribute Class). Authority is maintained by the World Registry.",
      "required": true
    },
    "DisplayName": {
      "type": "string",
      "description": "Sample: GridWorks TankModule1",
      "required": false
    },
    "TypeName": {
      "type": "string",
      "value": "hubitat.tank.cac.gt",
      "title": "The type name"
    },
    "Version": {
      "type": "string",
      "title": "The type version",
      "default": "000",
      "required": true
    }
  },
  "example": {
    "ComponentAttributeClassId": "60ac199d-679a-49f7-9142-8ca3e6428a5f",
    "DisplayName": "GridWorks TankModule1",
    "TypeName": "hubitat.tank.cac.gt",
    "Version": "000"
  }
}
```

1.3.26 hubitat.tank.component.gt.000


```
{
  "gtr_asl": "001",
  "type_name": "hubitat.tank.component.gt",
  "version": "000",
  "owner": "gridworks@gridworks-consulting.com",
  "description": "Hubitat Tank Component (GridWorks Type). This is a specific instance␣
↪of a GridWorks temp sensing Tank Module that uses a Hubitat to read the remote data.␣
↪It extends the component.gt.000 type. Authority for the attributes of the component.gt.
↪000 (ComponentId, ComponentAttributeClassId, DisplayName, HwUid) belongs to the␣
↪WorldRegistry. The WorldRegistry is part of the GridWorks 'BackOffice' structure for␣
↪managing relational device data . Notably, ComponentId and ComponentAttributeClass are␣
↪both required and immutable. HwUid is optional but once it is set to a non-null value␣
↪that is also immutable - it is meant to be an immutable identifier associated to a␣
↪specific physical device, ideally one that can be read remotely by the SCADA and also␣
↪by the naked eye. The DisplayName is mutable, with its current value in time governed␣
↪by the WorldRegistry.",
  "url": "https://gridworks-protocol.readthedocs.io/en/latest/gridworks-tank-module-1.
↪html",
  "properties": {
    "ComponentId": {
      "type": "string",
      "format": "UuidCanonicalTextual",
      "title": "Component Id",
      "description": "Primary GridWorks identifier for a specific physical instance of a␣
↪GridWorks TankModule1 and also as a more generic Component.",
      "required": true
    },
    "ComponentAttributeClassId": {
      "type": "string",
      "format": "UuidCanonicalTextual",
      "title": "Unique identifier for HubitatTankCac object articulated by the hubitat.
↪tank.cac.gt.000 type.",
      "description": "Unique identifier for the device class. Authority for these, as␣
↪well as the relationship between Components and ComponentAttributeClasses (Cacs) is␣
↪maintained by the World Registry.",
      "required": true
    },
    "Tank": {
      "type": "dictDict",
      "description": "The configuration information (HubitatTankSettingsGt) about the 4␣
↪analog temperature sensors for a GridWorks TankModule1.",
      "required": true
    },
    "DisplayName": {
      "type": "string",
      "description": "Sample: GridWorks TankModule <buffer> SN 1010",
      "required": false
    },
    "HwUid": {
      "type": "string",
      "description": "Use the GridWorks Serial number for GridWorks TankModule1.",
```


(continues on next page)

(continued from previous page)


```
      "required": false
    },
    "TypeName": {
      "type": "string",
      "value": "hubitat.tank.component.gt",
      "title": "The type name"
    },
    "Version": {
      "type": "string",
      "title": "The type version",
      "default": "000",
      "required": true
    }
  },
  "example": {
    "ComponentAttributeClassId": "60ac199d-679a-49f7-9142-8ca3e6428a5f",
    "ComponentId": "f26d412b-3918-427c-9bb9-cb17b7f2e7e4",
    "DisplayName": "Oak Tank Module <buffer>  SN 1010",
    "Tank": {
      "devices": [
        {
          "analog_input_id": 1,
          "device_id": 103,
          "enabled": true,
          "exponent": 1,
          "fibaro_component_id": "1fdd40dd-14d7-4da2-8cf8-7cf66484e385",
          "rest": null,
          "stack_depth": 1,
          "tank_label": "1010 A1 (Thermistor #1 TANK TOP)",
          "telemetry_name_gt_enum_symbol": "c89d0ba1",
          "temp_unit_gt_enum_symbol": "ec14bd47"
        },
        {
          "analog_input_id": 2,
          "device_id": 104,
          "enabled": true,
          "exponent": 1,
          "fibaro_component_id": "1fdd40dd-14d7-4da2-8cf8-7cf66484e385",
          "rest": null,
          "stack_depth": 2,
          "tank_label": "1010 A2 (Thermistor #2)",
          "telemetry_name_gt_enum_symbol": "c89d0ba1",
          "temp_unit_gt_enum_symbol": "ec14bd47"
        },
        {
          "analog_input_id": 1,
          "device_id": 24,
          "enabled": true,
          "exponent": 1,
          "fibaro_component_id": "a6241764-329d-462f-94f9-0283f707d195",
          "rest": null,
          "stack_depth": 3,
```


(continues on next page)

(continued from previous page)


```
          "tank_label": "1010 B1 (Thermistor #3)",
          "telemetry_name_gt_enum_symbol": "c89d0ba1",
          "temp_unit_gt_enum_symbol": "ec14bd47"
        },
        {
          "analog_input_id": 2,
          "device_id": 25,
          "enabled": true,
          "exponent": 1,
          "fibaro_component_id": "a6241764-329d-462f-94f9-0283f707d195",
          "rest": null,
          "stack_depth": 4,
          "tank_label": "1010 B2 (Thermistor #4 TANK BOTTOM)",
          "telemetry_name_gt_enum_symbol": "c89d0ba1",
          "temp_unit_gt_enum_symbol": "ec14bd47"
        }
      ],
      "hubitat_component_id": "48039704-7d45-4937-adda-0e362d13cef6",
      "sensor_supply_voltage": 23.7
    },
    "TypeName": "hubitat.tank.component.gt",
    "Version": "000"
  }
}
```


1.3.27 multipurpose.sensor.cac.gt.000


```
{
  "gtr_asl": "001",
  "type_name": "multipurpose.sensor.cac.gt",
  "version": "000",
  "owner": "gridworks@gridworks-consulting.com",
  "description": "Type for tracking Multipuprose Sensor ComponentAttributeClasses.␣
↪GridWorks Spaceheat SCADA uses the GridWorks GNodeRegistry structures and abstractions␣
↪for managing relational device data. The Cac, or ComponentAttributeClass, is part of␣
↪this structure.",
  "url": "https://g-node-registry.readthedocs.io/en/latest/component-attribute-class.html
↪",
  "properties": {
    "ComponentAttributeClassId": {
      "type": "string",
      "format": "UuidCanonicalTextual",
      "title": "ComponentAttributeClassId",
      "description": "Unique identifier for the device class (aka 'cac' or Component␣
↪Attribute Class). Authority is maintained by the World Registry.",
      "required": true
    },
    "MakeModel": {
      "type": "string",
      "format": "spaceheat.make.model",
```


(continues on next page)

(continued from previous page)

```
      "title": "MakeModel",
      "description": "Meant to be enough to articulate any difference in how GridWorks␣
↪code would interact with a device. Should be able to use this information to buy or␣
↪build a device.",
      "required": true
    },
    "PollPeriodMs": {
      "type": "integer",
      "description": "Poll Period refers to the period of time between two readings by␣
↪the local actor. This is in contrast to Capture Period, which refers to the period␣
↪between readings that are sent up to the cloud (or otherwise saved for the long-term).
↪",
      "required": true
    },
    "Exponent": {
      "type": "integer",
      "description": "Say the TelemetryName is WaterTempCTimes1000; this corresponds to␣
↪units of Celsius. To match the implication in the name, the Exponent should be 3, and␣
↪a Value of 65300 would indicate 65.3 deg C",
      "required": true
    },
    "TempUnit": {
      "type": "string",
      "format": "spaceheat.unit",
      "title": "Temp Unit",
      "required": true
    },
    "TelemetryNameList": {
      "type": "array",
      "items": {
        "type": "string"
      },
      "format": "spaceheat.telemetry.name",
      "title": "",
      "required": true
    },
    "MaxThermistors": {
      "type": "integer",
      "description": "The maximum number of temperature sensors this multipurpose sensor␣
↪can read.",
      "required": false
    },
    "DisplayName": {
      "type": "string",
      "description": "Sample: GridWorks TSnap1.0 as 12-channel analog temp sensor",
      "required": false
    },
    "CommsMethod": {
      "type": "string",
      "required": false
    },
    "TypeName": {
```

(continues on next page)

(continued from previous page)

```
      "type": "string",
      "value": "multipurpose.sensor.cac.gt",
      "title": "The type name"
    },
    "Version": {
      "type": "string",
      "title": "The type version",
      "default": "000",
      "required": true
    }
  },
  "formats": {
    "UuidCanonicalTextual": {
      "type": "string",
      "description": "A string of hex words separated by hyphens of length 8-4-4-4-12.",
      "example": "652ba6b0-c3bf-4f06-8a80-6b9832d60a25"
    }
  }
}
```

1.3.28 multipurpose.sensor.component.gt.000

```
{
  "gtr_asl": "001",
  "type_name": "multipurpose.sensor.component.gt",
  "version": "000",
  "owner": "gridworks@gridworks-consulting.com",
  "description": "Type for tracking Multupurpose Sensor Components. This type was first␣
↪designed to work with a 12-channel analog temp sensor built into the first generation␣
↪GridWorks SCADA box (GSCADA 1). It extends the component.gt.000 type. Authority for␣
↪the attributes of the component.gt.000 (ComponentId, ComponentAttributeClassId,␣
↪DisplayName, HwUid) belongs to the WorldRegistry. The WorldRegistry is part of the␣
↪GridWorks 'BackOffice' structure for managing relational device data . Notably,␣
↪ComponentId and ComponentAttributeClass are both required and immutable. HwUid is␣
↪optional but once it is set to a non-null value that is also immutable. The␣
↪DisplayName is mutable, with its current value in time governed by the WorldRegistry.",
  "url": "https://gridworks-protocol.readthedocs.io/en/latest/component.html",
  "properties": {
    "ComponentId": {
      "type": "string",
      "format": "UuidCanonicalTextual",
      "title": "Component Id",
      "description": "Primary GridWorks identifier for a specific physical instance of a␣
↪MultipurposeSensor (perhaps only the 12-channel analog temp sensor), and also as a␣
↪more generic Component.",
      "required": true
    },
    "ComponentAttributeClassId": {
      "type": "string",
      "format": "UuidCanonicalTextual",
```

(continues on next page)

(continued from previous page)

```
      "title": "Unique identifier for MultipurposeSensorCac object articulated by the␣
↪multipurpose.sensor.cac.gt.000 type.",
      "description": "Unique identifier for the device class. Authority for these, as␣
↪well as the relationship between Components and ComponentAttributeClasses (Cacs) is␣
↪maintained by the World Registry.",
      "required": true
    },
    "ChannelList": {
      "type": "array",
      "items": {
        "type": "integer"
      },
      "required": true
    },
    "ConfigList": {
      "type": "array",
      "items": {
        "type": "telemetry.reporting.config.000"
      },
      "required": true
    },
    "HwUid": {
      "type": "string",
      "required": false
    },
    "DisplayName": {
      "type": "string",
      "description": "Sample: Oak Multipurpose Temp Sensor Component <100>",
      "required": false
    },
    "TypeName": {
      "type": "string",
      "value": "multipurpose.sensor.component.gt",
      "title": "The type name"
    },
    "Version": {
      "type": "string",
      "title": "The type version",
      "default": "000",
      "required": true
    }
  },
  "example": {
    "ChannelList": [1, 2],
    "ComponentAttributeClassId": "432073b8-4d2b-4e36-9229-73893f33f846",
    "ComponentId": "109e0bde-2f04-4cd4-9e69-bb2732a368e2",
    "ConfigList": [
      {
        "AboutNodeName": "a.dist.swt.temp",
        "AsyncReportThreshold": 0.005,
        "Exponent": 3,
        "NameplateMaxValue": 100000,
```

(continues on next page)

(continued from previous page)

```
        "ReportOnChange": true,
        "SamplePeriodS": 60,
        "TelemetryNameGtEnumSymbol": "c89d0ba1",
        "TypeName": "telemetry.reporting.config",
        "UnitGtEnumSymbol": "ec14bd47",
        "Version": "000"
      },
      {
        "AboutNodeName": "a.dist.rwt.temp",
        "AsyncReportThreshold": 0.005,
        "Exponent": 3,
        "NameplateMaxValue": 100000,
        "ReportOnChange": true,
        "SamplePeriodS": 60,
        "TelemetryNameGtEnumSymbol": "c89d0ba1",
        "TypeName": "telemetry.reporting.config",
        "UnitGtEnumSymbol": "ec14bd47",
        "Version": "000"
      }
    ],
    "DisplayName": "Multipurpose Temp Sensor Component <100>",
    "HwUid": "100",
    "TypeName": "multipurpose.sensor.component.gt",
    "Version": "000"
  },
  "formats": {
    "UuidCanonicalTextual": {
      "type": "string",
      "description": "A string of hex words separated by hyphens of length 8-4-4-4-12.",
      "example": "652ba6b0-c3bf-4f06-8a80-6b9832d60a25"
    },
    "LeftRightDot": {
      "type": "string",
      "description": "Lowercase alphanumeric words separated by periods, with the most␣
↪significant word (on the left) starting with an alphabet character.",
      "example": "dw1.isone.me.freedom.apple"
    }
  }
}
```

1.3.29 pipe.flow.sensor.cac.gt.000

```
{
  "gtr_asl": "001",
  "type_name": "pipe.flow.sensor.cac.gt",
  "version": "000",
  "owner": "gridworks@gridworks-consulting.com",
  "description": "Type for tracking Pipe Flow Sensor ComponentAttributeClasses.␣
↪GridWorks Spaceheat SCADA uses the GridWorks GNodeRegistry structures and abstractions␣
↪for managing relational device data. The Cac, or ComponentAttributeClass, is part of␣
```

(continues on next page)

(continued from previous page)

```
↪this structure.",
  "url": "https://g-node-registry.readthedocs.io/en/latest/component-attribute-class.html
↪",
  "properties": {
    "ComponentAttributeClassId": {
      "type": "string",
      "format": "UuidCanonicalTextual",
      "title": "ComponentAttributeClassId",
      "description": "Unique identifier for the device class (aka 'cac' or Component␣
↪Attribute Class). Authority is maintained by the World Registry.",
      "required": true
    },
    "MakeModel": {
      "type": "string",
      "format": "spaceheat.make.model",
      "title": "",
      "required": true
    },
    "DisplayName": {
      "type": "string",
      "description": "Sample: Atlas Scientific EZO FLO i2c",
      "required": false
    },
    "CommsMethod": {
      "type": "string",
      "required": false
    },
    "TypeName": {
      "type": "string",
      "value": "pipe.flow.sensor.cac.gt",
      "title": "The type name"
    },
    "Version": {
      "type": "string",
      "title": "The type version",
      "default": "000",
      "required": true
    }
  },
  "formats": {
    "UuidCanonicalTextual": {
      "type": "string",
      "description": "A string of hex words separated by hyphens of length 8-4-4-4-12.",
      "example": "652ba6b0-c3bf-4f06-8a80-6b9832d60a25"
    }
  }
}
```

1.3.30 pipe.flow.sensor.component.gt.000


```
{
  "gtr_asl": "001",
  "type_name": "pipe.flow.sensor.component.gt",
  "version": "000",
  "owner": "gridworks@gridworks-consulting.com",
  "description": "Type for tracking Pipe Flow Sensor Components. Designed for Pipe Flow␣
↪Sensors. It extends the component.gt.000 type. Authority for the attributes of the␣
↪component.gt.000 (ComponentId, ComponentAttributeClassId, DisplayName, HwUid) belongs␣
↪to the WorldRegistry. The WorldRegistry is part of the GridWorks 'BackOffice'␣
↪structure for managing relational device data . Notably, ComponentId and␣
↪ComponentAttributeClass are both required and immutable. HwUid is optional but once it␣
↪is set to a non-null value that is also immutable - it is meant to be an immutable␣
↪identifier associated to a specific physical device, ideally one that can be read␣
↪remotely by the SCADA and also by the naked eye. The DisplayName is mutable, with its␣
↪current value in time governed by the WorldRegistry.",
  "url": "https://gridworks-protocol.readthedocs.io/en/latest/component.html",
  "properties": {
    "ComponentId": {
      "type": "string",
      "format": "UuidCanonicalTextual",
      "title": "Component Id",
      "description": "Primary GridWorks identifier for a specific physical instance of a␣
↪PipeFlowSensor, and also as a more generic Component.",
      "required": true
    },
    "ComponentAttributeClassId": {
      "type": "string",
      "format": "UuidCanonicalTextual",
      "title": "Unique identifier for PipeFlowSensorCac object articulated by the pipe.
↪flow.sensor.cac.gt.000 type.",
      "description": "Unique identifier for the device class. Authority for these, as␣
↪well as the relationship between Components and ComponentAttributeClasses (Cacs) is␣
↪maintained by the World Registry.",
      "required": true
    },
    "I2cAddress": {
      "type": "integer",
      "required": true
    },
    "ConversionFactor": {
      "type": "number",
      "required": true
    },
    "DisplayName": {
      "type": "string",
      "description": "Sample: Pipe Flow Meter Component <dist-flow>",
      "required": false
    },
    "HwUid": {
      "type": "string",
      "required": false
```


(continues on next page)

(continued from previous page)

```
    },
    "ExpectedMaxGpmTimes100": {
      "type": "integer",
      "required": false
    },
    "TypeName": {
      "type": "string",
      "value": "pipe.flow.sensor.component.gt",
      "title": "The type name"
    },
    "Version": {
      "type": "string",
      "title": "The type version",
      "default": "000",
      "required": true
    }
  },
  "formats": {
    "UuidCanonicalTextual": {
      "type": "string",
      "description": "A string of hex words separated by hyphens of length 8-4-4-4-12.",
      "example": "652ba6b0-c3bf-4f06-8a80-6b9832d60a25"
    }
  }
}
```

1.3.31 power.watts.000

```
{
  "gtr_asl": "001",
  "type_name": "power.watts",
  "version": "000",
  "owner": "gridworks@gridworks-consulting.com",
  "description": "Real-time power of TerminalAsset in Watts. Used by a SCADA -> Atn or␣
↪Atn -> AggregatedTNode to report real-time power of their TerminalAsset. Positive␣
↪number means WITHDRAWAL from the grid - so generating electricity creates a negative␣
↪number. This message is considered worse than useless to send after the first attempt,␣
↪and does not require an ack. Shares the same purpose as gs.pwr, but is not designed to␣
↪minimize bytes so comes in JSON format.",
  "properties": {
    "Watts": {
      "type": "integer",
      "required": true
    },
    "TypeName": {
      "type": "string",
      "value": "power.watts",
      "title": "The type name"
    },
    "Version": {
```

(continues on next page)

(continued from previous page)

```
      "type": "string",
      "title": "The type version",
      "default": "000",
      "required": true
    }
  }
}
```

1.3.32 relay.cac.gt.000

```
{
  "gtr_asl": "001",
  "type_name": "relay.cac.gt",
  "version": "000",
  "owner": "gridworks@gridworks-consulting.com",
  "description": "Type for tracking Relay ComponentAttributeClasses. GridWorks Spaceheat␣
↪SCADA uses the GridWorks GNodeRegistry structures and abstractions for managing␣
↪relational device data. The Cac, or ComponentAttributeClass, is part of this structure.
↪",
  "url": "https://g-node-registry.readthedocs.io/en/latest/component-attribute-class.html
↪",
  "properties": {
    "ComponentAttributeClassId": {
      "type": "string",
      "format": "UuidCanonicalTextual",
      "title": "ComponentAttributeClassId",
      "description": "Unique identifier for the device class (aka 'cac' or Component␣
↪Attribute Class). Authority is maintained by the World Registry.",
      "required": true
    },
    "MakeModel": {
      "type": "string",
      "format": "spaceheat.make.model",
      "title": "",
      "required": true
    },
    "DisplayName": {
      "type": "string",
      "required": false
    },
    "TypicalResponseTimeMs": {
      "type": "integer",
      "required": true
    },
    "TypeName": {
      "type": "string",
      "value": "relay.cac.gt",
      "title": "The type name"
    },
    "Version": {
```

(continues on next page)

(continued from previous page)

```
      "type": "string",
      "title": "The type version",
      "default": "000",
      "required": true
    }
  },
  "formats": {
    "UuidCanonicalTextual": {
      "type": "string",
      "description": "A string of hex words separated by hyphens of length 8-4-4-4-12.",
      "example": "652ba6b0-c3bf-4f06-8a80-6b9832d60a25"
    }
  }
}
```

1.3.33 relay.component.gt.000

```
{
  "gtr_asl": "001",
  "type_name": "relay.component.gt",
  "version": "000",
  "owner": "gridworks@gridworks-consulting.com",
  "description": "Type for tracking Relay Components. Designed for Relays. It extends␣
↪the component.gt.000 type. Authority for the attributes of the component.gt.000␣
↪(ComponentId, ComponentAttributeClassId, DisplayName, HwUid) belongs to the␣
↪WorldRegistry. The WorldRegistry is part of the GridWorks 'BackOffice' structure for␣
↪managing relational device data . Notably, ComponentId and ComponentAttributeClass are␣
↪both required and immutable. HwUid is optional but once it is set to a non-null value␣
↪that is also immutable - it is meant to be an immutable identifier associated to a␣
↪specific physical device, ideally one that can be read remotely by the SCADA and also␣
↪by the naked eye. The DisplayName is mutable, with its current value in time governed␣
↪by the WorldRegistry.",
  "url": "https://g-node-registry.readthedocs.io/en/latest/component.html",
  "properties": {
    "ComponentId": {
      "type": "string",
      "format": "UuidCanonicalTextual",
      "title": "Component Id",
      "description": "Primary GridWorks identifier for a specific physical instance of a␣
↪Relay, and also as a more generic Component.",
      "required": true
    },
    "ComponentAttributeClassId": {
      "type": "string",
      "format": "UuidCanonicalTextual",
      "title": "Unique identifier for RelayCac object articulated by the relay.cac.gt.
↪000 type.",
      "description": "Unique identifier for the device class. Authority for these, as␣
↪well as the relationship between Components and ComponentAttributeClasses (Cacs) is␣
↪maintained by the World Registry.",
```

(continues on next page)

(continued from previous page)

```
      "required": true
    },
    "DisplayName": {
      "type": "string",
      "required": false
    },
    "Gpio": {
      "type": "integer",
      "required": false
    },
    "HwUid": {
      "type": "string",
      "required": false
    },
    "NormallyOpen": {
      "type": "boolean",
      "description": "Normally open relays default in the open position, meaning that␣
↪when they're not in use, there is no contact between the circuits. When power is␣
↪introduced, an electromagnet pulls the first circuit into contact with the second,␣
↪thereby closing the circuit and allowing power to flow through",
      "required": true
    },
    "TypeName": {
      "type": "string",
      "value": "relay.component.gt",
      "title": "The type name"
    },
    "Version": {
      "type": "string",
      "title": "The type version",
      "default": "000",
      "required": true
    }
  },
  "formats": {
    "UuidCanonicalTextual": {
      "type": "string",
      "description": "A string of hex words separated by hyphens of length 8-4-4-4-12.",
      "example": "652ba6b0-c3bf-4f06-8a80-6b9832d60a25"
    }
  }
}
```

1.3.34 resistive.heater.cac.gt.000


```
{
  "gtr_asl": "001",
  "type_name": "resistive.heater.cac.gt",
  "version": "000",
  "owner": "gridworks@gridworks-consulting.com",
  "description": "Type for tracking Resistive Heater ComponentAttributeClasses.␣
↪GridWorks Spaceheat SCADA uses the GridWorks GNodeRegistry structures and abstractions␣
↪for managing relational device data. The Cac, or ComponentAttributeClass, is part of␣
↪this structure.",
  "url": "https://g-node-registry.readthedocs.io/en/latest/component-attribute-class.html
↪",
  "properties": {
    "ComponentAttributeClassId": {
      "type": "string",
      "format": "UuidCanonicalTextual",
      "title": "ComponentAttributeClassId",
      "description": "Unique identifier for the device class (aka 'cac' or Component␣
↪Attribute Class). Authority is maintained by the World Registry.",
      "required": true
    },
    "MakeModel": {
      "type": "string",
      "format": "spaceheat.make.model",
      "title": "",
      "required": true
    },
    "DisplayName": {
      "type": "string",
      "required": false
    },
    "NameplateMaxPowerW": {
      "type": "integer",
      "required": true
    },
    "RatedVoltageV": {
      "type": "integer",
      "format": "PositiveInteger",
      "title": "",
      "required": true
    },
    "TypeName": {
      "type": "string",
      "value": "resistive.heater.cac.gt",
      "title": "The type name"
    },
    "Version": {
      "type": "string",
      "title": "The type version",
      "default": "000",
      "required": true
    }
```


(continues on next page)

(continued from previous page)

```
  },
  "formats": {
    "UuidCanonicalTextual": {
      "type": "string",
      "description": "A string of hex words separated by hyphens of length 8-4-4-4-12.",
      "example": "652ba6b0-c3bf-4f06-8a80-6b9832d60a25"
    },
    "PositiveInteger": {
      "type": "string",
      "description": "Must be positive when interpreted as an integer. Interpretation as␣
↪an integer follows the pydantic rules for this - which will round down rational␣
↪numbers. So 1.7 will be interpreted as 1 and is also fine, while 0.5 is interpreted as␣
↪0 and will raise an exception.",
      "example": ""
    }
  }
}
```

1.3.35 resistive.heater.component.gt.000

```
{
  "gtr_asl": "001",
  "type_name": "resistive.heater.component.gt",
  "version": "000",
  "owner": "gridworks@gridworks-consulting.com",
  "description": "Type for tracking Resistive Heater Components. Designed for Resistive␣
↪Heaters. It extends the component.gt.000 type. Authority for the attributes of the␣
↪component.gt.000 (ComponentId, ComponentAttributeClassId, DisplayName, HwUid) belongs␣
↪to the WorldRegistry. The WorldRegistry is part of the GridWorks 'BackOffice'␣
↪structure for managing relational device data . Notably, ComponentId and␣
↪ComponentAttributeClass are both required and immutable. HwUid is optional but once it␣
↪is set to a non-null value that is also immutable - it is meant to be an immutable␣
↪identifier associated to a specific physical device, ideally one that can be read␣
↪remotely by the SCADA and also by the naked eye. The DisplayName is mutable, with its␣
↪current value in time governed by the WorldRegistry.",
  "url": "https://g-node-registry.readthedocs.io/en/latest/component.html",
  "properties": {
    "ComponentId": {
      "type": "string",
      "format": "UuidCanonicalTextual",
      "title": "Component Id",
      "description": "Primary GridWorks identifier for a specific physical instance of a␣
↪ResistiveHeater, and also as a more generic Component.",
      "required": true
    },
    "ComponentAttributeClassId": {
      "type": "string",
      "format": "UuidCanonicalTextual",
      "title": "Unique identifier for ResistiveHeaterCac object articulated by the␣
↪resistive.heater.cac.gt.000 type.",
```

(continues on next page)

(continued from previous page)

```
      "description": "Unique identifier for the device class. Authority for these, as␣
↪well as the relationship between Components and ComponentAttributeClasses (Cacs) is␣
↪maintained by the World Registry.",
      "required": true
    },
    "DisplayName": {
      "type": "string",
      "required": false
    },
    "HwUid": {
      "type": "string",
      "required": false
    },
    "TestedMaxHotMilliOhms": {
      "type": "integer",
      "required": false
    },
    "TestedMaxColdMilliOhms": {
      "type": "integer",
      "required": false
    },
    "TypeName": {
      "type": "string",
      "value": "resistive.heater.component.gt",
      "title": "The type name"
    },
    "Version": {
      "type": "string",
      "title": "The type version",
      "default": "000",
      "required": true
    }
  },
  "formats": {
    "UuidCanonicalTextual": {
      "type": "string",
      "description": "A string of hex words separated by hyphens of length 8-4-4-4-12.",
      "example": "652ba6b0-c3bf-4f06-8a80-6b9832d60a25"
    }
  }
}
```

1.3.36 rest.poller.cac.gt.000


```
{
  "gtr_asl": "001",
  "type_name": "rest.poller.cac.gt",
  "version": "000",
  "owner": "gridworks@gridworks-consulting.com",
  "description": "REST Poller Cac (GridWorks Type). Type for devices that can be polled␣
↪by SCADA software via a REST endpoint on the LAN. For example a HoneyWell ZWave␣
↪thermostat that can queried via a IoT hub like a Hubitat C7.",
  "properties": {
    "ComponentAttributeClassId": {
      "type": "string",
      "format": "UuidCanonicalTextual",
      "title": "ComponentAttributeClassId",
      "description": "Unique identifier for the device class (aka 'cac' or Component␣
↪Attribute Class). Authority is maintained by the World Registry.",
      "required": true
    },
    "DisplayName": {
      "type": "string",
      "required": false
    },
    "TypeName": {
      "type": "string",
      "value": "rest.poller.cac.gt",
      "title": "The type name"
    },
    "Version": {
      "type": "string",
      "title": "The type version",
      "default": "000",
      "required": true
    }
  }
}
```


1.3.37 rest.poller.component.gt.000


```
{
  "gtr_asl": "001",
  "type_name": "rest.poller.component.gt",
  "version": "000",
  "owner": "gridworks@gridworks-consulting.com",
  "description": "REST Poller Component (GridWorks Type). Designed for generic REST␣
↪Pollers (like Honeywell Z-Wave thermostats). It extends the component.gt.000 type.␣
↪Authority for the attributes of the component.gt.000 (ComponentId,␣
↪ComponentAttributeClassId, DisplayName, HwUid) belongs to the WorldRegistry. The␣
↪WorldRegistry is part of the GridWorks 'BackOffice' structure for managing relational␣
↪device data . Notably, ComponentId and ComponentAttributeClass are both required and␣
↪immutable. HwUid is optional but once it is set to a non-null value that is also␣
```


(continues on next page)

(continued from previous page)

```
↪immutable - it is meant to be an immutable identifier associated to a specific␣
↪physical device, ideally one that can be read remotely by the SCADA and also by the␣
↪naked eye. The DisplayName is mutable, with its current value in time governed by the␣
↪WorldRegistry.",
  "url": "https://g-node-registry.readthedocs.io/en/latest/component.html",
  "properties": {
    "TypeName": {
      "type": "string",
      "value": "rest.poller.component.gt",
      "title": "The type name"
    },
    "Version": {
      "type": "string",
      "title": "The type version",
      "default": "000",
      "required": true
    }
  }
}
```

1.3.38 simple.temp.sensor.cac.gt.000

```
{
  "gtr_asl": "001",
  "type_name": "simple.temp.sensor.cac.gt",
  "version": "000",
  "owner": "gridworks@gridworks-consulting.com",
  "description": "Type for tracking Simple Temp Sensor ComponentAttributeClasses.␣
↪GridWorks Spaceheat SCADA uses the GridWorks GNodeRegistry structures and abstractions␣
↪for managing relational device data. The Cac, or ComponentAttributeClass, is part of␣
↪this structure.",
  "url": "https://g-node-registry.readthedocs.io/en/latest/component-attribute-class.html
↪",
  "properties": {
    "ComponentAttributeClassId": {
      "type": "string",
      "format": "UuidCanonicalTextual",
      "title": "ComponentAttributeClassId",
      "description": "Unique identifier for the device class (aka 'cac' or Component␣
↪Attribute Class). Authority is maintained by the World Registry.",
      "required": true
    },
    "MakeModel": {
      "type": "string",
      "format": "spaceheat.make.model",
      "title": "",
      "required": true
    },
    "TypicalResponseTimeMs": {
      "type": "integer",
```

(continues on next page)

(continued from previous page)

```
      "required": true
    },
    "Exponent": {
      "type": "integer",
      "description": "Say the TelemetryName is WaterTempCTimes1000; this corresponds to␣
↪units of Celsius. To match the implication in the name, the Exponent should be 3, and␣
↪a Value of 65300 would indicate 65.3 deg C",
      "required": true
    },
    "TempUnit": {
      "type": "string",
      "format": "spaceheat.unit",
      "title": "",
      "required": true
    },
    "TelemetryName": {
      "type": "string",
      "format": "spaceheat.telemetry.name",
      "title": "",
      "required": true
    },
    "DisplayName": {
      "type": "string",
      "required": false
    },
    "CommsMethod": {
      "type": "string",
      "required": false
    },
    "TypeName": {
      "type": "string",
      "value": "simple.temp.sensor.cac.gt",
      "title": "The type name"
    },
    "Version": {
      "type": "string",
      "title": "The type version",
      "default": "000",
      "required": true
    }
  },
  "formats": {
    "UuidCanonicalTextual": {
      "type": "string",
      "description": "A string of hex words separated by hyphens of length 8-4-4-4-12.",
      "example": "652ba6b0-c3bf-4f06-8a80-6b9832d60a25"
    }
  }
}
```

1.3.39 simple.temp.sensor.component.gt.000


```
{
  "gtr_asl": "001",
  "type_name": "simple.temp.sensor.component.gt",
  "version": "000",
  "owner": "gridworks@gridworks-consulting.com",
  "description": "Type for tracking Simple Temp Sensor Components. Designed for simple␣
↪temp sensors that read only one temp. It extends the component.gt.000 type. Authority␣
↪for the attributes of the component.gt.000 (ComponentId, ComponentAttributeClassId,␣
↪DisplayName, HwUid) belongs to the WorldRegistry. The WorldRegistry is part of the␣
↪GridWorks 'BackOffice' structure for managing relational device data . Notably,␣
↪ComponentId and ComponentAttributeClass are both required and immutable. HwUid is␣
↪optional but once it is set to a non-null value that is also immutable - it is meant␣
↪to be an immutable identifier associated to a specific physical device, ideally one␣
↪that can be read remotely by the SCADA and also by the naked eye. The DisplayName is␣
↪mutable, with its current value in time governed by the WorldRegistry.",
  "url": "https://g-node-registry.readthedocs.io/en/latest/component.html",
  "properties": {
    "ComponentId": {
      "type": "string",
      "format": "UuidCanonicalTextual",
      "title": "Component Id",
      "description": "Primary GridWorks identifier for a specific physical instance of a␣
↪SimpleTempSensor, and also as a more generic Component.",
      "required": true
    },
    "ComponentAttributeClassId": {
      "type": "string",
      "format": "UuidCanonicalTextual",
      "title": "Unique identifier for SimpleTempSensorCac object articulated by the␣
↪simple.temp.sensor.cac.gt.000 type.",
      "description": "Unique identifier for the device class. Authority for these, as␣
↪well as the relationship between Components and ComponentAttributeClasses (Cacs) is␣
↪maintained by the World Registry.",
      "required": true
    },
    "DisplayName": {
      "type": "string",
      "required": false
    },
    "HwUid": {
      "type": "string",
      "required": false
    },
    "Channel": {
      "type": "integer",
      "required": false
    },
    "TypeName": {
      "type": "string",
      "value": "simple.temp.sensor.component.gt",
      "title": "The type name"
```


(continues on next page)

(continued from previous page)

```
    },
    "Version": {
      "type": "string",
      "title": "The type version",
      "default": "000",
      "required": true
    }
  },
  "formats": {
    "UuidCanonicalTextual": {
      "type": "string",
      "description": "A string of hex words separated by hyphens of length 8-4-4-4-12.",
      "example": "652ba6b0-c3bf-4f06-8a80-6b9832d60a25"
    }
  }
}
```

1.3.40 snapshot.spaceheat.000

```
{
  "gtr_asl": "001",
  "type_name": "snapshot.spaceheat",
  "version": "000",
  "owner": "gridworks@gridworks-consulting.com",
  "description": "",
  "properties": {
    "FromGNodeAlias": {
      "type": "string",
      "format": "LeftRightDot",
      "title": "",
      "required": true
    },
    "FromGNodeInstanceId": {
      "type": "string",
      "format": "UuidCanonicalTextual",
      "title": "",
      "required": true
    },
    "Snapshot": {
      "type": "telemetry.snapshot.spaceheat.000",
      "required": true
    },
    "TypeName": {
      "type": "string",
      "value": "snapshot.spaceheat",
      "title": "The type name"
    },
    "Version": {
      "type": "string",
      "title": "The type version",
```

(continues on next page)

(continued from previous page)

```
      "default": "000",
      "required": true
    }
  },
  "formats": {
    "UuidCanonicalTextual": {
      "type": "string",
      "description": "A string of hex words separated by hyphens of length 8-4-4-4-12.",
      "example": "652ba6b0-c3bf-4f06-8a80-6b9832d60a25"
    },
    "LeftRightDot": {
      "type": "string",
      "description": "Lowercase alphanumeric words separated by periods, with the most␣
↪significant word (on the left) starting with an alphabet character.",
      "example": "dw1.isone.me.freedom.apple"
    }
  }
}
```

1.3.41 spaceheat.node.gt.100

```
{
  "gtr_asl": "001",
  "type_name": "spaceheat.node.gt",
  "version": "100",
  "owner": "gridworks@gridworks-consulting.com",
  "description": "Spaceheat Node. A SpaceheatNode, or ShNode, is an organizing principal␣
↪for the SCADA software. ShNodes can represent both underlying physical objects (water␣
↪tank), measurements of these objects (temperature sensing at the top of a water tank),␣
↪and actors within the code (an actor measuring multiple temperatures, or an actor␣
↪responsible for filtering/smoothing temperature data for the purposes of thermostatic␣
↪control).",
  "url": "https://gridworks-protocol.readthedocs.io/en/latest/spaceheat-node.html",
  "properties": {
    "ShNodeId": {
      "type": "string",
      "format": "UuidCanonicalTextual",
      "title": "",
      "required": true
    },
    "Alias": {
      "type": "string",
      "format": "LeftRightDot",
      "title": "",
      "required": true
    },
    "ActorClass": {
      "type": "string",
      "format": "sh.actor.class",
      "title": "",
```

(continues on next page)

(continued from previous page)

```
      "required": true
    },
    "Role": {
      "type": "string",
      "format": "sh.node.role",
      "title": "",
      "required": true
    },
    "DisplayName": {
      "type": "string",
      "required": false
    },
    "ComponentId": {
      "type": "string",
      "format": "UuidCanonicalTextual",
      "title": "Unique identifier for Component object articulated by the component.gt.
↪000 type.",
      "description": "Used if a Spaceheat Node is associated with a physical device.",
      "required": false
    },
    "ReportingSamplePeriodS": {
      "type": "integer",
      "required": false
    },
    "RatedVoltageV": {
      "type": "integer",
      "format": "PositiveInteger",
      "title": "",
      "required": false
    },
    "TypicalVoltageV": {
      "type": "integer",
      "format": "PositiveInteger",
      "title": "",
      "required": false
    },
    "InPowerMetering": {
      "type": "boolean",
      "required": false
    },
    "TypeName": {
      "type": "string",
      "value": "spaceheat.node.gt",
      "title": "The type name"
    },
    "Version": {
      "type": "string",
      "title": "The type version",
      "default": "100",
      "required": true
    }
  },
```

(continues on next page)

(continued from previous page)

```
  "formats": {
    "UuidCanonicalTextual": {
      "type": "string",
      "description": "A string of hex words separated by hyphens of length 8-4-4-4-12.",
      "example": "652ba6b0-c3bf-4f06-8a80-6b9832d60a25"
    },
    "PositiveInteger": {
      "type": "string",
      "description": "Must be positive when interpreted as an integer. Interpretation as␣
↪an integer follows the pydantic rules for this - which will round down rational␣
↪numbers. So 1.7 will be interpreted as 1 and is also fine, while 0.5 is interpreted as␣
↪0 and will raise an exception.",
      "example": ""
    },
    "LeftRightDot": {
      "type": "string",
      "description": "Lowercase alphanumeric words separated by periods, with the most␣
↪significant word (on the left) starting with an alphabet character.",
      "example": "dw1.isone.me.freedom.apple"
    }
  }
}
```

1.3.42 ta.data.channels.000

```
{
  "gtr_asl": "001",
  "type_name": "ta.data.channels",
  "version": "000",
  "owner": "gridworks@gridworks-consulting.com",
  "description": "Terminal Asset Data Channels. A list of data channels associated to a␣
↪specific Terminal Asset.",
  "properties": {
    "TerminalAssetGNodeAlias": {
      "type": "string",
      "format": "LeftRightDot",
      "title": "GNodeAlias for the Terminal Asset",
      "description": "The Alias of the Terminal Asset about which the time series data␣
↪is providing information.",
      "required": true
    },
    "TerminalAssetGNodeId": {
      "type": "string",
      "format": "UuidCanonicalTextual",
      "title": "GNodeId for the Terminal Asset",
      "description": "The immutable unique identifier for the Terminal Asset.",
      "required": true
    },
    "TimeUnixS": {
      "type": "integer",
```

(continues on next page)

(continued from previous page)

```
      "format": "ReasonableUnixTimeS",
      "title": "TimeUnixS",
      "description": "The time that this list of data channels was created",
      "required": true
    },
    "Author": {
      "type": "string",
      "description": "Author of this list of data channels.",
      "required": true
    },
    "Channels": {
      "type": "array",
      "items": {
        "type": "data.channel.000"
      },
      "required": true
    },
    "Identifier": {
      "type": "string",
      "format": "UuidCanonicalTextual",
      "title": "Identifier",
      "description": "Unique identifier for a specific instance of this type that can be␣
↪used to establish how time series csv's were constructed.",
      "required": true
    },
    "TypeName": {
      "type": "string",
      "value": "ta.data.channels",
      "title": "The type name"
    },
    "Version": {
      "type": "string",
      "title": "The type version",
      "default": "000",
      "required": true
    }
  },
  "formats": {
    "ReasonableUnixTimeS": {
      "type": "string",
      "description": "Integer reflecting unix time seconds between 1970 and 3000",
      "example": ""
    },
    "UuidCanonicalTextual": {
      "type": "string",
      "description": "A string of hex words separated by hyphens of length 8-4-4-4-12.",
      "example": "652ba6b0-c3bf-4f06-8a80-6b9832d60a25"
    },
    "LeftRightDot": {
      "type": "string",
      "description": "Lowercase alphanumeric words separated by periods, with the most␣
↪significant word (on the left) starting with an alphabet character.",
```

(continues on next page)

(continued from previous page)

```
      "example": "dw1.isone.me.freedom.apple"
    }
  }
}
```

1.3.43 telemetry.reporting.config.000

```
{
  "gtr_asl": "001",
  "type_name": "telemetry.reporting.config",
  "version": "000",
  "owner": "gridworks@gridworks-consulting.com",
  "description": "",
  "properties": {
    "TelemetryName": {
      "type": "string",
      "format": "spaceheat.telemetry.name",
      "title": "",
      "required": true
    },
    "AboutNodeName": {
      "type": "string",
      "format": "LeftRightDot",
      "title": "",
      "description": "The name of the SpaceheatNode whose physical quantity is getting␣
↪captured.",
      "required": true
    },
    "ReportOnChange": {
      "type": "boolean",
      "required": true
    },
    "SamplePeriodS": {
      "type": "integer",
      "required": true
    },
    "Exponent": {
      "type": "integer",
      "description": "Say the TelemetryName is WaterTempCTimes1000; this corresponds to␣
↪units of Celsius. To match the implication in the name, the Exponent should be 3, and␣
↪a Value of 65300 would indicate 65.3 deg C",
      "required": true
    },
    "Unit": {
      "type": "string",
      "format": "spaceheat.unit",
      "title": "",
      "required": true
    },
    "AsyncReportThreshold": {
```

(continues on next page)

(continued from previous page)

```
      "type": "number",
      "required": false
    },
    "NameplateMaxValue": {
      "type": "integer",
      "format": "PositiveInteger",
      "title": "",
      "required": false
    },
    "TypeName": {
      "type": "string",
      "value": "telemetry.reporting.config",
      "title": "The type name"
    },
    "Version": {
      "type": "string",
      "title": "The type version",
      "default": "000",
      "required": true
    }
  },
  "axioms": {
    "Axiom1": {
      "title": "Async reporting consistency",
      "description": "If AsyncReportThreshold exists, so does NameplateMaxValue"
    }
  },
  "example": {
    "TelemetryNameGtEnumSymbol": "af39eec9",
    "AboutNodeName": "house-panel-power",
    "ReportOnChange": true,
    "SamplePeriodS": 300,
    "Exponent": 0,
    "UnitGtEnumSymbol": "f459a9c3",
    "AsyncReportThreshold": 0.02,
    "NameplateMaxValue": 3500,
    "TypeName": "telemetry.reporting.config",
    "Version": "000"
  },
  "formats": {
    "PositiveInteger": {
      "type": "string",
      "description": "Must be positive when interpreted as an integer. Interpretation as␣
↪an integer follows the pydantic rules for this - which will round down rational␣
↪numbers. So 1.7 will be interpreted as 1 and is also fine, while 0.5 is interpreted as␣
↪0 and will raise an exception.",
      "example": ""
    },
    "LeftRightDot": {
      "type": "string",
      "description": "Lowercase alphanumeric words separated by periods, with the most␣
↪significant word (on the left) starting with an alphabet character.",
```

(continues on next page)

(continued from previous page)

```
      "example": "dw1.isone.me.freedom.apple"
    }
  }
}
```

1.3.44 telemetry.snapshot.spaceheat.000

```
{
  "gtr_asl": "001",
  "type_name": "telemetry.snapshot.spaceheat",
  "version": "000",
  "owner": "gridworks@gridworks-consulting.com",
  "description": "Snapshot of Telemetry Data from a SpaceHeat SCADA. A snapshot of all␣
↪current sensed states, sent from a spaceheat SCADA to its AtomicTNode. The nth element␣
↪of each of the three lists refer to the same reading (i.e., what is getting read, what␣
↪the value is, what the TelemetryNames are.)",
  "url": "https://gridworks-protocol.readthedocs.io/en/latest/spaceheat-node.html",
  "properties": {
    "ReportTimeUnixMs": {
      "type": "integer",
      "format": "ReasonableUnixTimeMs",
      "title": "ReportTimeUnixMs",
      "description": "The time, in unix ms, that the SCADA creates this type. It may not␣
↪be when the SCADA sends the type to the atn (for example if Internet is down).",
      "required": true
    },
    "AboutNodeAliasList": {
      "type": "array",
      "items": {
        "type": "string"
      },
      "format": "LeftRightDot",
      "title": "AboutNodeAliases",
      "description": "The list of Spaceheat nodes in the snapshot.",
      "required": true
    },
    "ValueList": {
      "type": "array",
      "items": {
        "type": "integer"
      },
      "required": true
    },
    "TelemetryNameList": {
      "type": "array",
      "items": {
        "type": "string"
      },
      "format": "spaceheat.telemetry.name",
      "title": "",
```

(continues on next page)

(continued from previous page)

```
      "required": true
    },
    "TypeName": {
      "type": "string",
      "value": "telemetry.snapshot.spaceheat",
      "title": "The type name"
    },
    "Version": {
      "type": "string",
      "title": "The type version",
      "default": "000",
      "required": true
    }
  },
  "axioms": {
    "Axiom1": {
      "title": "ListLengthConsistency",
      "description": "AboutNodeAliastList, ValueList and TelemetryNameList must all have␣
↪the same length."
    }
  },
  "formats": {
    "LeftRightDot": {
      "type": "string",
      "description": "Lowercase alphanumeric words separated by periods, with the most␣
↪significant word (on the left) starting with an alphabet character.",
      "example": "dw1.isone.me.freedom.apple"
    },
    "ReasonableUnixTimeMs": {
      "type": "string",
      "description": "An integer reflecting unix time in MILLISECONDS between midnight␣
↪Jan 1 2000 and midnight Jan 1 3000 UTC",
      "example": "1702327940710"
    }
  }
}
```

1.4 Gwproto Enums

GridWorks Enums used in gwproto, the Application Shared Language (ASL) used by SCADA devices and AtomicTNodes to communicate with each other. These enums play a specific structural role as semantic "glue" within ASLs.

Key attributes:

- Enum values are translated into "GridWorks Type Enum Symbols" (GtEnumSymbols) when embedded

in a serialized type sent as a message from one Application and/or Actor to another. - Each Enum has a unique name in the type registry (like spaceheat.telemetry.name), along with a version (like 001). - That name are interpretted locally in the SDK and do not necessarily carry the larger context of the unique type registry name (for example gwproto uses TelemetryName, since the *spaceheat* context goes without saying). - Each Value/Symbol pair also has a version. Value/Symbol pairs cannot be changed or removed. The only adjustments that can be made to an enum are adding more Value/Symbols. This is to support forwards- and backwards- compatability

in GridWorks Types that use these enums.

If Enums are "glue", then GridWorks Types are the building blocks of SALs. Every SAL is comprised of a set of shared GridWorks Types.

Application Shared Languages are an evolution of the concept of Application Programming Interfaces. In a nutshell, an API can be viewed as a rather restricted version of an SAL, where only one application has anything complex/interesting to say and, in general, the developers/owners of that application have sole responsibility for managing the versioning and changing of that API. Note also that SALs do not make any a priori assumption about the relationship (i.e. the default client/server for an API) or the message delivery mechanism (i.e. via default GET/POST to RESTful URLs). For more information on these ideas:

- [GridWorks Enums](https://gridwork-type-registry.readthedocs.io/en/latest/types.html)
- [GridWorks Types](https://gridwork-type-registry.readthedocs.io/en/latest/types.html)
- [ASLs](https://gridwork-type-registry.readthedocs.io/en/latest/asls.html)

`class gwproto.enums.ActorClass(value)`

Determines the code running Spaceheat Nodes supervised by Spaceheat SCADA software

Enum sh.actor.class version 001 in the GridWorks Type registry.

Used by used by multiple Application Shared Languages (ASLs), including but not limited to gwproto. For more information:

- [ASLs](https://gridworks-type-registry.readthedocs.io/en/latest/)
- [Global Authority](https://gridworks-type-registry.readthedocs.io/en/latest/enums.html#shactorclass)
- [More Info](https://gridworks-protocol.readthedocs.io/en/latest/actor-class.html)

Values (with symbols in parens):

- NoActor (00000000): A SpaceheatNode that does not have any code running on its behalf within the SCADA, but is instead only a reference object (for example, a tank of hot water or a resistive element) that can be discussed (for example, the power drawn by the resistive element can be measured) or evaluated (for example, a set of 5 different temperatures in different places on the tank can be used to estimate total thermal energy in the tank).
- Scada (6d37aa41): The SCADA actor is the prime piece of code running and supervising other ProActors within the SCADA code. It is also responsible for managing the state of TalkingWith the AtomicTNode, as well maintaining and reporting a boolean state variable that indicates whether it is following dispatch commands from the AtomicTNode XOR following dispatch commands from its own HomeAlone actor.
- HomeAlone (32d3d19f): HomeAlone is an abstract Spaceheat Actor responsible for dispatching the SCADA when it is not talking with the AtomicTNode.
- BooleanActuator (fddd0064): A SpaceheatNode representing a generic boolean actuator capable of turning on (closing a circuit) or turning off (opening a circuit).
- PowerMeter (2ea112b9): A SpaceheatNode representing the power meter that is used to settle financial transactions with the TerminalAsset. That is, this is the power meter whose accuracy is certified in the creation of the TerminalAsset GNode via creation of the TaDeed. [More Info](https://gridworks.readthedocs.io/en/latest/terminal-asset.html).
- Atn (b103058f): A SpaceheatNode representing the AtomicTNode. Note that the code running the AtomicTNode is not local within the SCADA code, except for a stub used for testing purposes. [More Info](https://gridworks.readthedocs.io/en/latest/atomic-t-node.html).

- SimpleSensor (dae4b2f0): A SpaceheatNode representing a sensor that measures a single category of quantity (for example, temperature) for a single object (for example, on a pipe). [More Info](https://gridworks-protocol.readthedocs.io/en/latest/simple-sensor.html).
- MultipurposeSensor (7c483ad0): A sensor that either reads multiple kinds of readings from the same sensing device (for example reads current and voltage), reads multiple different objects (temperature from two different thermisters) or both. [More Info](https://gridworks-protocol.readthedocs.io/en/latest/multipurpose-sensor.html).
- Thermostat (4a9c1785): A SpaceheatNode representing a thermostat.
- HubitatTelemetryReader (0401b27e): A generic actor for reading telemetry data from a Hubitat Home Automation Hub LAN API. [More Info](https://drive.google.com/drive/u/0/folders/1AqAU_lC2phzuI9XRYvogiIYA7GXNtlr6).
- HubitatTankModule (e2877329): The actor for running a GridWorks TankModule, comprised of two Z-Wave Fibaro temp sensors built together inside a small container that has 4 thermistors attached. These are designed to be installed from top (1) to bottom (4) on a stratified thermal storage tank. [More Info](https://drive.google.com/drive/u/0/folders/1GSxDd8Naf1GKK_fSOgQU933M1UcJ4r8q).
- HubitatPoller (00000100): An actor for representing a somewhat generic ShNode (like a thermostat) that can be polled through the Hubitat.
- Hubitat: (0000101): An actor for representing a Hubitat for receiving Hubitat events over HTTP.
- HoneywellThermostat: (0000102): An actor for representing a Honeywell Hubitat thermostat which can load thermostat heating state change messages into status reports.

classmethod default()

Returns default value (in this case NoActor)

Return type
ActorClass

classmethod enum_name()

The name in the GridWorks Type Registry (sh.actor.class)

Return type
str

classmethod enum_version()

The version in the GridWorks Type Registry (001)

Return type
str

classmethod symbol_to_value(*symbol*)

Given the symbol sent in a serialized message, returns the encoded enum.

Parameters
symbol (*str*) – The candidate symbol.

Returns
The encoded value associated to that symbol. If the symbol is not recognized - which could happen if the actor making the symbol is using a later version of this enum, returns the default value of "NoActor".

Return type
str

classmethod symbols()

Returns a list of the enum symbols

Return type
List[str]

classmethod value_to_symbol(*value*)

Provides the encoding symbol for a ActorClass enum to send in seriliazed messages.

Parameters
value (*str*) – The candidate value.

Returns
The symbol encoding that value. If the value is not recognized - which could happen if the actor making the message used a later version of this enum than the actor decoding the message, returns the default symbol of "00000000".

Return type
str

classmethod values()

Returns enum choices

Return type
List[str]

classmethod version(*value*)

Returns the version of an enum value.

Once a value belongs to one version of the enum, it belongs to all future versions.

Parameters
value (*str*) – The candidate enum value.

Raises
ValueError – If value is not one of the enum values.

Returns
The earliest version of the enum containing value.

Return type
str

class gwproto.enums.LocalCommInterface(*value*)

Categorization of in-house comm mechanisms for SCADA

Enum local.comm.interface version 000 in the GridWorks Type registry.

Used by used by multiple Application Shared Languages (ASLs), including but not limited to gwproto. For more information:

- [ASLs](https://gridworks-type-registry.readthedocs.io/en/latest/)
- [Global Authority](https://gridworks-type-registry.readthedocs.io/en/latest/enums.html#localcomminterface)

Values (with symbols in parens):

- Unknown (00000000)
- I2C (9ec8bc49)
- Ethernet (c1e7a955)

- OneWire (ae2d4cd8)
- RS485 (a6a4ac9f)
- SimRabbit (efc144cd)
- Wifi (46ac6589)
- Analog_4_20_mA (653c73b8)
- RS232 (0843a726)

classmethod default()

Returns default value (in this case UNKNOWN)

Return type
LocalCommInterface

classmethod enum_name()

The name in the GridWorks Type Registry (local.comm.interface)

Return type
str

classmethod enum_version()

The version in the GridWorks Type Registry (000)

Return type
str

classmethod symbol_to_value(*symbol*)

Given the symbol sent in a serialized message, returns the encoded enum.

Parameters
symbol (*str*) – The candidate symbol.

Returns
The encoded value associated to that symbol. If the symbol is not recognized - which could happen if the actor making the symbol is using a later version of this enum, returns the default value of "Unknown".

Return type
str

classmethod symbols()

Returns a list of the enum symbols

Return type
List[str]

classmethod value_to_symbol(*value*)

Provides the encoding symbol for a LocalCommInterface enum to send in seriliazed messages.

Parameters
value (*str*) – The candidate value.

Returns
The symbol encoding that value. If the value is not recognized - which could happen if the actor making the message used a later version of this enum than the actor decoding the message, returns the default symbol of "00000000".

Return type
str

classmethod values()

Returns enum choices

Return type
List[str]

classmethod version(*value*)

Returns the version of an enum value.

Once a value belongs to one version of the enum, it belongs to all future versions.

Parameters
value (*str*) – The candidate enum value.

Raises
ValueError – If value is not one of the enum values.

Returns
The earliest version of the enum containing value.

Return type
str

class gwproto.enums.**MakeModel**(*value*)

Determines Make/Model of device associated to a Spaceheat Node supervised by SCADA

Enum spaceheat.make.model version 001 in the GridWorks Type registry.

Used by used by multiple Application Shared Languages (ASLs), including but not limited to gwproto. For more information:

- [ASLs](https://gridworks-type-registry.readthedocs.io/en/latest/)
- [Global Authority](https://gridworks-type-registry.readthedocs.io/en/latest/enums.html#spaceheatmakemodel)
- [More Info](https://gridworks-protocol.readthedocs.io/en/latest/make-model.html)

Values (with symbols in parens):

- UnknownMake__UnknownModel (00000000)
- Egauge__4030 (beb6d3fb): A power meter in Egauge's 403x line. [More Info](https://drive.google.com/drive/u/0/folders/1abJ-o9tlTscsQpMvT6SHxIm5j5aODgfA).
- NCD__PR8-14-SPST (fabfa505): NCD's 4-channel high-power relay controller + 4 GPIO with I2C interface. [More Info](https://store.ncd.io/product/4-channel-high-power-relay-controller-4-gpio-with-i2c-interface/?attribute_pa_choose-a-relay=20-amp-spdt).
- Adafruit__642 (acd93fb3): Adafruit's high-temp, water-proof 1-wire temp sensor. [More Info](https://www.adafruit.com/product/642).
- GridWorks__TSnap1 (d0178dc3): Actual GridWorks TSnap 1.0 SCADA Box.
- GridWorks__WaterTempHighPrecision (f8b497e8): Simulated temp sensor.
- Gridworks__SimPm1 (076da322): Simulated power meter.
- SchneiderElectric__Iem3455 (d300635e): Schneider Electric IEM 344 utility meter.
- GridWorks__SimBool30AmpRelay (e81d74a8): Simulated relay.
- OpenEnergy__EmonPi (c75d269f): Open Energy's open source multipurpose sensing device (including internal power meter). [More Info](https://docs.openenergymonitor.org/emonpi/technical.html).

- GridWorks__SimTSnap1 (3042c432): Simulated SCADA Box.
- Atlas__EzFlo (d0b0e375): Atlas Scientific EZO Embedded Flow Meter Totalizer, pulse to I2C. [More Info](https://drive.google.com/drive/u/0/folders/142bBV1pQIbMpyIR_0iRUr5gnzWgknOJp).
- Hubitat__C7__LAN1 (4d649420): This refers to a Hubitat C7 that has been configured in a specific way with respect to the APIs it presents on the Local Area Network. The Hubitat C7 is a home automation hub that supports building ZigBee and ZWave meshes, plugs into Ethernet, has a reasonable user interface and has an active community of open-source developers who create drivers and package managers for devices, and supports the creation of various types of APIs on the Local Area Network. [More Info](https://drive.google.com/drive/folders/1AqAU_lC2phzuI9XRYvogiIYA7GXNtlr6).
- GridWorks__Tank_Module_1 (bd759051): This refers to a small module designed and assembled by GridWorks that is meant to be mounted to the side of a hot water tank. It requires 24V DC and has 4 temperature sensors coming out of it labeled 1, 2, 3 and 4. It is meant to provide temperature readings (taken within a half a second of each other) of all 4 of its sensors once a minute. [More Info](https://drive.google.com/drive/folders/1GSxDd8Naf1GKK_fSOgQU933M1UcJ4r8q).
- Fibaro__Analog_Temp_Sensor (1f19839d): This enum refers to a Fibaro FGBS-222 home automation device that has been configured in a specific way. This includes (1) being attached to two 10K NTC thermistors and a specific voltage divider circuit that specifies its temperature as a function of voltage and (2) one of its potential free outputs being in-line with the power of a partner Fibaro, so that it can power cycle its partner (because there are reports of Fibaros no longer reporting temp change after weeks or months until power cylced). The Fibaro itself is a tiny (29 X 18 X 13 mm) Z-Wave device powered on 9-30V DC that can read up to 6 1-wire DS18B20 temp sensors, 2 0-10V analog inputs and also has 2 potential free outputs. [More Info](https://drive.google.com/drive/u/0/folders/1Muhsvw00goppHIfGSEmreX4hM6V78b-m).
- Amphenol__NTC_10K_Thermistor_MA100GG103BN (46f21cd5): A small gauge, low-cost, rapid response NTC 10K Thermistor designed for medical applications. [More Info](https://drive.google.com/drive/u/0/folders/11HW4ov66UvxKAwqApW6IrtoXatZBLQkd).
- YHDC__SCT013-100 (08da3f7d): YHDC current transformer [More Info](https://en.yhdc.com/product/SCT013-401.html).
- Magnelab__SCT-0300-050 (a8d9a70d): Magnelab 50A current transformer
- GridWorks__MultiTemp1 (bb31d136): GridWorks Analog temperature sensor that has 12 channels (labeled 1-12) to read 12 10K NTC Thermistors. It is comprised of 3 NCD ADS 1115 I2C temperature sensors with I2C Addresses 0x4b, 0x48, 0x49. [More Info](https://drive.google.com/drive/u/0/folders/1OuY0tunaad2Ie4Id3zFB7FcbEwHizWuL).
- Krida__Emr16-I2c-V3 (3353ce46): 16-Channel I2C Low Voltage Electromagnetic Relay Board [More Info](https://drive.google.com/drive/u/0/folders/1jL82MTRKEh9DDmxJFQ2yU2cjqnVD9Ik7).

classmethod default()

Returns default value (in this case UNKNOWNMAKE__UNKNOWNMODEL)

Return type
MakeModel

classmethod enum_name()

The name in the GridWorks Type Registry (spaceheat.make.model)

Return type
str

classmethod enum_version()

The version in the GridWorks Type Registry (001)

Return type
str

`classmethod symbol_to_value(symbol)`

Given the symbol sent in a serialized message, returns the encoded enum.

Parameters
symbol (*str*) – The candidate symbol.

Returns
The encoded value associated to that symbol. If the symbol is not recognized - which could happen if the actor making the symbol is using a later version of this enum, returns the default value of "UnknownMake__UnknownModel".

Return type
str

`classmethod symbols()`

Returns a list of the enum symbols

Return type
List[str]

`classmethod value_to_symbol(value)`

Provides the encoding symbol for a MakeModel enum to send in seriliazed messages.

Parameters
value (*str*) – The candidate value.

Returns
The symbol encoding that value. If the value is not recognized - which could happen if the actor making the message used a later version of this enum than the actor decoding the message, returns the default symbol of "00000000".

Return type
str

`classmethod values()`

Returns enum choices

Return type
List[str]

`classmethod version(value)`

Returns the version of an enum value.

Once a value belongs to one version of the enum, it belongs to all future versions.

Parameters
value (*str*) – The candidate enum value.

Raises
ValueError – If value is not one of the enum values.

Returns
The earliest version of the enum containing value.

Return type
str

class gwproto.enums.Role(*value*)

Categorizes SpaceheatNodes by their function within the heating system

Enum sh.node.role version 000 in the GridWorks Type registry.

Used by used by multiple Application Shared Languages (ASLs), including but not limited to gwproto. For more information:

- [ASLs](https://gridworks-type-registry.readthedocs.io/en/latest/)
- [Global Authority](https://gridworks-type-registry.readthedocs.io/en/latest/enums.html#shnoderole)
- [More Info](https://gridworks-protocol.readthedocs.io/en/latest/spaceheat-node-role.html)

Values (with symbols in parens):

- Unknown (00000000): Unknown Role
- Scada (d0afb424): Primary SCADA
- HomeAlone (863e50d1): HomeAlone GNode
- Atn (6ddff83b): AtomicTNode
- PowerMeter (9ac68b6e): A SpaceheatNode representing the power meter that is used to settle financial transactions with the TerminalAsset. That is, this is the power meter whose accuracy is certified in the creation of the TerminalAsset GNode via creation of the TaDeed. [More Info](https://gridworks.readthedocs.io/en/latest/terminal-asset.html).
- BoostElement (99c5f326): Resistive element used for providing heat to a thermal store.
- BooleanActuator (57b788ee): A solid state or mechanical relay with two states (open, closed)
- DedicatedThermalStore (3ecfe9b8): A dedicated thermal store within a thermal storage heating system - could be one or more water tanks, phase change material, etc.
- TankWaterTempSensor (73308a1f): A temperature sensor used for measuring temperature inside or on the immediate outside of a water tank.
- PipeTempSensor (c480f612): A temperature sensor used for measuring the temperature of a tank. Typically curved metal thermistor with thermal grease for good contact.
- RoomTempSensor (fec74958): A temperature sensor used for measuring room temperature, or temp in a heated space more generally.
- OutdoorTempSensor (5938bf1f): A temperature sensor used for measuring outdoor temperature.
- PipeFlowMeter (ece3b600): A meter that measures flow of liquid through a pipe, in units of VOLUME/TIME
- HeatedSpace (65725f44): A Heated Space.
- HydronicPipe (fe3cbdd5): A pipe carrying techinical water or other fluid (e.g. glycol) in a heating system.
- BaseboardRadiator (05fdd645): A baseboard radiator - one kind of emitter in a hydronic heating system.
- RadiatorFan (6896109b): A fan that can amplify the power out of a radiator.
- CirculatorPump (b0eaf2ba): Circulator pump for one or more of the hydronic pipe loops
- MultiChannelAnalogTempSensor (661d7e73): An analog multi channel temperature sensor
- Outdoors (dd975b31): The outdoors

classmethod default()

Returns default value (in this case Unknown)

Return type
Role

classmethod enum_name()

The name in the GridWorks Type Registry (sh.node.role)

Return type
str

classmethod enum_version()

The version in the GridWorks Type Registry (000)

Return type
str

classmethod symbol_to_value(*symbol*)

Given the symbol sent in a serialized message, returns the encoded enum.

Parameters
symbol (*str*) – The candidate symbol.

Returns
The encoded value associated to that symbol. If the symbol is not recognized - which could happen if the actor making the symbol is using a later version of this enum, returns the default value of "Unknown".

Return type
str

classmethod symbols()

Returns a list of the enum symbols

Return type
List[str]

classmethod value_to_symbol(*value*)

Provides the encoding symbol for a Role enum to send in seriliazed messages.

Parameters
value (*str*) – The candidate value.

Returns
The symbol encoding that value. If the value is not recognized - which could happen if the actor making the message used a later version of this enum than the actor decoding the message, returns the default symbol of "00000000".

Return type
str

classmethod values()

Returns enum choices

Return type
List[str]

classmethod version(*value*)

Returns the version of an enum value.

Once a value belongs to one version of the enum, it belongs to all future versions.

Parameters
value (*str*) – The candidate enum value.

Raises
ValueError – If value is not one of the enum values.

Returns
The earliest version of the enum containing value.

Return type
str

class gwproto.enums.**TelemetryName**(*value*)

Specifies the name of sensed data reported by a Spaceheat SCADA

Enum spaceheat.telemetry.name version 001 in the GridWorks Type registry.

Used by used by multiple Application Shared Languages (ASLs), including but not limited to gwproto. For more information:

- [ASLs](https://gridworks-type-registry.readthedocs.io/en/latest/)
- [Global Authority](https://gridworks-type-registry.readthedocs.io/en/latest/enums.html#spaceheattelemetryname)
- [More Info](https://gridworks-protocol.readthedocs.io/en/latest/telemetry-name.html)

Values (with symbols in parens):

- Unknown (00000000): Default Value - unknown telemetry name.
- PowerW (af39eec9): Power in Watts.
- RelayState (5a71d4b3): An associated read must be either 0 or 1, with 0 meaning that the relay is open and current CANNOT flow and 1 meaning that the relay is closed and current CAN flow. Note in particular that this TelemetryName is NOT meant to be used to reflect whether a relay is energized or de-energized and in particular '1' means the same thing for both Normally Open and Normally Closed relays. Also, it is not meant to be used for a double-throw relay.
- WaterTempCTimes1000 (c89d0ba1): Water temperature, in Degrees Celcius multiplied by 1000. Example: 43200 means 43.2 deg Celcius.
- WaterTempFTimes1000 (793505aa): Water temperature, in Degrees F multiplied by 1000. Example: 142100 means 142.1 deg Fahrenheit.
- GpmTimes100 (d70cce28): Gallons Per Minute multiplied by 100. Example: 433 means 4.33 gallons per minute.
- CurrentRmsMicroAmps (ad19e79c): Current measurement in Root Mean Square MicroAmps.
- GallonsTimes100 (329a68c0): Gallons multipled by 100. This is useful for flow meters that report cumulative gallons as their raw output. Example: 55300 means 55.3 gallons.
- VoltageRmsMilliVolts (bb6fdd59): Voltage in Root Mean Square MilliVolts.
- MilliWattHours (e0bb014b): Energy in MilliWattHours.
- FrequencyMicroHz (337b8659): Frequency in MicroHz. Example: 59,965,332 means 59.965332 Hz.
- AirTempCTimes1000 (0f627faa): Air temperature, in Degrees Celsius multiplied by 1000. Example: 6234 means 6.234 deg Celcius.
- AirTempFTimes1000 (4c3f8c78): Air temperature, in Degrees F multiplied by 1000. Example: 69329 means 69.329 deg Fahrenheit.

- ThermostatState (00002000): An enum representing the state of the thermostat heat call.

classmethod default()

Returns default value (in this case Unknown)

Return type
TelemetryName

classmethod enum_name()

The name in the GridWorks Type Registry (spaceheat.telemetry.name)

Return type
str

classmethod enum_version()

The version in the GridWorks Type Registry (001)

Return type
str

classmethod symbol_to_value(*symbol*)

Given the symbol sent in a serialized message, returns the encoded enum.

Parameters
symbol (*str*) – The candidate symbol.

Returns
The encoded value associated to that symbol. If the symbol is not recognized - which could happen if the actor making the symbol is using a later version of this enum, returns the default value of "Unknown".

Return type
str

classmethod symbols()

Returns a list of the enum symbols

Return type
List[str]

classmethod value_to_symbol(*value*)

Provides the encoding symbol for a TelemetryName enum to send in seriliazed messages.

Parameters
value (*str*) – The candidate value.

Returns
The symbol encoding that value. If the value is not recognized - which could happen if the actor making the message used a later version of this enum than the actor decoding the message, returns the default symbol of "00000000".

Return type
str

classmethod values()

Returns enum choices

Return type
List[str]

classmethod version(*value*)

Returns the version of an enum value.

Once a value belongs to one version of the enum, it belongs to all future versions.

Parameters
value (*str*) – The candidate enum value.

Raises
ValueError – If value is not one of the enum values.

Returns
The earliest version of the enum containing value.

Return type
str

class gwproto.enums.**Unit**(*value*)

Specifies the physical unit of sensed data reported by SCADA

Enum spaceheat.unit version 000 in the GridWorks Type registry.

Used by used by multiple Application Shared Languages (ASLs), including but not limited to gwproto. For more information:

- [ASLs](https://gridworks-type-registry.readthedocs.io/en/latest/)
- [Global Authority](https://gridworks-type-registry.readthedocs.io/en/latest/enums.html#spaceheatunit)

Values (with symbols in parens):

- Unknown (00000000)
- Unitless (ec972387)
- W (f459a9c3)
- Celcius (ec14bd47)
- Fahrenheit (7d8832f8)
- Gpm (b4580361)
- WattHours (d66f1622)
- AmpsRms (a969ac7c)
- VoltsRms (e5d7555c)
- Gallons (8e123a26)
- ThermostatStateEnum (00003000)

classmethod default()

Returns default value (in this case Unknown)

Return type
Unit

classmethod enum_name()

The name in the GridWorks Type Registry (spaceheat.unit)

Return type
str

classmethod enum_version()

The version in the GridWorks Type Registry (000)

Return type
str

classmethod symbol_to_value(*symbol*)

Given the symbol sent in a serialized message, returns the encoded enum.

Parameters
symbol (*str*) – The candidate symbol.

Returns
The encoded value associated to that symbol. If the symbol is not recognized - which could happen if the actor making the symbol is using a later version of this enum, returns the default value of "Unknown".

Return type
str

classmethod symbols()

Returns a list of the enum symbols

Return type
List[str]

classmethod value_to_symbol(*value*)

Provides the encoding symbol for a Unit enum to send in seriliazed messages.

Parameters
value (*str*) – The candidate value.

Returns
The symbol encoding that value. If the value is not recognized - which could happen if the actor making the message used a later version of this enum than the actor decoding the message, returns the default symbol of "00000000".

Return type
str

classmethod values()

Returns enum choices

Return type
List[str]

classmethod version(*value*)

Returns the version of an enum value.

Once a value belongs to one version of the enum, it belongs to all future versions.

Parameters
value (*str*) – The candidate enum value.

Raises
ValueError – If value is not one of the enum values.

Returns
The earliest version of the enum containing value.

Return type
str

1.5 SDK for gridworks-protocol Types

The Python classes enumerated below provide an interpretation of gridworks-protocol type instances (serialized JSON) as Python objects. Types are the building blocks for all GridWorks APIs. You can read more about how they work here, and examine their API specifications here. The Python classes below also come with methods for translating back and forth between type instances and Python objects. List of all the types

1.5.1 ComponentAttributeClassGt

Python pydantic class corresponding to json type *component.attribute.class.gt*, version *000*.

class gwproto.types.**ComponentAttributeClassGt**(*, *ComponentAttributeClassId*, *DisplayName=None*, *TypeName='component.attribute.class.gt'*, *Version='000'*)

Component Attribute Class Gt.

Authority for the attributes of the component.attribute.class.gt.000 belongs to the WorldRegistry. The WorldRegistry is part of the GridWorks 'BackOffice' structure for managing relational device data. Generally speaking, a component attribute class is meant to specify WHAT you might order from a plumbing supply store to 'get the same part.' The Component refers to something that will have a specific serial number.

[More info](https://g-node-registry.readthedocs.io/en/latest/component-attribute-class.html)

Parameters

- **ComponentAttributeClassId** (*str*)
- **DisplayName** (*str | None*)
- **TypeName** (*Literal['component.attribute.class.gt']*)
- **Version** (*Literal['000']*)

as_dict()

Translate the object into a dictionary representation that can be serialized into a component.attribute.class.gt.000 object.

This method prepares the object for serialization by the as_type method, creating a dictionary with key-value pairs that follow the requirements for an instance of the component.attribute.class.gt.000 type. Unlike the standard python dict method, it makes the following substantive changes: - Enum Values: Translates between the values used locally by the actor to the symbol sent in messages. - Removes any key-value pairs where the value is None for a clearer message, especially in cases with many optional attributes.

It also applies these changes recursively to sub-types.

Return type
Dict[str, *Any*]

as_type()

Serialize to the component.attribute.class.gt.000 representation.

Instances in the class are python-native representations of component.attribute.class.gt.000 objects, while the actual component.attribute.class.gt.000 object is the serialized UTF-8 byte string designed for sending in a message.

This method calls the as_dict() method, which differs from the native python dict() in the following key ways: - Enum Values: Translates between the values used locally by the actor to the symbol sent in messages. - - Removes any key-value pairs where the value is None for a clearer message, especially in cases with many optional attributes.

It also applies these changes recursively to sub-types.

Its near-inverse is ComponentAttributeClassGt.type_to_tuple(). If the type (or any sub-types) includes an enum, then the type_to_tuple will map an unrecognized symbol to the default enum value. This is why these two methods are only 'near' inverses.

Return type
bytes

ComponentAttributeClassId:

- Description: ComponentAttributeClassId. Unique identifier for the device class (aka 'cac' or Component Attribute Class). This identifier is used to associate a make/model with a specific component (i.e. the component will point to its ComponentAttributeClassId).
- Format: UuidCanonicalTextual

DisplayName:

- Description: DisplayName. Optional Mutable field to include manufacturer's model name. Note that several different models may be given the same spaceheat.make.model enum name.

TypeName:

- Description: All GridWorks Versioned Types have a fixed TypeName, which is a string of lowercase alphanumeric words separated by periods, most significant word (on the left) starting with an alphabet character, and final word NOT all Hindu-Arabic numerals.

Version:

- Description: All GridWorks Versioned Types have a fixed version, which is a string of three Hindu-Arabic numerals.

class gwproto.types.component_attribute_class_gt.**check_is_uuid_canonical_textual**(*v*)

Checks UuidCanonicalTextual format

UuidCanonicalTextual format: A string of hex words separated by hyphens of length 8-4-4-4-12.

Parameters
v (*str*) – the candidate

Raises
ValueError – if v is not UuidCanonicalTextual format

class gwproto.types.**ComponentAttributeClassGt_Maker**(*component_attribute_class_id*, *display_name*)

Parameters

- **component_attribute_class_id** (*str*)
- **display_name** (*str | None*)

classmethod **dict_to_tuple**(*d*)

Deserialize a dictionary representation of a component.attribute.class.gt.000 message object into a ComponentAttributeClassGt python object for internal use.

This is the near-inverse of the ComponentAttributeClassGt.as_dict() method:

- Enums: translates between the symbols sent in messages between actors and

the values used by the actors internally once they've deserialized the messages.

- Types: recursively validates and deserializes sub-types.

Note that if a required attribute with a default value is missing in a dict, this method will raise a SchemaError. This differs from the pydantic BaseModel practice of auto-completing missing attributes with default values when they exist.

Parameters
d (*dict*) – the dictionary resulting from json.loads(t) for a serialized JSON type object t.

Raises
SchemaError – if the dict cannot be turned into a ComponentAttributeClassGt object.

Returns
ComponentAttributeClassGt

Return type
ComponentAttributeClassGt

classmethod tuple_to_type(*tuple_*)

Given a Python class object, returns the serialized JSON type object.

Parameters
tuple_ (ComponentAttributeClassGt)

Return type
bytes

classmethod type_to_tuple(*t*)

Given a serialized JSON type object, returns the Python class object.

Parameters
t (*bytes*)

Return type
ComponentAttributeClassGt

1.5.2 ComponentGt

Python pydantic class corresponding to json type *component.gt*, version *000*.

class gwproto.types.**ComponentGt**(*, *ComponentId*, *ComponentAttributeClassId*, *DisplayName=None*, *HwUid=None*, *TypeName='component.gt'*, *Version='000'*)

Component Gt.

Authority for the attributes of the component.gt.000 (ComponentId, ComponentAttributeClassId, DisplayName, HwUid) belongs to the WorldRegistry. The WorldRegistry is part of the GridWorks 'BackOffice' structure for managing relational device data . Notably, ComponentId and ComponentAttributeClass are both required and immutable. HwUid is optional but once it is set to a non-null value that is also immutable - it is meant to be an immutable identifier associated to a specific physical device, ideally one that can be read remotely by the SCADA and also by the naked eye. The DisplayName is mutable, with its current value in time governed by the WorldRegistry.

[More info](https://g-node-registry.readthedocs.io/en/latest/component.html)

Parameters

- **ComponentId** (*str*)
- **ComponentAttributeClassId** (*str*)
- **DisplayName** (*str | None*)
- **HwUid** (*str | None*)

- **TypeName** (*Literal['component.gt']*)
- **Version** (*Literal['000']*)

as_dict()

Translate the object into a dictionary representation that can be serialized into a component.gt.000 object.

This method prepares the object for serialization by the as_type method, creating a dictionary with key-value pairs that follow the requirements for an instance of the component.gt.000 type. Unlike the standard python dict method, it makes the following substantive changes: - Enum Values: Translates between the values used locally by the actor to the symbol sent in messages. - Removes any key-value pairs where the value is None for a clearer message, especially in cases with many optional attributes.

It also applies these changes recursively to sub-types.

Return type
Dict[str, *Any*]

as_type()

Serialize to the component.gt.000 representation.

Instances in the class are python-native representations of component.gt.000 objects, while the actual component.gt.000 object is the serialized UTF-8 byte string designed for sending in a message.

This method calls the as_dict() method, which differs from the native python dict() in the following key ways: - Enum Values: Translates between the values used locally by the actor to the symbol sent in messages. - - Removes any key-value pairs where the value is None for a clearer message, especially in cases with many optional attributes.

It also applies these changes recursively to sub-types.

Its near-inverse is ComponentGt.type_to_tuple(). If the type (or any sub-types) includes an enum, then the type_to_tuple will map an unrecognized symbol to the default enum value. This is why these two methods are only 'near' inverses.

Return type
bytes

ComponentId:

- Description: Component Id. Primary identifier for components in all GridWorks registries.
- Format: UuidCanonicalTextual

ComponentAttributeClassId:

- Description: ComponentAttributeClassId. Unique identifier for the device class. Authority for these, as well as the relationship between Components and ComponentAttributeClasses (Cacs) is maintained by the World Registry.
- Format: UuidCanonicalTextual

DisplayName:

- Description: This is an optional, mutable field whose use is strongly encouraged. It may include information about HOW the component is used in a hardware layout. It may also include the HwUid for the component.

HwUid:

- Description: Usually this is determined by the inheriting class.

TypeName:

- Description: All GridWorks Versioned Types have a fixed TypeName, which is a string of lowercase alphanumeric words separated by periods, most significant word (on the left) starting with an alphabet character, and final word NOT all Hindu-Arabic numerals.

Version:

- Description: All GridWorks Versioned Types have a fixed version, which is a string of three Hindu-Arabic numerals.

class gwproto.types.component_gt.**check_is_uuid_canonical_textual**(*v*)

Checks UuidCanonicalTextual format

UuidCanonicalTextual format: A string of hex words separated by hyphens of length 8-4-4-4-12.

Parameters
v (*str*) – the candidate

Raises
ValueError – if v is not UuidCanonicalTextual format

class gwproto.types.**ComponentGt_Maker**(*component_id*, *component_attribute_class_id*, *display_name*, *hw_uid*)

Parameters

- **component_id** (*str*)
- **component_attribute_class_id** (*str*)
- **display_name** (*str | None*)
- **hw_uid** (*str | None*)

classmethod **dict_to_tuple**(*d*)

Deserialize a dictionary representation of a component.gt.000 message object into a ComponentGt python object for internal use.

This is the near-inverse of the ComponentGt.as_dict() method:

- Enums: translates between the symbols sent in messages between actors and

the values used by the actors internally once they've deserialized the messages.

- Types: recursively validates and deserializes sub-types.

Note that if a required attribute with a default value is missing in a dict, this method will raise a SchemaError. This differs from the pydantic BaseModel practice of auto-completing missing attributes with default values when they exist.

Parameters
d (*dict*) – the dictionary resulting from json.loads(t) for a serialized JSON type object t.

Raises
SchemaError – if the dict cannot be turned into a ComponentGt object.

Returns
ComponentGt

Return type
ComponentGt

classmethod **tuple_to_type**(*tuple_*)

Given a Python class object, returns the serialized JSON type object.

Parameters
tuple_ (ComponentGt)

Return type
bytes

classmethod type_to_tuple(*t*)

Given a serialized JSON type object, returns the Python class object.

Parameters
t (*bytes*)

Return type
ComponentGt

1.5.3 DataChannel

Python pydantic class corresponding to json type *data.channel*, version *000*.

class gwproto.types.**DataChannel**(*, *DisplayName*, *AboutName*, *CapturedByName*, *TelemetryName*, *TypeName='data.channel'*, *Version='000'*)

Data Channel.

A data channel is a concept of some collection of readings that share all characteristics other than time.

Parameters

- **DisplayName** (*str*)
- **AboutName** (*str*)
- **CapturedByName** (*str*)
- **TelemetryName** (TelemetryName)
- **TypeName** (*Literal['data.channel']*)
- **Version** (*Literal['000']*)

as_dict()

Translate the object into a dictionary representation that can be serialized into a data.channel.000 object.

This method prepares the object for serialization by the as_type method, creating a dictionary with key-value pairs that follow the requirements for an instance of the data.channel.000 type. Unlike the standard python dict method, it makes the following substantive changes: - Enum Values: Translates between the values used locally by the actor to the symbol sent in messages. - Removes any key-value pairs where the value is None for a clearer message, especially in cases with many optional attributes.

It also applies these changes recursively to sub-types.

Return type
Dict[str, *Any*]

as_type()

Serialize to the data.channel.000 representation.

Instances in the class are python-native representations of data.channel.000 objects, while the actual data.channel.000 object is the serialized UTF-8 byte string designed for sending in a message.

This method calls the as_dict() method, which differs from the native python dict() in the following key ways: - Enum Values: Translates between the values used locally by the actor to the symbol sent in messages. - - Removes any key-value pairs where the value is None for a clearer message, especially in cases with many optional attributes.

It also applies these changes recursively to sub-types.

Its near-inverse is DataChannel.type_to_tuple(). If the type (or any sub-types) includes an enum, then the type_to_tuple will map an unrecognized symbol to the default enum value. This is why these two methods are only 'near' inverses.

Return type
bytes

DisplayName:

- Description: Display Name. This display name is the handle for the data channel. It is meant to be set by the person/people who will be analyzing time series data. It is only expected to be unique within the data channels associated to a specific Terminal Asset.

AboutName:

- Description: About Name. The name of the SpaceheatNode whose physical quantities are getting captured.
- Format: SpaceheatName

CapturedByName:

- Description: The name of the SpaceheatNode that is capturing the physical quantities (which can be AboutName but does not have to be).
- Format: SpaceheatName

TelemetryName:

- Description: The name of the physical quantity getting measured.

TypeName:

- Description: All GridWorks Versioned Types have a fixed TypeName, which is a string of lowercase alphanumeric words separated by periods, most significant word (on the left) starting with an alphabet character, and final word NOT all Hindu-Arabic numerals.

Version:

- Description: All GridWorks Versioned Types have a fixed version, which is a string of three Hindu-Arabic numerals.

`class gwproto.types.data_channel.check_is_spaceheat_name(v)`

Check SpaceheatName Format.

Validates if the provided string adheres to the SpaceheatName format: Lowercase words separated by periods, where word characters can be alphanumeric or a hyphen, and the first word starts with an alphabet character.

Parameters
v (*str*) – The string to be validated.

Raises
ValueError – If the provided string is not in SpaceheatName format.

`class gwproto.types.DataChannel_Maker(display_name, about_name, captured_by_name, telemetry_name)`

Parameters

- **display_name** (*str*)

- **about_name** (*str*)
- **captured_by_name** (*str*)
- **telemetry_name** (TelemetryName)

classmethod dict_to_tuple(*d*)

Deserialize a dictionary representation of a data.channel.000 message object into a DataChannel python object for internal use.

This is the near-inverse of the DataChannel.as_dict() method:

- Enums: translates between the symbols sent in messages between actors and

the values used by the actors internally once they've deserialized the messages.

- Types: recursively validates and deserializes sub-types.

Note that if a required attribute with a default value is missing in a dict, this method will raise a SchemaError. This differs from the pydantic BaseModel practice of auto-completing missing attributes with default values when they exist.

Parameters
d (*dict*) – the dictionary resulting from json.loads(t) for a serialized JSON type object t.

Raises
SchemaError – if the dict cannot be turned into a DataChannel object.

Returns
DataChannel

Return type
DataChannel

classmethod tuple_to_type(*tpl*)

Given a Python class object, returns the serialized JSON type object.

Parameters
tpl (DataChannel)

Return type
bytes

classmethod type_to_tuple(*t*)

Given a serialized JSON type object, returns the Python class object.

Parameters
t (*bytes*)

Return type
DataChannel

1.5.4 EgaugeIo

Python pydantic class corresponding to json type *egauge.io*, version *000*.

class gwproto.types.**EgaugeIo**(*, *InputConfig*, *OutputConfig*, *TypeName='egauge.io'*, *Version='000'*)

Used for an eGauge meter's component information in a hardware layout.

When the component associated to a PowerMeter ShNode has MakeModel EGAUGE__4030, there is a significant amount of configuration required to specify both what is read from the eGauge (input) and what is then sent up to the SCADA (output). This type handles that information.

[More info](https://gridworks-protocol.readthedocs.io/en/latest/egauge-io.html)

Parameters

- **InputConfig** (EgaugeRegisterConfig)
- **OutputConfig** (TelemetryReportingConfig)
- **TypeName** (*Literal['egauge.io']*)
- **Version** (*Literal['000']*)

as_dict()

Translate the object into a dictionary representation that can be serialized into a egauge.io.000 object.

This method prepares the object for serialization by the as_type method, creating a dictionary with key-value pairs that follow the requirements for an instance of the egauge.io.000 type. Unlike the standard python dict method, it makes the following substantive changes: - Enum Values: Translates between the values used locally by the actor to the symbol sent in messages. - Removes any key-value pairs where the value is None for a clearer message, especially in cases with many optional attributes.

It also applies these changes recursively to sub-types.

Return type
Dict[str, *Any*]

as_type()

Serialize to the egauge.io.000 representation.

Instances in the class are python-native representations of egauge.io.000 objects, while the actual egauge.io.000 object is the serialized UTF-8 byte string designed for sending in a message.

This method calls the as_dict() method, which differs from the native python dict() in the following key ways: - Enum Values: Translates between the values used locally by the actor to the symbol sent in messages. - - Removes any key-value pairs where the value is None for a clearer message, especially in cases with many optional attributes.

It also applies these changes recursively to sub-types.

Its near-inverse is EgaugeIo.type_to_tuple(). If the type (or any sub-types) includes an enum, then the type_to_tuple will map an unrecognized symbol to the default enum value. This is why these two methods are only 'near' inverses.

Return type
bytes

InputConfig:

- Description: Input config for one channel of data for a specific eGauge meter. This is the data available from the modbus csv map provided by eGauge for this component, for example http://egauge14875.egaug.es/6001C/settings.html for a eGauge device with ID 14875

OutputConfig:

- Description: Output config for the same channel . This is the data as the Scada proactor expects to consume it from the power meter driver proactor.

TypeName:

- Description: All GridWorks Versioned Types have a fixed TypeName, which is a string of lowercase alphanumeric words separated by periods, most significant word (on the left) starting with an alphabet character, and final word NOT all Hindu-Arabic numerals.

Version:

- Description: All GridWorks Versioned Types have a fixed version, which is a string of three Hindu-Arabic numerals.

class gwproto.types.**EgaugeIo_Maker**(*input_config*, *output_config*)

Parameters

- **input_config** (EgaugeRegisterConfig)
- **output_config** (TelemetryReportingConfig)

classmethod **dict_to_tuple**(*d*)

Deserialize a dictionary representation of a egauge.io.000 message object into a EgaugeIo python object for internal use.

This is the near-inverse of the EgaugeIo.as_dict() method:

- Enums: translates between the symbols sent in messages between actors and

the values used by the actors internally once they've deserialized the messages.

- Types: recursively validates and deserializes sub-types.

Note that if a required attribute with a default value is missing in a dict, this method will raise a SchemaError. This differs from the pydantic BaseModel practice of auto-completing missing attributes with default values when they exist.

Parameters
d (*dict*) – the dictionary resulting from json.loads(t) for a serialized JSON type object t.

Raises
SchemaError – if the dict cannot be turned into a EgaugeIo object.

Returns
EgaugeIo

Return type
EgaugeIo

classmethod **tuple_to_type**(*tpl*)

Given a Python class object, returns the serialized JSON type object.

Parameters
tpl (EgaugeIo)

Return type
bytes

classmethod **type_to_tuple**(*t*)

Given a serialized JSON type object, returns the Python class object.

Parameters
t (*bytes*)

Return type
EgaugeIo

1.5.5 EgaugeRegisterConfig

Python pydantic class corresponding to json type *egauge.register.config*, version *000*.

class gwproto.types.**EgaugeRegisterConfig**(*, *Address*, *Name*, *Description*, *Type*, *Denominator*, *Unit*, *TypeName='egauge.register.config'*, *Version='000'*)

Used to translate eGauge's Modbus Map.

This type captures the information provided by eGauge in its modbus csv map, when reading current, power, energy, voltage, frequency etc from an eGauge 4030.

Parameters

- **Address** (*int*)
- **Name** (*str*)
- **Description** (*str*)
- **Type** (*str*)
- **Denominator** (*int*)
- **Unit** (*str*)
- **TypeName** (*Literal['egauge.register.config']*)
- **Version** (*Literal['000']*)

as_dict()

Translate the object into a dictionary representation that can be serialized into a egauge.register.config.000 object.

This method prepares the object for serialization by the as_type method, creating a dictionary with key-value pairs that follow the requirements for an instance of the egauge.register.config.000 type. Unlike the standard python dict method, it makes the following substantive changes: - Enum Values: Translates between the values used locally by the actor to the symbol sent in messages. - Removes any key-value pairs where the value is None for a clearer message, especially in cases with many optional attributes.

It also applies these changes recursively to sub-types.

Return type
Dict[str, *Any*]

as_type()

Serialize to the egauge.register.config.000 representation.

Instances in the class are python-native representations of egauge.register.config.000 objects, while the actual egauge.register.config.000 object is the serialized UTF-8 byte string designed for sending in a message.

This method calls the as_dict() method, which differs from the native python dict() in the following key ways: - Enum Values: Translates between the values used locally by the actor to the symbol sent in messages. - - Removes any key-value pairs where the value is None for a clearer message, especially in cases with many optional attributes.

It also applies these changes recursively to sub-types.

Its near-inverse is EgaugeRegisterConfig.type_to_tuple(). If the type (or any sub-types) includes an enum, then the type_to_tuple will map an unrecognized symbol to the default enum value. This is why these two methods are only 'near' inverses.

Return type
bytes

Address:

- Description: Address. EGauge's modbus holding address. Note that the EGauge modbus map for holding address 100 will be 30100 - the '+30000' indicates it is a holding address. We use the 4-digit address after the '3'.

Name:

- Description: Name. The name assigned in the EGauge's modbus map. This is configured by the user (see URL)

Description:

- Description: Description. Again, assigned by the EGauge modbus map. Is usually 'change in value'

Type:

- Description: Type. EGauge's numerical data type. Typically our power measurements are f32 (32-bit floating-point number). The serial number & firmware are t16 (which work to treat as 16-bit unsigned integer) and timestamps are u32 (32-bit unsigned integer).

Denominator:

- Description: Denominator. Some of the modbus registers divide by 3.60E+06 (cumulative energy registers typically). For the power, current, voltage and phase angle the denominator is 1.

Unit:

- Description: Unit. The EGauge unit - typically A, Hz, or W.

TypeName:

- Description: All GridWorks Versioned Types have a fixed TypeName, which is a string of lowercase alphanumeric words separated by periods, most significant word (on the left) starting with an alphabet character, and final word NOT all Hindu-Arabic numerals.

Version:

- Description: All GridWorks Versioned Types have a fixed version, which is a string of three Hindu-Arabic numerals.

`class gwproto.types.`**`EgaugeRegisterConfig_Maker`**`(`*address*, *name*, *description*, *type*, *denominator*, *unit*`)`

Parameters

- **address** (*int*)
- **name** (*str*)
- **description** (*str*)
- **type** (*str*)
- **denominator** (*int*)
- **unit** (*str*)

classmethod dict_to_tuple(*d*)

Deserialize a dictionary representation of a egauge.register.config.000 message object into a EgaugeRegisterConfig python object for internal use.

This is the near-inverse of the EgaugeRegisterConfig.as_dict() method:

- Enums: translates between the symbols sent in messages between actors and

the values used by the actors internally once they've deserialized the messages.

- Types: recursively validates and deserializes sub-types.

Note that if a required attribute with a default value is missing in a dict, this method will raise a SchemaError. This differs from the pydantic BaseModel practice of auto-completing missing attributes with default values when they exist.

Parameters
d (*dict*) – the dictionary resulting from json.loads(t) for a serialized JSON type object t.

Raises
SchemaError – if the dict cannot be turned into a EgaugeRegisterConfig object.

Returns
EgaugeRegisterConfig

Return type
EgaugeRegisterConfig

classmethod tuple_to_type(*tpl*)

Given a Python class object, returns the serialized JSON type object.

Parameters
tpl (EgaugeRegisterConfig)

Return type
bytes

classmethod type_to_tuple(*t*)

Given a serialized JSON type object, returns the Python class object.

Parameters
t (*bytes*)

Return type
EgaugeRegisterConfig

1.5.6 ElectricMeterCacGt

Python pydantic class corresponding to json type *electric.meter.cac.gt*, version *000*.

class gwproto.types.**ElectricMeterCacGt**(*, *ComponentAttributeClassId*, *MakeModel*, *DisplayName=None*, *TelemetryNameList*, *PollPeriodMs*, *Interface*, *DefaultBaud=None*, *TypeName='electric.meter.cac.gt'*, *Version='000'*)

Type for tracking Electric Meter ComponentAttributeClasses.

GridWorks Spaceheat SCADA uses the GridWorks GNodeRegistry structures and abstractions for managing relational device data. The Cac, or ComponentAttributeClass, is part of this structure.

[More info](https://g-node-registry.readthedocs.io/en/latest/component-attribute-class.html)

Parameters

- **ComponentAttributeClassId** (*str*)
- **MakeModel** (MakeModel)
- **DisplayName** (*str | None*)
- **TelemetryNameList** (*List[TelemetryName]*)
- **PollPeriodMs** (*int*)
- **Interface** (LocalCommInterface)
- **DefaultBaud** (*int | None*)
- **TypeName** (*Literal['electric.meter.cac.gt']*)
- **Version** (*Literal['000']*)

as_dict()

Translate the object into a dictionary representation that can be serialized into a electric.meter.cac.gt.000 object.

This method prepares the object for serialization by the as_type method, creating a dictionary with key-value pairs that follow the requirements for an instance of the electric.meter.cac.gt.000 type. Unlike the standard python dict method, it makes the following substantive changes: - Enum Values: Translates between the values used locally by the actor to the symbol sent in messages. - Removes any key-value pairs where the value is None for a clearer message, especially in cases with many optional attributes.

It also applies these changes recursively to sub-types.

Return type
Dict[str, *Any*]

as_type()

Serialize to the electric.meter.cac.gt.000 representation.

Instances in the class are python-native representations of electric.meter.cac.gt.000 objects, while the actual electric.meter.cac.gt.000 object is the serialized UTF-8 byte string designed for sending in a message.

This method calls the as_dict() method, which differs from the native python dict() in the following key ways: - Enum Values: Translates between the values used locally by the actor to the symbol sent in messages. - - Removes any key-value pairs where the value is None for a clearer message, especially in cases with many optional attributes.

It also applies these changes recursively to sub-types.

Its near-inverse is ElectricMeterCacGt.type_to_tuple(). If the type (or any sub-types) includes an enum, then the type_to_tuple will map an unrecognized symbol to the default enum value. This is why these two methods are only 'near' inverses.

Return type
bytes

ComponentAttributeClassId:

- Description: ComponentAttributeClassId. Unique identifier for the device class (aka 'cac' or Component Attribute Class). Authority is maintained by the World Registry.
- Format: UuidCanonicalTextual

MakeModel:

- Description: MakeModel. The brand name identifier for the electric meter (what you would specify in order to buy one).

DisplayName:

- Description: Sample: EGauge 4030

TelemetryNameList:

- Description: TelemetryNames read by this power meter.

PollPeriodMs:

- Description: Poll Period in Milliseconds. Poll Period refers to the period of time between two readings by the local actor. This is in contrast to Capture Period, which refers to the period between readings that are sent up to the cloud (or otherwise saved for the long-term).

Interface:

- Description:

DefaultBaud:

- Description: To be used when the comms method requires a baud rate.

TypeName:

- Description: All GridWorks Versioned Types have a fixed TypeName, which is a string of lowercase alphanumeric words separated by periods, most significant word (on the left) starting with an alphabet character, and final word NOT all Hindu-Arabic numerals.

Version:

- Description: All GridWorks Versioned Types have a fixed version, which is a string of three Hindu-Arabic numerals.

class gwproto.types.electric_meter_cac_gt.**check_is_uuid_canonical_textual**(*v*)

Checks UuidCanonicalTextual format

UuidCanonicalTextual format: A string of hex words separated by hyphens of length 8-4-4-4-12.

Parameters
v (*str*) – the candidate

Raises
ValueError – if v is not UuidCanonicalTextual format

class gwproto.types.electric_meter_cac_gt.**check_is_positive_integer**(*v*)

Must be positive when interpreted as an integer. Interpretation as an integer follows the pydantic rules for this - which will round down rational numbers. So 1.7 will be interpreted as 1 and is also fine, while 0.5 is interpreted as 0 and will raise an exception.

Parameters
v (*int*) – the candidate

Raises
ValueError – if v < 1

class gwproto.types.**ElectricMeterCacGt_Maker**(*component_attribute_class_id*, *make_model*, *display_name*, *telemetry_name_list*, *poll_period_ms*, *interface*, *default_baud*)

Parameters

- **component_attribute_class_id** (*str*)
- **make_model** (MakeModel)
- **display_name** (*str | None*)

- **telemetry_name_list** (*List[TelemetryName]*)
- **poll_period_ms** (*int*)
- **interface** (LocalCommInterface)
- **default_baud** (*int | None*)

classmethod dict_to_tuple(*d*)

Deserialize a dictionary representation of a electric.meter.cac.gt.000 message object into a ElectricMeterCacGt python object for internal use.

This is the near-inverse of the ElectricMeterCacGt.as_dict() method:

- Enums: translates between the symbols sent in messages between actors and

the values used by the actors internally once they've deserialized the messages.

- Types: recursively validates and deserializes sub-types.

Note that if a required attribute with a default value is missing in a dict, this method will raise a SchemaError. This differs from the pydantic BaseModel practice of auto-completing missing attributes with default values when they exist.

Parameters
d (*dict*) – the dictionary resulting from json.loads(t) for a serialized JSON type object t.

Raises
SchemaError – if the dict cannot be turned into a ElectricMeterCacGt object.

Returns
ElectricMeterCacGt

Return type
ElectricMeterCacGt

classmethod tuple_to_type(*tpl*)

Given a Python class object, returns the serialized JSON type object.

Parameters
tpl (ElectricMeterCacGt)

Return type
bytes

classmethod type_to_tuple(*t*)

Given a serialized JSON type object, returns the Python class object.

Parameters
t (*bytes*)

Return type
ElectricMeterCacGt

1.5.7 ElectricMeterComponentGt

Python pydantic class corresponding to json type *electric.meter.component.gt*, version *000*.

ComponentId:

- Description: Component Id. Primary GridWorks identifier for a specific physical instance of an ElectricMeter, and also as a more generic Component.
- Format: UuidCanonicalTextual

ComponentAttributeClassId:

- Description: ComponentAttributeClassId. Unique identifier for the device class. Authority for these, as well as the relationship between Components and ComponentAttributeClasses (Cacs) is maintained by the World Registry.
- Format: UuidCanonicalTextual

DisplayName:

- Description: Display Name for the Power Meter. Sample: Oak EGauge6074

ConfigList:

- Description: List of Data Channel configs . This power meter will produce multiple data channels. Each data channel measures a certain quantities (like power, current) for certain ShNodes (like a boost element or heat pump).

HwUid:

- Description: Unique Hardware Id for the Power Meter. For eGauge, use what comes back over modbus address 100.

ModbusHost:

- Description: Host on LAN when power meter is modbus over Ethernet.

ModbusPort:

- Description:
- Format: NonNegativeInteger

EgaugeIoList:

- Description: Bijecton from EGauge4030 input to ConfigList output. This should be empty unless the MakeModel of the corresponding component attribute class is EGauge 4030. The channels that can be read from an EGauge 4030 are configurable by the person who installs the device. The information is encapsulated in a modbus map provided by eGauge as a csv from a device-specific API. The EGaugeIoList maps the data from this map to the data that the SCADA expects to see.

TypeName:

- Description: All GridWorks Versioned Types have a fixed TypeName, which is a string of lowercase alphanumeric words separated by periods, most significant word (on the left) starting with an alphabet character, and final word NOT all Hindu-Arabic numerals.

Version:

- Description: All GridWorks Versioned Types have a fixed version, which is a string of three Hindu-Arabic numerals.

class gwproto.types.electric_meter_component_gt.**check_is_uuid_canonical_textual**(*v*)

Checks UuidCanonicalTextual format

UuidCanonicalTextual format: A string of hex words separated by hyphens of length 8-4-4-4-12.

Parameters
v (*str*) – the candidate

Raises
ValueError – if v is not UuidCanonicalTextual format

class gwproto.types.electric_meter_component_gt.**check_is_positive_integer**(*v*)

Must be positive when interpreted as an integer. Interpretation as an integer follows the pydantic rules for this - which will round down rational numbers. So 1.7 will be interpreted as 1 and is also fine, while 0.5 is interpreted as 0 and will raise an exception.

Parameters
v (*int*) – the candidate

Raises
ValueError – if v < 1

class gwproto.types.electric_meter_component_gt.**check_is_non_negative_integer**(*v*)

Must be non-negative when interpreted as an integer. Interpretation as an integer follows the pydantic rules for this - which will round down rational numbers. So 0 is fine, and 1.7 will be interpreted as 1 and is also fine.

Parameters
v (*int*) – the candidate

Raises
ValueError – if v < 0

1.5.8 FibaroSmartImplantCacGt

Python pydantic class corresponding to json type *fibaro.smart.implant.cac.gt*, version *000*.

class gwproto.types.**FibaroSmartImplantCacGt**(**, ComponentAttributeClassId, DisplayName=None, TypeName='fibaro.smart.implant.cac.gt', Version='000', Model='', **extra_data*)

Parameters

- **ComponentAttributeClassId** (*str*)
- **DisplayName** (*str | None*)
- **TypeName** (*Literal['fibaro.smart.implant.cac.gt']*)
- **Version** (*Literal['000']*)
- **Model** (*str*)
- **extra_data** (*Any*)

ComponentAttributeClassId:

- Description: ComponentAttributeClassId. Unique identifier for the device class (aka 'cac' or Component Attribute Class). Authority is maintained by the World Registry.
- Format: UuidCanonicalTextual

Model:

- Description:

DisplayName:

- Description: Sample: FGBS-222 v5.2

TypeName:

- Description: All GridWorks Versioned Types have a fixed TypeName, which is a string of lowercase alphanumeric words separated by periods, most significant word (on the left) starting with an alphabet character, and final word NOT all Hindu-Arabic numerals.

Version:

- Description: All GridWorks Versioned Types have a fixed version, which is a string of three Hindu-Arabic numerals.

class gwproto.types.**FibaroSmartImplantCacGt_Maker**(*cac*)

Parameters
cac (*FibaroSmartImplantCac*)

1.5.9 FibaroSmartImplantComponentGt

Python pydantic class corresponding to json type *fibaro.smart.implant.component.gt*, version *000*.

class gwproto.types.**FibaroSmartImplantComponentGt**(*, *ComponentId*, *ComponentAttributeClassId*, *DisplayName=None*, *HwUid=None*, *TypeName='fibaro.smart.implant.component.gt'*, *Version='000'*, *ZWaveDSK=''*)

Parameters

- **ComponentId** (*str*)
- **ComponentAttributeClassId** (*str*)
- **DisplayName** (*str | None*)
- **HwUid** (*str | None*)
- **TypeName** (*Literal['fibaro.smart.implant.component.gt']*)
- **Version** (*Literal['000']*)
- **ZWaveDSK** (*str*)

ComponentId:

- Description: Component Id. Primary GridWorks identifier for a specific physical instance of an Fibaro, and also as a more generic Component.
- Format: UuidCanonicalTextual

ComponentAttributeClassId:

- Description: ComponentAttributeClassId. Unique identifier for the device class. Authority for these, as well as the relationship between Components and ComponentAttributeClasses (Cacs) is maintained by the World Registry.
- Format: UuidCanonicalTextual

ZWaveDSK:

- Description: The Z-Wave DSK (Device Specific Key) is a unique identifier associated with a Z-Wave device, used during the process of securely including the device into a Z-Wave network. It helps establish secure communication between the Z-Wave controller and the device, ensuring that only authorized devices can join the network. Unfortunately Hubitat does not currently provide a way to view the ZWave DSK of a Fibaro.

DisplayName:

- Description: Sample: Fibaro Smart Implant 1010 A (For Fibaro A as opposed to B for GridWorks TankModule1 with Serial Number 1010).

HwUid:

- Description: Hardware Unique Id. Use the Fibaro S2 PIN Code, which is printed on the back of each Fibaro Implant.

TypeName:

- Description: All GridWorks Versioned Types have a fixed TypeName, which is a string of lowercase alphanumeric words separated by periods, most significant word (on the left) starting with an alphabet character, and final word NOT all Hindu-Arabic numerals.

Version:

- Description: All GridWorks Versioned Types have a fixed version, which is a string of three Hindu-Arabic numerals.

class gwproto.types.**FibaroSmartImplantComponentGt_Maker**(*component*)

Parameters
component (*FibaroSmartImplantComponent*)

1.5.10 GtDispatchBoolean

Python pydantic class corresponding to json type *gt.dispatch.boolean*, version *110*.

class gwproto.types.**GtDispatchBoolean**(*, *AboutNodeName*, *ToGNodeAlias*, *FromGNodeAlias*, *FromGNodeInstanceId*, *RelayState*, *SendTimeUnixMs*, *TypeName='gt.dispatch.boolean'*, *Version='110'*)

GridWorks Type Boolean Dispatch.

Boolean dispatch command designed to be sent from an AtomicTNode to a SCADA.

Parameters

- **AboutNodeName** (*str*)
- **ToGNodeAlias** (*str*)
- **FromGNodeAlias** (*str*)
- **FromGNodeInstanceId** (*str*)
- **RelayState** (*int*)
- **SendTimeUnixMs** (*int*)
- **TypeName** (*Literal['gt.dispatch.boolean']*)
- **Version** (*Literal['110']*)

`as_dict()`

Translate the object into a dictionary representation that can be serialized into a gt.dispatch.boolean.110 object.

This method prepares the object for serialization by the as_type method, creating a dictionary with key-value pairs that follow the requirements for an instance of the gt.dispatch.boolean.110 type. Unlike the standard python dict method, it makes the following substantive changes: - Enum Values: Translates between the values used locally by the actor to the symbol sent in messages. - Removes any key-value pairs where the value is None for a clearer message, especially in cases with many optional attributes.

It also applies these changes recursively to sub-types.

Return type
Dict[str, *Any*]

`as_type()`

Serialize to the gt.dispatch.boolean.110 representation.

Instances in the class are python-native representations of gt.dispatch.boolean.110 objects, while the actual gt.dispatch.boolean.110 object is the serialized UTF-8 byte string designed for sending in a message.

This method calls the as_dict() method, which differs from the native python dict() in the following key ways: - Enum Values: Translates between the values used locally by the actor to the symbol sent in messages. - - Removes any key-value pairs where the value is None for a clearer message, especially in cases with many optional attributes.

It also applies these changes recursively to sub-types.

Its near-inverse is GtDispatchBoolean.type_to_tuple(). If the type (or any sub-types) includes an enum, then the type_to_tuple will map an unrecognized symbol to the default enum value. This is why these two methods are only 'near' inverses.

Return type
bytes

AboutNodeName:

- Description: The Spaceheat Node getting dispatched.
- Format: LeftRightDot

ToGNodeAlias:

- Description: GNodeAlias of the SCADA.
- Format: LeftRightDot

FromGNodeAlias:

- Description: GNodeAlias of AtomicTNode.
- Format: LeftRightDot

FromGNodeInstanceId:

- Description: GNodeInstance of the AtomicTNode.
- Format: UuidCanonicalTextual

RelayState:

- Description: Relay State (0 or 1). A Relay State of *0* indicates the relay is OPEN (off). A Relay State of *1* indicates the relay is CLOSED (on). Note that *0* means the relay is open whether or not the relay is normally open or normally closed (For a normally open relay, the relay is ENERGIZED when it is in state *0* and DE-ENERGIZED when it is in state *1*.)

- Format: Bit

SendTimeUnixMs:

- Description: Time the AtomicTNode sends the dispatch, by its clock.
- Format: ReasonableUnixTimeMs

TypeName:

- Description: All GridWorks Versioned Types have a fixed TypeName, which is a string of lowercase alphanumeric words separated by periods, most significant word (on the left) starting with an alphabet character, and final word NOT all Hindu-Arabic numerals.

Version:

- Description: All GridWorks Versioned Types have a fixed version, which is a string of three Hindu-Arabic numerals.

class gwproto.types.gt_dispatch_boolean.**check_is_uuid_canonical_textual**(*v*)

Checks UuidCanonicalTextual format

UuidCanonicalTextual format: A string of hex words separated by hyphens of length 8-4-4-4-12.

Parameters
v (*str*) – the candidate

Raises
ValueError – if v is not UuidCanonicalTextual format

class gwproto.types.gt_dispatch_boolean.**check_is_bit**(*v*)

Checks Bit format

Bit format: The value must be the integer 0 or the integer 1.

Will not attempt to first interpret as an integer. For example, 1.3 will not be interpreted as 1 but will raise an error.

Parameters
v (*int*) – the candidate

Raises
ValueError – if v is not 0 or 1

class gwproto.types.gt_dispatch_boolean.**check_is_left_right_dot**(*v*)

Checks LeftRightDot Format

LeftRightDot format: Lowercase alphanumeric words separated by periods, with the most significant word (on the left) starting with an alphabet character.

Parameters
v (*str*) – the candidate

Raises
ValueError – if v is not LeftRightDot format

class gwproto.types.gt_dispatch_boolean.**check_is_reasonable_unix_time_ms**(*v*)

Checks ReasonableUnixTimeMs format

ReasonableUnixTimeMs format: unix milliseconds between Jan 1 2000 and Jan 1 3000

Parameters
v (*int*) – the candidate

Raises
ValueError – if v is not ReasonableUnixTimeMs format

class gwproto.types.**GtDispatchBoolean_Maker**(*about_node_name*, *to_g_node_alias*, *from_g_node_alias*, *from_g_node_instance_id*, *relay_state*, *send_time_unix_ms*)

Parameters

- **about_node_name** (*str*)
- **to_g_node_alias** (*str*)
- **from_g_node_alias** (*str*)
- **from_g_node_instance_id** (*str*)
- **relay_state** (*int*)
- **send_time_unix_ms** (*int*)

classmethod **dict_to_tuple**(*d*)

Deserialize a dictionary representation of a gt.dispatch.boolean.110 message object into a GtDispatchBoolean python object for internal use.

This is the near-inverse of the GtDispatchBoolean.as_dict() method:

- Enums: translates between the symbols sent in messages between actors and

the values used by the actors internally once they've deserialized the messages.

- Types: recursively validates and deserializes sub-types.

Note that if a required attribute with a default value is missing in a dict, this method will raise a SchemaError. This differs from the pydantic BaseModel practice of auto-completing missing attributes with default values when they exist.

Parameters
d (*dict*) – the dictionary resulting from json.loads(t) for a serialized JSON type object t.

Raises
SchemaError – if the dict cannot be turned into a GtDispatchBoolean object.

Returns
GtDispatchBoolean

Return type
GtDispatchBoolean

classmethod **tuple_to_type**(*tpl*)

Given a Python class object, returns the serialized JSON type object.

Parameters
tpl (GtDispatchBoolean)

Return type
bytes

classmethod **type_to_tuple**(*t*)

Given a serialized JSON type object, returns the Python class object.

Parameters
t (*bytes*)

Return type
GtDispatchBoolean

1.5.11 GtDispatchBooleanLocal

Python pydantic class corresponding to json type *gt.dispatch.boolean.local*, version *110*.

class gwproto.types.**GtDispatchBooleanLocal**(*, *RelayState*, *AboutNodeName*, *FromNodeName*, *SendTimeUnixMs*, *TypeName='gt.dispatch.boolean.local'*, *Version='110'*)

Dispatch message sent locally by SCADA HomeAlone actor.

By Locally, this means sent without access to Internet. The HomeAlone actor must reside within the Local Area Network of the SCADA - typically it should reside on the same hardware.

Parameters

- **RelayState** (*int*)
- **AboutNodeName** (*str*)
- **FromNodeName** (*str*)
- **SendTimeUnixMs** (*int*)
- **TypeName** (*Literal['gt.dispatch.boolean.local']*)
- **Version** (*Literal['110']*)

as_dict()

Translate the object into a dictionary representation that can be serialized into a gt.dispatch.boolean.local.110 object.

This method prepares the object for serialization by the as_type method, creating a dictionary with key-value pairs that follow the requirements for an instance of the gt.dispatch.boolean.local.110 type. Unlike the standard python dict method, it makes the following substantive changes: - Enum Values: Translates between the values used locally by the actor to the symbol sent in messages. - Removes any key-value pairs where the value is None for a clearer message, especially in cases with many optional attributes.

It also applies these changes recursively to sub-types.

Return type
Dict[str, *Any*]

as_type()

Serialize to the gt.dispatch.boolean.local.110 representation.

Instances in the class are python-native representations of gt.dispatch.boolean.local.110 objects, while the actual gt.dispatch.boolean.local.110 object is the serialized UTF-8 byte string designed for sending in a message.

This method calls the as_dict() method, which differs from the native python dict() in the following key ways: - Enum Values: Translates between the values used locally by the actor to the symbol sent in messages. - - Removes any key-value pairs where the value is None for a clearer message, especially in cases with many optional attributes.

It also applies these changes recursively to sub-types.

Its near-inverse is GtDispatchBooleanLocal.type_to_tuple(). If the type (or any sub-types) includes an enum, then the type_to_tuple will map an unrecognized symbol to the default enum value. This is why these two methods are only 'near' inverses.

Return type
bytes

RelayState:

- Description: Relay State (0 or 1). A Relay State of *0* indicates the relay is OPEN (off). A Relay State of *1* indicates the relay is CLOSED (on). Note that *0* means the relay is open whether or not the relay is normally open or normally closed (For a normally open relay, the relay is ENERGIZED when it is in state *0* and DE-ENERGIZED when it is in state *1*.)
- Format: Bit

AboutNodeName:

- Description: About Node Name. The boolean actuator Spaceheat Node getting turned on or off.
- Format: LeftRightDot

FromNodeName:

- Description: From Node Name. The Spaceheat Node sending the command.
- Format: LeftRightDot

SendTimeUnixMs:

- Description: Send Time in Unix Milliseconds.
- Format: ReasonableUnixTimeMs

TypeName:

- Description: All GridWorks Versioned Types have a fixed TypeName, which is a string of lowercase alphanumeric words separated by periods, most significant word (on the left) starting with an alphabet character, and final word NOT all Hindu-Arabic numerals.

Version:

- Description: All GridWorks Versioned Types have a fixed version, which is a string of three Hindu-Arabic numerals.

`class gwproto.types.gt_dispatch_boolean_local.check_is_bit(v)`

Checks Bit format

Bit format: The value must be the integer 0 or the integer 1.

Will not attempt to first interpret as an integer. For example, 1.3 will not be interpreted as 1 but will raise an error.

Parameters
v (*int*) – the candidate

Raises
ValueError – if v is not 0 or 1

`class gwproto.types.gt_dispatch_boolean_local.check_is_left_right_dot(v)`

Checks LeftRightDot Format

LeftRightDot format: Lowercase alphanumeric words separated by periods, with the most significant word (on the left) starting with an alphabet character.

Parameters
v (*str*) – the candidate

Raises
ValueError – if v is not LeftRightDot format

class gwproto.types.gt_dispatch_boolean_local.**check_is_reasonable_unix_time_ms**(*v*)

Checks ReasonableUnixTimeMs format

ReasonableUnixTimeMs format: unix milliseconds between Jan 1 2000 and Jan 1 3000

Parameters
v (*int*) – the candidate

Raises
ValueError – if v is not ReasonableUnixTimeMs format

class gwproto.types.**GtDispatchBooleanLocal_Maker**(*relay_state*, *about_node_name*, *from_node_name*, *send_time_unix_ms*)

Parameters

- **relay_state** (*int*)
- **about_node_name** (*str*)
- **from_node_name** (*str*)
- **send_time_unix_ms** (*int*)

classmethod dict_to_tuple(*d*)

Deserialize a dictionary representation of a gt.dispatch.boolean.local.110 message object into a GtDispatchBooleanLocal python object for internal use.

This is the near-inverse of the GtDispatchBooleanLocal.as_dict() method:

- Enums: translates between the symbols sent in messages between actors and

the values used by the actors internally once they've deserialized the messages.

- Types: recursively validates and deserializes sub-types.

Note that if a required attribute with a default value is missing in a dict, this method will raise a SchemaError. This differs from the pydantic BaseModel practice of auto-completing missing attributes with default values when they exist.

Parameters
d (*dict*) – the dictionary resulting from json.loads(t) for a serialized JSON type object t.

Raises
SchemaError – if the dict cannot be turned into a GtDispatchBooleanLocal object.

Returns
GtDispatchBooleanLocal

Return type
GtDispatchBooleanLocal

classmethod tuple_to_type(*tpl*)

Given a Python class object, returns the serialized JSON type object.

Parameters
tpl (GtDispatchBooleanLocal)

Return type
bytes

classmethod type_to_tuple(*t*)

Given a serialized JSON type object, returns the Python class object.

Parameters
t (*bytes*)

Return type
GtDispatchBooleanLocal

1.5.12 GtDriverBooleanactuatorCmd

Python pydantic class corresponding to json type *gt.driver.booleanactuator.cmd*, version *100*.

class gwproto.types.**GtDriverBooleanactuatorCmd**(*, *RelayState*, *ShNodeAlias*, *CommandTimeUnixMs*, *TypeName='gt.driver.booleanactuator.cmd'*, *Version='100'*)

Boolean Actuator Driver Command.

The boolean actuator actor reports when it has sent an actuation command to its driver so that the SCADA can add this to information to be sent up to the AtomicTNode.

[More info](https://gridworks.readthedocs.io/en/latest/relay-state.html)

Parameters

- **RelayState** (*int*)
- **ShNodeAlias** (*str*)
- **CommandTimeUnixMs** (*int*)
- **TypeName** (*Literal['gt.driver.booleanactuator.cmd']*)
- **Version** (*Literal['100']*)

as_dict()

Translate the object into a dictionary representation that can be serialized into a gt.driver.booleanactuator.cmd.100 object.

This method prepares the object for serialization by the as_type method, creating a dictionary with key-value pairs that follow the requirements for an instance of the gt.driver.booleanactuator.cmd.100 type. Unlike the standard python dict method, it makes the following substantive changes: - Enum Values: Translates between the values used locally by the actor to the symbol sent in messages. - Removes any key-value pairs where the value is None for a clearer message, especially in cases with many optional attributes.

It also applies these changes recursively to sub-types.

Return type
Dict[str, *Any*]

as_type()

Serialize to the gt.driver.booleanactuator.cmd.100 representation.

Instances in the class are python-native representations of gt.driver.booleanactuator.cmd.100 objects, while the actual gt.driver.booleanactuator.cmd.100 object is the serialized UTF-8 byte string designed for sending in a message.

This method calls the as_dict() method, which differs from the native python dict() in the following key ways: - Enum Values: Translates between the values used locally by the actor to the symbol sent in messages. - - Removes any key-value pairs where the value is None for a clearer message, especially in cases with many optional attributes.

It also applies these changes recursively to sub-types.

Its near-inverse is GtDriverBooleanactuatorCmd.type_to_tuple(). If the type (or any sub-types) includes an enum, then the type_to_tuple will map an unrecognized symbol to the default enum value. This is why these two methods are only 'near' inverses.

Return type
bytes

RelayState:

- Description:
- Format: Bit

ShNodeAlias:

- Description:
- Format: LeftRightDot

CommandTimeUnixMs:

- Description:
- Format: ReasonableUnixTimeMs

TypeName:

- Description: All GridWorks Versioned Types have a fixed TypeName, which is a string of lowercase alphanumeric words separated by periods, most significant word (on the left) starting with an alphabet character, and final word NOT all Hindu-Arabic numerals.

Version:

- Description: All GridWorks Versioned Types have a fixed version, which is a string of three Hindu-Arabic numerals.

`class gwproto.types.gt_driver_booleanactuator_cmd.check_is_bit(v)`

Checks Bit format

Bit format: The value must be the integer 0 or the integer 1.

Will not attempt to first interpret as an integer. For example, 1.3 will not be interpreted as 1 but will raise an error.

Parameters
v (*int*) – the candidate

Raises
ValueError – if v is not 0 or 1

`class gwproto.types.gt_driver_booleanactuator_cmd.check_is_left_right_dot(v)`

Checks LeftRightDot Format

LeftRightDot format: Lowercase alphanumeric words separated by periods, with the most significant word (on the left) starting with an alphabet character.

Parameters
v (*str*) – the candidate

Raises
ValueError – if v is not LeftRightDot format

`class gwproto.types.gt_driver_booleanactuator_cmd.check_is_reasonable_unix_time_ms(v)`

Checks ReasonableUnixTimeMs format

ReasonableUnixTimeMs format: unix milliseconds between Jan 1 2000 and Jan 1 3000

Parameters
v (*int*) – the candidate

Raises
ValueError – if v is not ReasonableUnixTimeMs format

class gwproto.types.**GtDriverBooleanactuatorCmd_Maker**(*relay_state*, *sh_node_alias*, *command_time_unix_ms*)

Parameters

- **relay_state** (*int*)
- **sh_node_alias** (*str*)
- **command_time_unix_ms** (*int*)

classmethod **dict_to_tuple**(*d*)

Deserialize a dictionary representation of a gt.driver.booleanactuator.cmd.100 message object into a GtDriverBooleanactuatorCmd python object for internal use.

This is the near-inverse of the GtDriverBooleanactuatorCmd.as_dict() method:

- Enums: translates between the symbols sent in messages between actors and

the values used by the actors internally once they've deserialized the messages.

- Types: recursively validates and deserializes sub-types.

Note that if a required attribute with a default value is missing in a dict, this method will raise a SchemaError. This differs from the pydantic BaseModel practice of auto-completing missing attributes with default values when they exist.

Parameters
d (*dict*) – the dictionary resulting from json.loads(t) for a serialized JSON type object t.

Raises
SchemaError – if the dict cannot be turned into a GtDriverBooleanactuatorCmd object.

Returns
GtDriverBooleanactuatorCmd

Return type
GtDriverBooleanactuatorCmd

classmethod **tuple_to_type**(*tpl*)

Given a Python class object, returns the serialized JSON type object.

Parameters
tpl (GtDriverBooleanactuatorCmd)

Return type
bytes

classmethod **type_to_tuple**(*t*)

Given a serialized JSON type object, returns the Python class object.

Parameters
t (*bytes*)

Return type
GtDriverBooleanactuatorCmd

1.5.13 GtShBooleanactuatorCmdStatus

Python pydantic class corresponding to json type *gt.sh.booleanactuator.cmd.status*, version *100*.

class gwproto.types.**GtShBooleanactuatorCmdStatus**(*, *ShNodeAlias*, *RelayStateCommandList*, *CommandTimeUnixMsList*, *TypeName='gt.sh.booleanactuator.cmd.status'*, *Version='100'*)

Boolean Actuator Driver Command Status Package.

This is a subtype of the status message sent from a SCADA to its AtomicTNode. It contains a list of all the commands that a particular boolean actuator actor has reported as sending as actuation commands to its driver in the last transmission period (typically 5 minutes).

[More info](https://gridworks.readthedocs.io/en/latest/relay-state.html)

Parameters

- **ShNodeAlias** (*str*)
- **RelayStateCommandList** (*List[int]*)
- **CommandTimeUnixMsList** (*List[int]*)
- **TypeName** (*Literal['gt.sh.booleanactuator.cmd.status']*)
- **Version** (*Literal['100']*)

as_dict()

Translate the object into a dictionary representation that can be serialized into a gt.sh.booleanactuator.cmd.status.100 object.

This method prepares the object for serialization by the as_type method, creating a dictionary with key-value pairs that follow the requirements for an instance of the gt.sh.booleanactuator.cmd.status.100 type. Unlike the standard python dict method, it makes the following substantive changes: - Enum Values: Translates between the values used locally by the actor to the symbol sent in messages. - Removes any key-value pairs where the value is None for a clearer message, especially in cases with many optional attributes.

It also applies these changes recursively to sub-types.

Return type
Dict[str, *Any*]

as_type()

Serialize to the gt.sh.booleanactuator.cmd.status.100 representation.

Instances in the class are python-native representations of gt.sh.booleanactuator.cmd.status.100 objects, while the actual gt.sh.booleanactuator.cmd.status.100 object is the serialized UTF-8 byte string designed for sending in a message.

This method calls the as_dict() method, which differs from the native python dict() in the following key ways: - Enum Values: Translates between the values used locally by the actor to the symbol sent in messages. - - Removes any key-value pairs where the value is None for a clearer message, especially in cases with many optional attributes.

It also applies these changes recursively to sub-types.

Its near-inverse is GtShBooleanactuatorCmdStatus.type_to_tuple(). If the type (or any sub-types) includes an enum, then the type_to_tuple will map an unrecognized symbol to the default enum value. This is why these two methods are only 'near' inverses.

Return type
bytes

ShNodeAlias:

- Description: SpaceheatNodeAlias. The alias of the spaceheat node that is getting actuated. For example, *a.elt1.relay* would likely indicate the relay for a resistive element.
- Format: LeftRightDot

RelayStateCommandList:

- Description: List of RelayStateCommands.

CommandTimeUnixMsList:

- Description: List of Command Times.
- Format: ReasonableUnixTimeMs

TypeName:

- Description: All GridWorks Versioned Types have a fixed TypeName, which is a string of lowercase alphanumeric words separated by periods, most significant word (on the left) starting with an alphabet character, and final word NOT all Hindu-Arabic numerals.

Version:

- Description: All GridWorks Versioned Types have a fixed version, which is a string of three Hindu-Arabic numerals.

`class` gwproto.types.gt_sh_booleanactuator_cmd_status.**check_is_left_right_dot**(*v*)

Checks LeftRightDot Format

LeftRightDot format: Lowercase alphanumeric words separated by periods, with the most significant word (on the left) starting with an alphabet character.

Parameters
v (*str*) – the candidate

Raises
ValueError – if v is not LeftRightDot format

`class` gwproto.types.gt_sh_booleanactuator_cmd_status.**check_is_reasonable_unix_time_ms**(*v*)

Checks ReasonableUnixTimeMs format

ReasonableUnixTimeMs format: unix milliseconds between Jan 1 2000 and Jan 1 3000

Parameters
v (*int*) – the candidate

Raises
ValueError – if v is not ReasonableUnixTimeMs format

`class` gwproto.types.**GtShBooleanactuatorCmdStatus_Maker**(*sh_node_alias*, *relay_state_command_list*, *command_time_unix_ms_list*)

Parameters

- **sh_node_alias** (*str*)
- **relay_state_command_list** (*List[int]*)
- **command_time_unix_ms_list** (*List[int]*)

classmethod dict_to_tuple(*d*)

Deserialize a dictionary representation of a gt.sh.booleanactuator.cmd.status.100 message object into a GtShBooleanactuatorCmdStatus python object for internal use.

This is the near-inverse of the GtShBooleanactuatorCmdStatus.as_dict() method:

- Enums: translates between the symbols sent in messages between actors and

the values used by the actors internally once they've deserialized the messages.

- Types: recursively validates and deserializes sub-types.

Note that if a required attribute with a default value is missing in a dict, this method will raise a SchemaError. This differs from the pydantic BaseModel practice of auto-completing missing attributes with default values when they exist.

Parameters
d (*dict*) – the dictionary resulting from json.loads(t) for a serialized JSON type object t.

Raises
SchemaError – if the dict cannot be turned into a GtShBooleanactuatorCmdStatus object.

Returns
GtShBooleanactuatorCmdStatus

Return type
GtShBooleanactuatorCmdStatus

classmethod tuple_to_type(*tpl*)

Given a Python class object, returns the serialized JSON type object.

Parameters
tpl (GtShBooleanactuatorCmdStatus)

Return type
bytes

classmethod type_to_tuple(*t*)

Given a serialized JSON type object, returns the Python class object.

Parameters
t (*bytes*)

Return type
GtShBooleanactuatorCmdStatus

1.5.14 GtShCliAtnCmd

Python pydantic class corresponding to json type *gt.sh.cli.atn.cmd*, version *110*.

class gwproto.types.**GtShCliAtnCmd**(*, *FromGNodeAlias*, *SendSnapshot*, *FromGNodeId*, *TypeName='gt.sh.cli.atn.cmd'*, *Version='110'*)

AtomicTNode CLI Command.

This is a generic type mechanism for a crude command line interface on a SCADA, brokered by the AtomicTNode.

Parameters

- **FromGNodeAlias** (*str*)
- **SendSnapshot** (*bool*)
- **FromGNodeId** (*str*)
- **TypeName** (*Literal['gt.sh.cli.atn.cmd']*)

- **Version** (*Literal['110']*)

as_dict()

Translate the object into a dictionary representation that can be serialized into a gt.sh.cli.atn.cmd.110 object.

This method prepares the object for serialization by the as_type method, creating a dictionary with key-value pairs that follow the requirements for an instance of the gt.sh.cli.atn.cmd.110 type. Unlike the standard python dict method, it makes the following substantive changes: - Enum Values: Translates between the values used locally by the actor to the symbol sent in messages. - Removes any key-value pairs where the value is None for a clearer message, especially in cases with many optional attributes.

It also applies these changes recursively to sub-types.

Return type
Dict[str, *Any*]

as_type()

Serialize to the gt.sh.cli.atn.cmd.110 representation.

Instances in the class are python-native representations of gt.sh.cli.atn.cmd.110 objects, while the actual gt.sh.cli.atn.cmd.110 object is the serialized UTF-8 byte string designed for sending in a message.

This method calls the as_dict() method, which differs from the native python dict() in the following key ways: - Enum Values: Translates between the values used locally by the actor to the symbol sent in messages. - - Removes any key-value pairs where the value is None for a clearer message, especially in cases with many optional attributes.

It also applies these changes recursively to sub-types.

Its near-inverse is GtShCliAtnCmd.type_to_tuple(). If the type (or any sub-types) includes an enum, then the type_to_tuple will map an unrecognized symbol to the default enum value. This is why these two methods are only 'near' inverses.

Return type
bytes

FromGNodeAlias:

- Description: GNodeAlias. Must be the SCADA's AtomicTNode.
- Format: LeftRightDot

SendSnapshot:

- Description: Send Snapshot. Asks SCADA to send back a snapshot. For this version of the type, nothing would happen if SendSnapshot were set to False. However, we include this in case additional variations are added later.

FromGNodeId:

- Description: GNodeId.
- Format: UuidCanonicalTextual

TypeName:

- Description: All GridWorks Versioned Types have a fixed TypeName, which is a string of lowercase alphanumeric words separated by periods, most significant word (on the left) starting with an alphabet character, and final word NOT all Hindu-Arabic numerals.

Version:

- Description: All GridWorks Versioned Types have a fixed version, which is a string of three Hindu-Arabic numerals.

class gwproto.types.gt_sh_cli_atn_cmd.**check_is_uuid_canonical_textual**(*v*)

Checks UuidCanonicalTextual format

UuidCanonicalTextual format: A string of hex words separated by hyphens of length 8-4-4-4-12.

Parameters
v (*str*) – the candidate

Raises
ValueError – if v is not UuidCanonicalTextual format

class gwproto.types.gt_sh_cli_atn_cmd.**check_is_left_right_dot**(*v*)

Checks LeftRightDot Format

LeftRightDot format: Lowercase alphanumeric words separated by periods, with the most significant word (on the left) starting with an alphabet character.

Parameters
v (*str*) – the candidate

Raises
ValueError – if v is not LeftRightDot format

class gwproto.types.**GtShCliAtnCmd_Maker**(*from_g_node_alias*, *send_snapshot*, *from_g_node_id*)

Parameters

- **from_g_node_alias** (*str*)
- **send_snapshot** (*bool*)
- **from_g_node_id** (*str*)

classmethod dict_to_tuple(*d*)

Deserialize a dictionary representation of a gt.sh.cli.atn.cmd.110 message object into a GtShCliAtnCmd python object for internal use.

This is the near-inverse of the GtShCliAtnCmd.as_dict() method:

- Enums: translates between the symbols sent in messages between actors and

the values used by the actors internally once they've deserialized the messages.

- Types: recursively validates and deserializes sub-types.

Note that if a required attribute with a default value is missing in a dict, this method will raise a SchemaError. This differs from the pydantic BaseModel practice of auto-completing missing attributes with default values when they exist.

Parameters
d (*dict*) – the dictionary resulting from json.loads(t) for a serialized JSON type object t.

Raises
SchemaError – if the dict cannot be turned into a GtShCliAtnCmd object.

Returns
GtShCliAtnCmd

Return type
GtShCliAtnCmd

classmethod tuple_to_type(*tpl*)

Given a Python class object, returns the serialized JSON type object.

Parameters
tpl (GtShCliAtnCmd)

Return type
bytes

classmethod type_to_tuple(*t*)

Given a serialized JSON type object, returns the Python class object.

Parameters
t (*bytes*)

Return type
GtShCliAtnCmd

1.5.15 GtShMultipurposeTelemetryStatus

Python pydantic class corresponding to json type *gt.sh.multipurpose.telemetry.status*, version *100*.

class gwproto.types.**GtShMultipurposeTelemetryStatus**(*, *AboutNodeAlias*, *SensorNodeAlias*, *TelemetryName*, *ValueList*, *ReadTimeUnixMsList*, *TypeName='gt.sh.multipurpose.telemetry.status'*, *Version='100'*)

Data read from a MultipurposeSensor run by a Spaceheat SCADA.

A list of readings about a specific SpaceheatNode made by a MultipurposeSensor node, for a Spaceheat SCADA. Designed as part of a status message sent from the SCADA to its AtomicTNode typically once every 5 minutes. The nth element of each of its two lists refer to the same reading (i.e. what the value is, when it was read).

[More info](https://gridworks-protocol.readthedocs.io/en/latest/multipurpose-sensor.html)

Parameters

- **AboutNodeAlias** (*str*)
- **SensorNodeAlias** (*str*)
- **TelemetryName** (TelemetryName)
- **ValueList** (*List[int]*)
- **ReadTimeUnixMsList** (*List[int]*)
- **TypeName** (*Literal['gt.sh.multipurpose.telemetry.status']*)
- **Version** (*Literal['100']*)

as_dict()

Translate the object into a dictionary representation that can be serialized into a gt.sh.multipurpose.telemetry.status.100 object.

This method prepares the object for serialization by the as_type method, creating a dictionary with key-value pairs that follow the requirements for an instance of the gt.sh.multipurpose.telemetry.status.100 type. Unlike the standard python dict method, it makes the following substantive changes: - Enum Values: Translates between the values used locally by the actor to the symbol sent in messages. - Removes any key-value pairs where the value is None for a clearer message, especially in cases with many optional attributes.

It also applies these changes recursively to sub-types.

Return type
Dict[str, *Any*]

as_type()

Serialize to the gt.sh.multipurpose.telemetry.status.100 representation.

Instances in the class are python-native representations of gt.sh.multipurpose.telemetry.status.100 objects, while the actual gt.sh.multipurpose.telemetry.status.100 object is the serialized UTF-8 byte string designed for sending in a message.

This method calls the as_dict() method, which differs from the native python dict() in the following key ways: - Enum Values: Translates between the values used locally by the actor to the symbol sent in messages. - - Removes any key-value pairs where the value is None for a clearer message, especially in cases with many optional attributes.

It also applies these changes recursively to sub-types.

Its near-inverse is GtShMultipurposeTelemetryStatus.type_to_tuple(). If the type (or any sub-types) includes an enum, then the type_to_tuple will map an unrecognized symbol to the default enum value. This is why these two methods are only 'near' inverses.

Return type
bytes

classmethod check_axiom_1(*v*)

Axiom 1: ListLengthConsistency. ValueList and ReadTimeUnixMsList must have the same length.

Parameters
v (*dict*)

Return type
dict

AboutNodeAlias:

- Description: AboutNodeAlias. The SpaceheatNode representing the physical object that the sensor reading is collecting data about. For example, a multipurpose temp sensor that reads 12 temperatures would have data for 12 different AboutNodeAliases, including say *a.tank1.temp1* for a temp sensor at the top of a water tank.
- Format: LeftRightDot

SensorNodeAlias:

- Description: SensorNodeAlias. The alias of the SpaceheatNode representing the telemetry device

TelemetryName:

- Description: TelemetryName. The TelemetryName of the readings. This is used to interpet the meaning of the reading values. For example, WaterTempCTimes1000 means the reading is measuring the a reading of 37 deg C.

ValueList:

- Description: List of Values. The values of the readings.

ReadTimeUnixMsList:

- Description: List of Read Times. The times that the MultipurposeSensor took the readings, in unix milliseconds
- Format: ReasonableUnixTimeMs

TypeName:

- Description: All GridWorks Versioned Types have a fixed TypeName, which is a string of lowercase alphanumeric words separated by periods, most significant word (on the left) starting with an alphabet character, and final word NOT all Hindu-Arabic numerals.

Version:

- Description: All GridWorks Versioned Types have a fixed version, which is a string of three Hindu-Arabic numerals.

class gwproto.types.gt_sh_multipurpose_telemetry_status.**check_is_left_right_dot**(*v*)

Checks LeftRightDot Format

LeftRightDot format: Lowercase alphanumeric words separated by periods, with the most significant word (on the left) starting with an alphabet character.

Parameters
v (*str*) – the candidate

Raises
ValueError – if v is not LeftRightDot format

class gwproto.types.gt_sh_multipurpose_telemetry_status.**check_is_reasonable_unix_time_ms**(*v*)

Checks ReasonableUnixTimeMs format

ReasonableUnixTimeMs format: unix milliseconds between Jan 1 2000 and Jan 1 3000

Parameters
v (*int*) – the candidate

Raises
ValueError – if v is not ReasonableUnixTimeMs format

class gwproto.types.**GtShMultipurposeTelemetryStatus_Maker**(*about_node_alias*, *sensor_node_alias*, *telemetry_name*, *value_list*, *read_time_unix_ms_list*)

Parameters

- **about_node_alias** (*str*)
- **sensor_node_alias** (*str*)
- **telemetry_name** (TelemetryName)
- **value_list** (*List[int]*)
- **read_time_unix_ms_list** (*List[int]*)

classmethod dict_to_tuple(*d*)

Deserialize a dictionary representation of a gt.sh.multipurpose.telemetry.status.100 message object into a GtShMultipurposeTelemetryStatus python object for internal use.

This is the near-inverse of the GtShMultipurposeTelemetryStatus.as_dict() method:

- Enums: translates between the symbols sent in messages between actors and

the values used by the actors internally once they've deserialized the messages.

- Types: recursively validates and deserializes sub-types.

Note that if a required attribute with a default value is missing in a dict, this method will raise a SchemaError. This differs from the pydantic BaseModel practice of auto-completing missing attributes with default values when they exist.

Parameters
d (*dict*) – the dictionary resulting from json.loads(t) for a serialized JSON type object t.

Raises
SchemaError – if the dict cannot be turned into a GtShMultipurposeTelemetryStatus object.

Returns
GtShMultipurposeTelemetryStatus

Return type
GtShMultipurposeTelemetryStatus

classmethod tuple_to_type(*tpl*)

Given a Python class object, returns the serialized JSON type object.

Parameters
tpl (GtShMultipurposeTelemetryStatus)

Return type
bytes

classmethod type_to_tuple(*t*)

Given a serialized JSON type object, returns the Python class object.

Parameters
t (*bytes*)

Return type
GtShMultipurposeTelemetryStatus

1.5.16 GtShSimpleTelemetryStatus

Python pydantic class corresponding to json type *gt.sh.simple.telemetry.status*, version *100*.

class gwproto.types.**GtShSimpleTelemetryStatus**(**, ShNodeAlias, TelemetryName, ValueList, ReadTimeUnixMsList, TypeName='gt.sh.simple.telemetry.status', Version='100'*)

Data read from a SimpleSensor run by a SpaceHeat SCADA.

A list of readings from a simple sensor for a Spaceheat SCADA. Designed as part of a status message sent from the SCADA to its AtomicTNode typically once every 5 minutes. The nth element of each of its two lists refer to the same reading (i.e. what the value is, when it was read).

[More info](https://gridworks-protocol.readthedocs.io/en/latest/simple-sensor.html)

Parameters

- **ShNodeAlias** (*str*)
- **TelemetryName** (TelemetryName)
- **ValueList** (*List[int]*)
- **ReadTimeUnixMsList** (*List[int]*)
- **TypeName** (*Literal['gt.sh.simple.telemetry.status']*)
- **Version** (*Literal['100']*)

as_dict()

Translate the object into a dictionary representation that can be serialized into a gt.sh.simple.telemetry.status.100 object.

This method prepares the object for serialization by the as_type method, creating a dictionary with key-value pairs that follow the requirements for an instance of the gt.sh.simple.telemetry.status.100 type. Unlike the standard python dict method, it makes the following substantive changes: - Enum Values: Translates

between the values used locally by the actor to the symbol sent in messages. - Removes any key-value pairs where the value is None for a clearer message, especially in cases with many optional attributes.

It also applies these changes recursively to sub-types.

Return type
Dict[str, *Any*]

`as_type()`

Serialize to the gt.sh.simple.telemetry.status.100 representation.

Instances in the class are python-native representations of gt.sh.simple.telemetry.status.100 objects, while the actual gt.sh.simple.telemetry.status.100 object is the serialized UTF-8 byte string designed for sending in a message.

This method calls the as_dict() method, which differs from the native python dict() in the following key ways: - Enum Values: Translates between the values used locally by the actor to the symbol sent in messages. - - Removes any key-value pairs where the value is None for a clearer message, especially in cases with many optional attributes.

It also applies these changes recursively to sub-types.

Its near-inverse is GtShSimpleTelemetryStatus.type_to_tuple(). If the type (or any sub-types) includes an enum, then the type_to_tuple will map an unrecognized symbol to the default enum value. This is why these two methods are only 'near' inverses.

Return type
bytes

`classmethod check_axiom_1(v)`

Axiom 1: ListLengthConsistency. ValueList and ReadTimeUnixMsList must have the same length.

Parameters
v (*dict*)

Return type
dict

ShNodeAlias:

- Description: SpaceheatNodeAlias. The Alias of the SimpleSensor associated to the readings
- Format: LeftRightDot

TelemetryName:

- **Description: TelemetryName. The TelemetryName of the readings. This is used to interpet the meaning of the**
  reading values. For example, WaterTempCTimes1000 means the reading is measuring the temperature of water, in Celsius multiplied by 1000. So a value of 37000 would be a reading of 37 deg C.

ValueList:

- Description: List of Values. The values of the readings.

ReadTimeUnixMsList:

- Description: List of Read Times. The times that the SImpleSensor took the readings, in unix milliseconds
- Format: ReasonableUnixTimeMs

TypeName:

- Description: All GridWorks Versioned Types have a fixed TypeName, which is a string of lowercase alphanumeric words separated by periods, most significant word (on the left) starting with an alphabet character, and final word NOT all Hindu-Arabic numerals.

Version:

- Description: All GridWorks Versioned Types have a fixed version, which is a string of three Hindu-Arabic numerals.

class gwproto.types.gt_sh_simple_telemetry_status.**check_is_left_right_dot**(*v*)

Checks LeftRightDot Format

LeftRightDot format: Lowercase alphanumeric words separated by periods, with the most significant word (on the left) starting with an alphabet character.

Parameters
v (*str*) – the candidate

Raises
ValueError – if v is not LeftRightDot format

class gwproto.types.gt_sh_simple_telemetry_status.**check_is_reasonable_unix_time_ms**(*v*)

Checks ReasonableUnixTimeMs format

ReasonableUnixTimeMs format: unix milliseconds between Jan 1 2000 and Jan 1 3000

Parameters
v (*int*) – the candidate

Raises
ValueError – if v is not ReasonableUnixTimeMs format

class gwproto.types.**GtShSimpleTelemetryStatus_Maker**(*sh_node_alias*, *telemetry_name*, *value_list*, *read_time_unix_ms_list*)

Parameters

- **sh_node_alias** (*str*)
- **telemetry_name** (TelemetryName)
- **value_list** (*List[int]*)
- **read_time_unix_ms_list** (*List[int]*)

classmethod dict_to_tuple(*d*)

Deserialize a dictionary representation of a gt.sh.simple.telemetry.status.100 message object into a GtShSimpleTelemetryStatus python object for internal use.

This is the near-inverse of the GtShSimpleTelemetryStatus.as_dict() method:

- Enums: translates between the symbols sent in messages between actors and

the values used by the actors internally once they've deserialized the messages.

- Types: recursively validates and deserializes sub-types.

Note that if a required attribute with a default value is missing in a dict, this method will raise a SchemaError. This differs from the pydantic BaseModel practice of auto-completing missing attributes with default values when they exist.

Parameters
d (*dict*) – the dictionary resulting from json.loads(t) for a serialized JSON type object t.

Raises
SchemaError – if the dict cannot be turned into a GtShSimpleTelemetryStatus object.

Returns
GtShSimpleTelemetryStatus

Return type
GtShSimpleTelemetryStatus

classmethod tuple_to_type(*tpl*)

Given a Python class object, returns the serialized JSON type object.

Parameters
tpl (GtShSimpleTelemetryStatus)

Return type
bytes

classmethod type_to_tuple(*t*)

Given a serialized JSON type object, returns the Python class object.

Parameters
t (*bytes*)

Return type
GtShSimpleTelemetryStatus

1.5.17 GtShStatus

Python pydantic class corresponding to json type *gt.sh.status*, version *110*.

class gwproto.types.**GtShStatus**(**, FromGNodeAlias, FromGNodeId, AboutGNodeAlias, SlotStartUnixS, ReportingPeriodS, SimpleTelemetryList, MultipurposeTelemetryList, BooleanactuatorCmdList, StatusUid, TypeName='gt.sh.status', Version='110'*)

Status message sent by a Spaceheat SCADA every 5 minutes

Parameters

- **FromGNodeAlias** (*str*)
- **FromGNodeId** (*str*)
- **AboutGNodeAlias** (*str*)
- **SlotStartUnixS** (*int*)
- **ReportingPeriodS** (*int*)
- **SimpleTelemetryList** (*List[GtShSimpleTelemetryStatus]*)
- **MultipurposeTelemetryList** (*List[GtShMultipurposeTelemetryStatus]*)
- **BooleanactuatorCmdList** (*List[GtShBooleanactuatorCmdStatus]*)
- **StatusUid** (*str*)
- **TypeName** (*Literal['gt.sh.status']*)
- **Version** (*Literal['110']*)

`as_dict()`

Translate the object into a dictionary representation that can be serialized into a gt.sh.status.110 object.

This method prepares the object for serialization by the as_type method, creating a dictionary with key-value pairs that follow the requirements for an instance of the gt.sh.status.110 type. Unlike the standard python dict method, it makes the following substantive changes: - Enum Values: Translates between the values used locally by the actor to the symbol sent in messages. - Removes any key-value pairs where the value is None for a clearer message, especially in cases with many optional attributes.

It also applies these changes recursively to sub-types.

Return type
Dict[str, *Any*]

`as_type()`

Serialize to the gt.sh.status.110 representation.

Instances in the class are python-native representations of gt.sh.status.110 objects, while the actual gt.sh.status.110 object is the serialized UTF-8 byte string designed for sending in a message.

This method calls the as_dict() method, which differs from the native python dict() in the following key ways: - Enum Values: Translates between the values used locally by the actor to the symbol sent in messages. - - Removes any key-value pairs where the value is None for a clearer message, especially in cases with many optional attributes.

It also applies these changes recursively to sub-types.

Its near-inverse is GtShStatus.type_to_tuple(). If the type (or any sub-types) includes an enum, then the type_to_tuple will map an unrecognized symbol to the default enum value. This is why these two methods are only 'near' inverses.

Return type
bytes

FromGNodeAlias:

- Description:
- Format: LeftRightDot

FromGNodeId:

- Description:
- Format: UuidCanonicalTextual

AboutGNodeAlias:

- Description:
- Format: LeftRightDot

SlotStartUnixS:

- Description:
- Format: ReasonableUnixTimeS

ReportingPeriodS:

- Description:

SimpleTelemetryList:

- Description:

MultipurposeTelemetryList:

- Description:

BooleanactuatorCmdList:

- Description:

StatusUid:

- Description:
- Format: UuidCanonicalTextual

TypeName:

- Description: All GridWorks Versioned Types have a fixed TypeName, which is a string of lowercase alphanumeric words separated by periods, most significant word (on the left) starting with an alphabet character, and final word NOT all Hindu-Arabic numerals.

Version:

- Description: All GridWorks Versioned Types have a fixed version, which is a string of three Hindu-Arabic numerals.

class gwproto.types.gt_sh_status.**check_is_reasonable_unix_time_s**(*v*)

Checks ReasonableUnixTimeS format

ReasonableUnixTimeS format: unix seconds between Jan 1 2000 and Jan 1 3000

Parameters
v (*int*) – the candidate

Raises
ValueError – if v is not ReasonableUnixTimeS format

class gwproto.types.gt_sh_status.**check_is_uuid_canonical_textual**(*v*)

Checks UuidCanonicalTextual format

UuidCanonicalTextual format: A string of hex words separated by hyphens of length 8-4-4-4-12.

Parameters
v (*str*) – the candidate

Raises
ValueError – if v is not UuidCanonicalTextual format

class gwproto.types.gt_sh_status.**check_is_left_right_dot**(*v*)

Checks LeftRightDot Format

LeftRightDot format: Lowercase alphanumeric words separated by periods, with the most significant word (on the left) starting with an alphabet character.

Parameters
v (*str*) – the candidate

Raises
ValueError – if v is not LeftRightDot format

class gwproto.types.**GtShStatus_Maker**(*from_g_node_alias*, *from_g_node_id*, *about_g_node_alias*, *slot_start_unix_s*, *reporting_period_s*, *simple_telemetry_list*, *multipurpose_telemetry_list*, *booleanactuator_cmd_list*, *status_uid*)

Parameters

- **from_g_node_alias** (*str*)
- **from_g_node_id** (*str*)
- **about_g_node_alias** (*str*)
- **slot_start_unix_s** (*int*)
- **reporting_period_s** (*int*)
- **simple_telemetry_list** (*List[GtShSimpleTelemetryStatus]*)
- **multipurpose_telemetry_list** (*List[GtShMultipurposeTelemetryStatus]*)
- **booleanactuator_cmd_list** (*List[GtShBooleanactuatorCmdStatus]*)
- **status_uid** (*str*)

classmethod dict_to_tuple(*d*)

Deserialize a dictionary representation of a gt.sh.status.110 message object into a GtShStatus python object for internal use.

This is the near-inverse of the GtShStatus.as_dict() method:

- Enums: translates between the symbols sent in messages between actors and

the values used by the actors internally once they've deserialized the messages.

- Types: recursively validates and deserializes sub-types.

Note that if a required attribute with a default value is missing in a dict, this method will raise a SchemaError. This differs from the pydantic BaseModel practice of auto-completing missing attributes with default values when they exist.

Parameters
d (*dict*) – the dictionary resulting from json.loads(t) for a serialized JSON type object t.

Raises
SchemaError – if the dict cannot be turned into a GtShStatus object.

Returns
GtShStatus

Return type
GtShStatus

classmethod tuple_to_type(*tpl*)

Given a Python class object, returns the serialized JSON type object.

Parameters
tpl (GtShStatus)

Return type
bytes

classmethod type_to_tuple(*t*)

Given a serialized JSON type object, returns the Python class object.

Parameters
t (*bytes*)

Return type
GtShStatus

1.5.18 GtShTelemetryFromMultipurposeSensor

Python pydantic class corresponding to json type *gt.sh.telemetry.from.multipurpose.sensor*, version *100*.

class gwproto.types.**GtShTelemetryFromMultipurposeSensor**(*, *ScadaReadTimeUnixMs*, *AboutNodeAliasList*, *TelemetryNameList*, *ValueList*, *TypeName='gt.sh.telemetry.from.multipurpose.sensor'*, *Version='100'*)

Data sent from a MultipurposeSensor to a Spaceheat SCADA.

A set of readings made at the same time by a multipurpose sensor, sent by the MultipurposeSensor SpaceheatNode actor to its SCADA. The nth element of each of its three readings (what is getting read, what the value is, what the TelemetryNames are).

[More info](https://gridworks-protocol.readthedocs.io/en/latest/multipurpose-sensor.html)

Parameters

- **ScadaReadTimeUnixMs** (*int*)
- **AboutNodeAliasList** (*List[str]*)
- **TelemetryNameList** (*List[TelemetryName]*)
- **ValueList** (*List[int]*)
- **TypeName** (*Literal['gt.sh.telemetry.from.multipurpose.sensor']*)
- **Version** (*Literal['100']*)

as_dict()

Translate the object into a dictionary representation that can be serialized into a gt.sh.telemetry.from.multipurpose.sensor.100 object.

This method prepares the object for serialization by the as_type method, creating a dictionary with key-value pairs that follow the requirements for an instance of the gt.sh.telemetry.from.multipurpose.sensor.100 type. Unlike the standard python dict method, it makes the following substantive changes: - Enum Values: Translates between the values used locally by the actor to the symbol sent in messages. - Removes any key-value pairs where the value is None for a clearer message, especially in cases with many optional attributes.

It also applies these changes recursively to sub-types.

Return type
Dict[str, *Any*]

as_type()

Serialize to the gt.sh.telemetry.from.multipurpose.sensor.100 representation.

Instances in the class are python-native representations of gt.sh.telemetry.from.multipurpose.sensor.100 objects, while the actual gt.sh.telemetry.from.multipurpose.sensor.100 object is the serialized UTF-8 byte string designed for sending in a message.

This method calls the as_dict() method, which differs from the native python dict() in the following key ways: - Enum Values: Translates between the values used locally by the actor to the symbol sent in messages. - - Removes any key-value pairs where the value is None for a clearer message, especially in cases with many optional attributes.

It also applies these changes recursively to sub-types.

Its near-inverse is GtShTelemetryFromMultipurposeSensor.type_to_tuple(). If the type (or any sub-types) includes an enum, then the type_to_tuple will map an unrecognized symbol to the default enum value. This is why these two methods are only 'near' inverses.

Return type
bytes

classmethod check_axiom_1(*v*)

Axiom 1: ListLengthConsistency. AboutNodeAliasList, ValueList and TelemetryNameList must all have the same length.

Parameters
v (*dict*)

Return type
dict

ScadaReadTimeUnixMs:

- Description: ScadaReadTime in Unix MilliSeconds.
- Format: ReasonableUnixTimeMs

AboutNodeAliasList:

- Description: AboutNodeAliasList. List of aliases of the SpaceHeat Nodes getting measured
- Format: LeftRightDot

TelemetryNameList:

- Description: TelemetryNameList. List of the TelemetryNames. The nth name in this list indicates the TelemetryName of the nth alias in the AboutNodeAliasList.

ValueList:

- Description: ValueList.

TypeName:

- Description: All GridWorks Versioned Types have a fixed TypeName, which is a string of lowercase alphanumeric words separated by periods, most significant word (on the left) starting with an alphabet character, and final word NOT all Hindu-Arabic numerals.

Version:

- Description: All GridWorks Versioned Types have a fixed version, which is a string of three Hindu-Arabic numerals.

class gwproto.types.gt_sh_telemetry_from_multipurpose_sensor.check_is_left_right_dot(*v*)

Checks LeftRightDot Format

LeftRightDot format: Lowercase alphanumeric words separated by periods, with the most significant word (on the left) starting with an alphabet character.

Parameters
v (*str*) – the candidate

Raises
ValueError – if v is not LeftRightDot format

class gwproto.types.gt_sh_telemetry_from_multipurpose_sensor.check_is_reasonable_unix_time_ms(*v*)

Checks ReasonableUnixTimeMs format

ReasonableUnixTimeMs format: unix milliseconds between Jan 1 2000 and Jan 1 3000

Parameters
v (*int*) – the candidate

Raises
ValueError – if v is not ReasonableUnixTimeMs format

class gwproto.types.**GtShTelemetryFromMultipurposeSensor_Maker**(*scada_read_time_unix_ms*, *about_node_alias_list*, *telemetry_name_list*, *value_list*)

Parameters

- **scada_read_time_unix_ms** (*int*)
- **about_node_alias_list** (*List[str]*)
- **telemetry_name_list** (*List[TelemetryName]*)
- **value_list** (*List[int]*)

classmethod **dict_to_tuple**(*d*)

Deserialize a dictionary representation of a gt.sh.telemetry.from.multipurpose.sensor.100 message object into a GtShTelemetryFromMultipurposeSensor python object for internal use.

This is the near-inverse of the GtShTelemetryFromMultipurposeSensor.as_dict() method:

- Enums: translates between the symbols sent in messages between actors and

the values used by the actors internally once they've deserialized the messages.

- Types: recursively validates and deserializes sub-types.

Note that if a required attribute with a default value is missing in a dict, this method will raise a SchemaError. This differs from the pydantic BaseModel practice of auto-completing missing attributes with default values when they exist.

Parameters
d (*dict*) – the dictionary resulting from json.loads(t) for a serialized JSON type object t.

Raises
SchemaError – if the dict cannot be turned into a GtShTelemetryFromMultipurposeSensor object.

Returns
GtShTelemetryFromMultipurposeSensor

Return type
GtShTelemetryFromMultipurposeSensor

classmethod **tuple_to_type**(*tpl*)

Given a Python class object, returns the serialized JSON type object.

Parameters
tpl (GtShTelemetryFromMultipurposeSensor)

Return type
bytes

classmethod **type_to_tuple**(*t*)

Given a serialized JSON type object, returns the Python class object.

Parameters
t (*bytes*)

Return type
GtShTelemetryFromMultipurposeSensor

1.5.19 GtTelemetry

Python pydantic class corresponding to json type *gt.telemetry*, version *110*.

class gwproto.types.**GtTelemetry**(*, *ScadaReadTimeUnixMs*, *Value*, *Name*, *Exponent*, *TypeName='gt.telemetry'*, *Version='110'*)

Data sent from a SimpleSensor to a SCADA.

This type is meant to be used by a SimpleSensor, where _what_ is doing the reading can be conflated with _what_ is being read.

Parameters

- **ScadaReadTimeUnixMs** (*int*)
- **Value** (*int*)
- **Name** (TelemetryName)
- **Exponent** (*int*)
- **TypeName** (*Literal['gt.telemetry']*)
- **Version** (*Literal['110']*)

as_dict()

Translate the object into a dictionary representation that can be serialized into a gt.telemetry.110 object.

This method prepares the object for serialization by the as_type method, creating a dictionary with key-value pairs that follow the requirements for an instance of the gt.telemetry.110 type. Unlike the standard python dict method, it makes the following substantive changes: - Enum Values: Translates between the values used locally by the actor to the symbol sent in messages. - Removes any key-value pairs where the value is None for a clearer message, especially in cases with many optional attributes.

It also applies these changes recursively to sub-types.

Return type
Dict[str, *Any*]

as_type()

Serialize to the gt.telemetry.110 representation.

Instances in the class are python-native representations of gt.telemetry.110 objects, while the actual gt.telemetry.110 object is the serialized UTF-8 byte string designed for sending in a message.

This method calls the as_dict() method, which differs from the native python dict() in the following key ways: - Enum Values: Translates between the values used locally by the actor to the symbol sent in messages. - - Removes any key-value pairs where the value is None for a clearer message, especially in cases with many optional attributes.

It also applies these changes recursively to sub-types.

Its near-inverse is GtTelemetry.type_to_tuple(). If the type (or any sub-types) includes an enum, then the type_to_tuple will map an unrecognized symbol to the default enum value. This is why these two methods are only 'near' inverses.

Return type
bytes

ScadaReadTimeUnixMs:

- Description: Scada Read Time in Unix Milliseconds.
- Format: ReasonableUnixTimeMs

Value:

- Description: Value. The value of the reading.

Name:

- Description: Name. The name of the Simple Sensing Spaceheat Node. This is both the AboutNodeName and FromNodeName for a data channel. The TelemetryName (and thus Units) are expected to be inferred by the Spaceheat Node. For example this is done initially in SCADA code according to whether the component of the Node is a PipeFlowSensorComponent, SimpleTempSensorComponent etc.

Exponent:

- Description: Exponent. Say the TelemetryName is WaterTempCTimes1000; this corresponds to units of Celsius. To match the implication in the name, the Exponent should be 3, and a Value of 65300 would indicate 65.3 deg C

TypeName:

- Description: All GridWorks Versioned Types have a fixed TypeName, which is a string of lowercase alphanumeric words separated by periods, most significant word (on the left) starting with an alphabet character, and final word NOT all Hindu-Arabic numerals.

Version:

- Description: All GridWorks Versioned Types have a fixed version, which is a string of three Hindu-Arabic numerals.

class gwproto.types.gt_telemetry.**check_is_reasonable_unix_time_ms**(*v*)

Checks ReasonableUnixTimeMs format

ReasonableUnixTimeMs format: unix milliseconds between Jan 1 2000 and Jan 1 3000

Parameters
v (*int*) – the candidate

Raises
ValueError – if v is not ReasonableUnixTimeMs format

class gwproto.types.**GtTelemetry_Maker**(*scada_read_time_unix_ms*, *value*, *name*, *exponent*)

Parameters

- **scada_read_time_unix_ms** (*int*)
- **value** (*int*)
- **name** (TelemetryName)
- **exponent** (*int*)

classmethod **dict_to_tuple**(*d*)

Deserialize a dictionary representation of a gt.telemetry.110 message object into a GtTelemetry python object for internal use.

This is the near-inverse of the GtTelemetry.as_dict() method:

- Enums: translates between the symbols sent in messages between actors and

the values used by the actors internally once they've deserialized the messages.

- Types: recursively validates and deserializes sub-types.

Note that if a required attribute with a default value is missing in a dict, this method will raise a SchemaError. This differs from the pydantic BaseModel practice of auto-completing missing attributes with default values when they exist.

Parameters
d (*dict*) – the dictionary resulting from json.loads(t) for a serialized JSON type object t.

Raises
SchemaError – if the dict cannot be turned into a GtTelemetry object.

Returns
GtTelemetry

Return type
GtTelemetry

classmethod tuple_to_type(*tpl*)

Given a Python class object, returns the serialized JSON type object.

Parameters
tpl (GtTelemetry)

Return type
bytes

classmethod type_to_tuple(*t*)

Given a serialized JSON type object, returns the Python class object.

Parameters
t (*bytes*)

Return type
GtTelemetry

1.5.20 HeartbeatB

Python pydantic class corresponding to json type *heartbeat.b*, version *001*.

class gwproto.types.**HeartbeatB**(*, *FromGNodeAlias*, *FromGNodeInstanceId*, *MyHex='0'*, *YourLastHex*, *LastReceivedTimeUnixMs*, *SendTimeUnixMs*, *StartingOver*, *TypeName='heartbeat.b'*, *Version='001'*)

Heartbeat B.

This is the Heartbeat intended to be sent between the Scada and the AtomicTNode to allow for block-chain validation of the status of their communication.

[More info](https://gridworks.readthedocs.io/en/latest/dispatch-contract.html)

Parameters

- **FromGNodeAlias** (*str*)
- **FromGNodeInstanceId** (*str*)
- **MyHex** (*str*)
- **YourLastHex** (*str*)
- **LastReceivedTimeUnixMs** (*int*)
- **SendTimeUnixMs** (*int*)
- **StartingOver** (*bool*)
- **TypeName** (*Literal['heartbeat.b']*)
- **Version** (*Literal['001']*)

`as_dict()`

Translate the object into a dictionary representation that can be serialized into a heartbeat.b.001 object.

This method prepares the object for serialization by the as_type method, creating a dictionary with key-value pairs that follow the requirements for an instance of the heartbeat.b.001 type. Unlike the standard python dict method, it makes the following substantive changes: - Enum Values: Translates between the values used locally by the actor to the symbol sent in messages. - Removes any key-value pairs where the value is None for a clearer message, especially in cases with many optional attributes.

It also applies these changes recursively to sub-types.

Return type
Dict[str, *Any*]

`as_type()`

Serialize to the heartbeat.b.001 representation.

Instances in the class are python-native representations of heartbeat.b.001 objects, while the actual heartbeat.b.001 object is the serialized UTF-8 byte string designed for sending in a message.

This method calls the as_dict() method, which differs from the native python dict() in the following key ways: - Enum Values: Translates between the values used locally by the actor to the symbol sent in messages. - - Removes any key-value pairs where the value is None for a clearer message, especially in cases with many optional attributes.

It also applies these changes recursively to sub-types.

Its near-inverse is HeartbeatB.type_to_tuple(). If the type (or any sub-types) includes an enum, then the type_to_tuple will map an unrecognized symbol to the default enum value. This is why these two methods are only 'near' inverses.

Return type
bytes

FromGNodeAlias:

- Description: My GNodeAlias.
- Format: LeftRightDot

FromGNodeInstanceId:

- Description: My GNodeInstanceId.
- Format: UuidCanonicalTextual

MyHex:

- Description: Hex character getting sent.
- Format: HexChar

YourLastHex:

- Description: Last hex character received from heartbeat partner..
- Format: HexChar

LastReceivedTimeUnixMs:

- Description: Time YourLastHex was received on my clock.
- Format: ReasonableUnixTimeMs

SendTimeUnixMs:

- Description: Time this message is made and sent on my clock.
- Format: ReasonableUnixTimeMs

StartingOver:

- Description: True if the heartbeat initiator wants to start the volley over. (typically the AtomicTNode in an AtomicTNode / SCADA pair) wants to start the heartbeating volley over. The result is that its partner will not expect the initiator to know its last Hex.

TypeName:

- Description: All GridWorks Versioned Types have a fixed TypeName, which is a string of lowercase alphanumeric words separated by periods, most significant word (on the left) starting with an alphabet character, and final word NOT all Hindu-Arabic numerals.

Version:

- Description: All GridWorks Versioned Types have a fixed version, which is a string of three Hindu-Arabic numerals.

class gwproto.types.heartbeat_b.**check_is_uuid_canonical_textual**(*v*)

Checks UuidCanonicalTextual format

UuidCanonicalTextual format: A string of hex words separated by hyphens of length 8-4-4-4-12.

Parameters
v (*str*) – the candidate

Raises
ValueError – if v is not UuidCanonicalTextual format

class gwproto.types.heartbeat_b.**check_is_hex_char**(*v*)

Checks HexChar format

HexChar format: single-char string in '0123456789abcdefABCDEF'

Parameters
v (*str*) – the candidate

Raises
ValueError – if v is not HexChar format

class gwproto.types.heartbeat_b.**check_is_left_right_dot**(*v*)

Checks LeftRightDot Format

LeftRightDot format: Lowercase alphanumeric words separated by periods, with the most significant word (on the left) starting with an alphabet character.

Parameters
v (*str*) – the candidate

Raises
ValueError – if v is not LeftRightDot format

class gwproto.types.heartbeat_b.**check_is_reasonable_unix_time_ms**(*v*)

Checks ReasonableUnixTimeMs format

ReasonableUnixTimeMs format: unix milliseconds between Jan 1 2000 and Jan 1 3000

Parameters
v (*int*) – the candidate

Raises
ValueError – if v is not ReasonableUnixTimeMs format

class gwproto.types.**HeartbeatB_Maker**(*from_g_node_alias*, *from_g_node_instance_id*, *my_hex*, *your_last_hex*, *last_received_time_unix_ms*, *send_time_unix_ms*, *starting_over*)

Parameters

- **from_g_node_alias** (*str*)
- **from_g_node_instance_id** (*str*)
- **my_hex** (*str*)
- **your_last_hex** (*str*)
- **last_received_time_unix_ms** (*int*)
- **send_time_unix_ms** (*int*)
- **starting_over** (*bool*)

classmethod dict_to_tuple(*d*)

Deserialize a dictionary representation of a heartbeat.b.001 message object into a HeartbeatB python object for internal use.

This is the near-inverse of the HeartbeatB.as_dict() method:

- Enums: translates between the symbols sent in messages between actors and

the values used by the actors internally once they've deserialized the messages.

- Types: recursively validates and deserializes sub-types.

Note that if a required attribute with a default value is missing in a dict, this method will raise a SchemaError. This differs from the pydantic BaseModel practice of auto-completing missing attributes with default values when they exist.

Parameters
d (*dict*) – the dictionary resulting from json.loads(t) for a serialized JSON type object t.

Raises
SchemaError – if the dict cannot be turned into a HeartbeatB object.

Returns
HeartbeatB

Return type
HeartbeatB

classmethod tuple_to_type(*tpl*)

Given a Python class object, returns the serialized JSON type object.

Parameters
tpl (HeartbeatB)

Return type
bytes

classmethod type_to_tuple(*t*)

Given a serialized JSON type object, returns the Python class object.

Parameters
t (*bytes*)

Return type
HeartbeatB

1.5.21 HubitatCacGt

Python pydantic class corresponding to json type *hubitat.cac.gt*, version *000*.

class gwproto.types.**HubitatCacGt**(*, *ComponentAttributeClassId*, *DisplayName=None*, *TypeName='hubitat.cac.gt'*, *Version='000'*)

Parameters

- **ComponentAttributeClassId** (*str*)
- **DisplayName** (*str | None*)
- **TypeName** (*Literal['hubitat.cac.gt']*)
- **Version** (*Literal['000']*)

ComponentAttributeClassId:

- Description: ComponentAttributeClassId. Unique identifier for the device class (aka 'cac' or Component Attribute Class). Authority is maintained by the World Registry.
- Format: UuidCanonicalTextual

DisplayName:

- Description: DisplayName. Sample: Hubitat Elevation C-7

TypeName:

- Description: All GridWorks Versioned Types have a fixed TypeName, which is a string of lowercase alphanumeric words separated by periods, most significant word (on the left) starting with an alphabet character, and final word NOT all Hindu-Arabic numerals.

Version:

- Description: All GridWorks Versioned Types have a fixed version, which is a string of three Hindu-Arabic numerals.

class gwproto.types.**HubitatCacGt_Maker**(*cac*)

Parameters

cac (*HubitatCac*)

1.5.22 HubitatComponentGt

Python pydantic class corresponding to json type *hubitat.component.gt*, version *000*.

class gwproto.types.**HubitatComponentGt**(*, *ComponentId*, *ComponentAttributeClassId*, *DisplayName=None*, *HwUid=None*, *TypeName='hubitat.component.gt'*, *Version='000'*, *Hubitat*)

Parameters

- **ComponentId** (*str*)
- **ComponentAttributeClassId** (*str*)
- **DisplayName** (*str | None*)
- **HwUid** (*str | None*)
- **TypeName** (*Literal['hubitat.component.gt']*)
- **Version** (*Literal['000']*)

- **Hubitat** (*HubitatGt*)

ComponentId:

- Description: Component Id. Primary GridWorks identifier for a specific physical instance of a Hubitat, and also as a more generic Component.
- Format: UuidCanonicalTextual

ComponentAttributeClassId:

- Description: ComponentAttributeClassId. Unique identifier for the device class. Authority for these, as well as the relationship between Components and ComponentAttributeClasses (Cacs) is maintained by the World Registry.
- Format: UuidCanonicalTextual

Hubitat:

- Description: Hubitat Type Helper. Includes the information needed to access the MakerAPI of a Hubitat on the Local area network: Host, MakerApiID, AccessToken and MacAddress for the Hubitat.

DisplayName:

- Description: Sample: Oak Hubitat 81:37:82 (using the last 6 digits of the Hubitat MacId in the display name, as well as the short alias for the associated g node.)

HwUid:

- Description: Hardware Unique Id. Use the final 6 characters of the Hubitat mac address.

TypeName:

- Description: All GridWorks Versioned Types have a fixed TypeName, which is a string of lowercase alphanumeric words separated by periods, most significant word (on the left) starting with an alphabet character, and final word NOT all Hindu-Arabic numerals.

Version:

- Description: All GridWorks Versioned Types have a fixed version, which is a string of three Hindu-Arabic numerals.

class gwproto.types.**HubitatComponentGt_Maker**(*component*)

Parameters
component (*HubitatComponent*)

1.5.23 HubitatPollerCacGt

Python pydantic class corresponding to json type *hubitat.poller.cac.gt*, version *000*.

class gwproto.types.**HubitatPollerCacGt**(*, *ComponentAttributeClassId*, *DisplayName=None*, *TypeName='hubitat.poller.cac.gt'*, *Version='000'*)

Parameters

- **ComponentAttributeClassId** (*str*)
- **DisplayName** (*str | None*)
- **TypeName** (*Literal['hubitat.poller.cac.gt']*)
- **Version** (*Literal['000']*)

ComponentAttributeClassId:

- Description: ComponentAttributeClassId. Unique identifier for the device class (aka 'cac' or Component Attribute Class). Authority is maintained by the World Registry.

DisplayName:

- Description: DisplayName. Sample: Honeywell T6 ZWave Thermostat

TypeName:

- Description: All GridWorks Versioned Types have a fixed TypeName, which is a string of lowercase alphanumeric words separated by periods, most significant word (on the left) starting with an alphabet character, and final word NOT all Hindu-Arabic numerals.

Version:

- Description: All GridWorks Versioned Types have a fixed version, which is a string of three Hindu-Arabic numerals.

`class gwproto.types.HubitatPollerCacGt_Maker`

1.5.24 HubitatPollerComponentGt

Python pydantic class corresponding to json type *hubitat.poller.component.gt*, version *000*.

`class gwproto.types.HubitatPollerComponentGt(*, ComponentId, ComponentAttributeClassId, DisplayName=None, HwUid=None, TypeName='hubitat.poller.component.gt', Version='000', Poller)`

Parameters

- **ComponentId** (*str*)
- **ComponentAttributeClassId** (*str*)
- **DisplayName** (*str | None*)
- **HwUid** (*str | None*)
- **TypeName** (*Literal['hubitat.poller.component.gt']*)
- **Version** (*Literal['000']*)
- **Poller** (*HubitatPollerGt*)

ComponentId:

- Description: ComponentId.

ComponentAttributeClassId:

- Description: ComponentAttributeClassId.
- Format: UuidCanonicalTextual

DisplayName:

- Description: DisplayName. Sample: Downstairs Thermostat

HwUid:

- Description: HwUid. Unique Hardware Identifier

Poller:

- Description: Poller. Includes hubitat_component_id (str), device_id (int), enabled (bool), poll_period_s (int) and attributes.

TypeName:

- Description: All GridWorks Versioned Types have a fixed TypeName, which is a string of lowercase alphanumeric words separated by periods, most significant word (on the left) starting with an alphabet character, and final word NOT all Hindu-Arabic numerals.

Version:

- Description: All GridWorks Versioned Types have a fixed version, which is a string of three Hindu-Arabic numerals.

class gwproto.types.**HubitatPollerComponentGt_Maker**

1.5.25 HubitatTankCacGt

Python pydantic class corresponding to json type *hubitat.tank.cac.gt*, version *000*.

class gwproto.types.**HubitatTankCacGt**(*, *ComponentAttributeClassId*, *DisplayName=None*, *TypeName='hubitat.tank.cac.gt'*, *Version='000'*)

Parameters

- **ComponentAttributeClassId** (*str*)
- **DisplayName** (*str | None*)
- **TypeName** (*Literal['hubitat.tank.cac.gt']*)
- **Version** (*Literal['000']*)

ComponentAttributeClassId:

- Description: ComponentAttributeClassId. Unique identifier for the device class (aka 'cac' or Component Attribute Class). Authority is maintained by the World Registry.
- Format: UuidCanonicalTextual

DisplayName:

- Description: Sample: GridWorks TankModule1

TypeName:

- Description: All GridWorks Versioned Types have a fixed TypeName, which is a string of lowercase alphanumeric words separated by periods, most significant word (on the left) starting with an alphabet character, and final word NOT all Hindu-Arabic numerals.

Version:

- Description: All GridWorks Versioned Types have a fixed version, which is a string of three Hindu-Arabic numerals.

class gwproto.types.**HubitatTankCacGt_Maker**(*cac*)

Parameters

cac (*HubitatTankModuleCac*)

1.5.26 HubitatTankComponentGt

Python pydantic class corresponding to json type *hubitat.tank.component.gt*, version *000*.

class gwproto.types.**HubitatTankComponentGt**(*, *ComponentId*, *ComponentAttributeClassId*, *DisplayName=None*, *HwUid=None*, *TypeName='hubitat.tank.component.gt'*, *Version='000'*, *Tank*)

Parameters

- **ComponentId** (*str*)
- **ComponentAttributeClassId** (*str*)
- **DisplayName** (*str | None*)
- **HwUid** (*str | None*)
- **TypeName** (*Literal['hubitat.tank.component.gt']*)
- **Version** (*Literal['000']*)
- **Tank** (*HubitatTankSettingsGt*)

ComponentId:

- Description: Component Id. Primary GridWorks identifier for a specific physical instance of a GridWorks TankModule1 and also as a more generic Component.
- Format: UuidCanonicalTextual

ComponentAttributeClassId:

- Description: ComponentAttributeClassId. Unique identifier for the device class. Authority for these, as well as the relationship between Components and ComponentAttributeClasses (Cacs) is maintained by the World Registry.
- Format: UuidCanonicalTextual

Tank:

- Description: Tank. The configuration information (HubitatTankSettingsGt) about the 4 analog temperature sensors for a GridWorks TankModule1.

DisplayName:

- Description: Sample: GridWorks TankModule <buffer> SN 1010

HwUid:

- Description: Hardware Unique Id. Use the GridWorks Serial number for GridWorks TankModule1.

TypeName:

- Description: All GridWorks Versioned Types have a fixed TypeName, which is a string of lowercase alphanumeric words separated by periods, most significant word (on the left) starting with an alphabet character, and final word NOT all Hindu-Arabic numerals.

Version:

- Description: All GridWorks Versioned Types have a fixed version, which is a string of three Hindu-Arabic numerals.

class gwproto.types.**HubitatTankComponentGt_Maker**(*component*)

Parameters
component (*HubitatTankComponent*)

1.5.27 MultipurposeSensorCacGt

Python pydantic class corresponding to json type *multipurpose.sensor.cac.gt*, version *000*.

class gwproto.types.**MultipurposeSensorCacGt**(*, *ComponentAttributeClassId*, *MakeModel*, *PollPeriodMs*, *Exponent*, *TempUnit*, *TelemetryNameList*, *MaxThermistors=None*, *DisplayName=None*, *CommsMethod=None*, *TypeName='multipurpose.sensor.cac.gt'*, *Version='000'*)

Type for tracking Multipuprose Sensor ComponentAttributeClasses.

GridWorks Spaceheat SCADA uses the GridWorks GNodeRegistry structures and abstractions for managing relational device data. The Cac, or ComponentAttributeClass, is part of this structure.

[More info](https://g-node-registry.readthedocs.io/en/latest/component-attribute-class.html)

Parameters

- **ComponentAttributeClassId** (*str*)
- **MakeModel** (MakeModel)
- **PollPeriodMs** (*int*)
- **Exponent** (*int*)
- **TempUnit** (Unit)
- **TelemetryNameList** (*List[TelemetryName]*)
- **MaxThermistors** (*int | None*)
- **DisplayName** (*str | None*)
- **CommsMethod** (*str | None*)
- **TypeName** (*Literal['multipurpose.sensor.cac.gt']*)
- **Version** (*Literal['000']*)

as_dict()

Translate the object into a dictionary representation that can be serialized into a multipurpose.sensor.cac.gt.000 object.

This method prepares the object for serialization by the as_type method, creating a dictionary with key-value pairs that follow the requirements for an instance of the multipurpose.sensor.cac.gt.000 type. Unlike the standard python dict method, it makes the following substantive changes: - Enum Values: Translates between the values used locally by the actor to the symbol sent in messages. - Removes any key-value pairs where the value is None for a clearer message, especially in cases with many optional attributes.

It also applies these changes recursively to sub-types.

Return type
Dict[str, *Any*]

`as_type()`

Serialize to the multipurpose.sensor.cac.gt.000 representation.

Instances in the class are python-native representations of multipurpose.sensor.cac.gt.000 objects, while the actual multipurpose.sensor.cac.gt.000 object is the serialized UTF-8 byte string designed for sending in a message.

This method calls the as_dict() method, which differs from the native python dict() in the following key ways: - Enum Values: Translates between the values used locally by the actor to the symbol sent in messages. - - Removes any key-value pairs where the value is None for a clearer message, especially in cases with many optional attributes.

It also applies these changes recursively to sub-types.

Its near-inverse is MultipurposeSensorCacGt.type_to_tuple(). If the type (or any sub-types) includes an enum, then the type_to_tuple will map an unrecognized symbol to the default enum value. This is why these two methods are only 'near' inverses.

Return type
bytes

ComponentAttributeClassId:

- Description: ComponentAttributeClassId. Unique identifier for the device class (aka 'cac' or Component Attribute Class). Authority is maintained by the World Registry.
- Format: UuidCanonicalTextual

MakeModel:

- Description: MakeModel. Meant to be enough to articulate any difference in how GridWorks code would interact with a device. Should be able to use this information to buy or build a device.

PollPeriodMs:

- Description: Poll Period in Milliseconds. Poll Period refers to the period of time between two readings by the local actor. This is in contrast to Capture Period, which refers to the period between readings that are sent up to the cloud (or otherwise saved for the long-term).

Exponent:

- Description: Exponent. Say the TelemetryName is WaterTempCTimes1000; this corresponds to units of Celsius. To match the implication in the name, the Exponent should be 3, and a Value of 65300 would indicate 65.3 deg C

TempUnit:

- Description: Temp Unit.

TelemetryNameList:

- Description:

MaxThermistors:

- Description: The maximum number of temperature sensors this multipurpose sensor can read.

DisplayName:

- Description: Sample: GridWorks TSnap1.0 as 12-channel analog temp sensor

CommsMethod:

- Description:

TypeName:

- Description: All GridWorks Versioned Types have a fixed TypeName, which is a string of lowercase alphanumeric words separated by periods, most significant word (on the left) starting with an alphabet character, and final word NOT all Hindu-Arabic numerals.

Version:

- Description: All GridWorks Versioned Types have a fixed version, which is a string of three Hindu-Arabic numerals.

class gwproto.types.multipurpose_sensor_cac_gt.**check_is_uuid_canonical_textual**(*v*)

Checks UuidCanonicalTextual format

UuidCanonicalTextual format: A string of hex words separated by hyphens of length 8-4-4-4-12.

Parameters
v (*str*) – the candidate

Raises
ValueError – if v is not UuidCanonicalTextual format

class gwproto.types.**MultipurposeSensorCacGt_Maker**(*component_attribute_class_id*, *make_model*, *poll_period_ms*, *exponent*, *temp_unit*, *telemetry_name_list*, *max_thermistors*, *display_name*, *comms_method*)

Parameters

- **component_attribute_class_id** (*str*)
- **make_model** (MakeModel)
- **poll_period_ms** (*int*)
- **exponent** (*int*)
- **temp_unit** (Unit)
- **telemetry_name_list** (*List[TelemetryName]*)
- **max_thermistors** (*int | None*)
- **display_name** (*str | None*)
- **comms_method** (*str | None*)

classmethod dict_to_tuple(*d*)

Deserialize a dictionary representation of a multipurpose.sensor.cac.gt.000 message object into a MultipurposeSensorCacGt python object for internal use.

This is the near-inverse of the MultipurposeSensorCacGt.as_dict() method:

- Enums: translates between the symbols sent in messages between actors and

the values used by the actors internally once they've deserialized the messages.

- Types: recursively validates and deserializes sub-types.

Note that if a required attribute with a default value is missing in a dict, this method will raise a SchemaError. This differs from the pydantic BaseModel practice of auto-completing missing attributes with default values when they exist.

Parameters
d (*dict*) – the dictionary resulting from json.loads(t) for a serialized JSON type object t.

Raises
SchemaError – if the dict cannot be turned into a MultipurposeSensorCacGt object.

Returns
MultipurposeSensorCacGt

Return type
MultipurposeSensorCacGt

`classmethod tuple_to_type(tpl)`

Given a Python class object, returns the serialized JSON type object.

Parameters
tpl (MultipurposeSensorCacGt)

Return type
bytes

`classmethod type_to_tuple(t)`

Given a serialized JSON type object, returns the Python class object.

Parameters
t (*bytes*)

Return type
MultipurposeSensorCacGt

1.5.28 MultipurposeSensorComponentGt

Python pydantic class corresponding to json type *multipurpose.sensor.component.gt*, version *000*.

ComponentId:

- Description: Component Id. Primary GridWorks identifier for a specific physical instance of a MultipurposeSensor (perhaps only the 12-channel analog temp sensor), and also as a more generic Component.
- Format: UuidCanonicalTextual

ComponentAttributeClassId:

- Description: ComponentAttributeClassId. Unique identifier for the device class. Authority for these, as well as the relationship between Components and ComponentAttributeClasses (Cacs) is maintained by the World Registry.
- Format: UuidCanonicalTextual

ChannelList:

- Description:

ConfigList:

- Description:

HwUid:

- Description: Hardware Unique Id.

DisplayName:

- Description: Sample: Oak Multipurpose Temp Sensor Component <100>

TypeName:

- Description: All GridWorks Versioned Types have a fixed TypeName, which is a string of lowercase alphanumeric words separated by periods, most significant word (on the left) starting with an alphabet character, and final word NOT all Hindu-Arabic numerals.

Version:

- Description: All GridWorks Versioned Types have a fixed version, which is a string of three Hindu-Arabic numerals.

class gwproto.types.multipurpose_sensor_component_gt.**check_is_uuid_canonical_textual**(*v*)

Checks UuidCanonicalTextual format

UuidCanonicalTextual format: A string of hex words separated by hyphens of length 8-4-4-4-12.

Parameters
v (*str*) – the candidate

Raises
ValueError – if v is not UuidCanonicalTextual format

class gwproto.types.multipurpose_sensor_component_gt.**check_is_left_right_dot**(*v*)

Checks LeftRightDot Format

LeftRightDot format: Lowercase alphanumeric words separated by periods, with the most significant word (on the left) starting with an alphabet character.

Parameters
v (*str*) – the candidate

Raises
ValueError – if v is not LeftRightDot format

1.5.29 PipeFlowSensorCacGt

Python pydantic class corresponding to json type *pipe.flow.sensor.cac.gt*, version *000*.

class gwproto.types.**PipeFlowSensorCacGt**(*, *ComponentAttributeClassId*, *MakeModel*, *DisplayName=None*, *CommsMethod=None*, *TypeName='pipe.flow.sensor.cac.gt'*, *Version='000'*)

Type for tracking Pipe Flow Sensor ComponentAttributeClasses.

GridWorks Spaceheat SCADA uses the GridWorks GNodeRegistry structures and abstractions for managing relational device data. The Cac, or ComponentAttributeClass, is part of this structure.

[More info](https://g-node-registry.readthedocs.io/en/latest/component-attribute-class.html)

Parameters

- **ComponentAttributeClassId** (*str*)
- **MakeModel** (MakeModel)
- **DisplayName** (*str | None*)
- **CommsMethod** (*str | None*)
- **TypeName** (*Literal['pipe.flow.sensor.cac.gt']*)
- **Version** (*Literal['000']*)

as_dict()

Translate the object into a dictionary representation that can be serialized into a pipe.flow.sensor.cac.gt.000 object.

This method prepares the object for serialization by the as_type method, creating a dictionary with key-value pairs that follow the requirements for an instance of the pipe.flow.sensor.cac.gt.000 type. Unlike the standard python dict method, it makes the following substantive changes: - Enum Values: Translates

between the values used locally by the actor to the symbol sent in messages. - Removes any key-value pairs where the value is None for a clearer message, especially in cases with many optional attributes.

It also applies these changes recursively to sub-types.

Return type
Dict[str, *Any*]

`as_type()`

Serialize to the pipe.flow.sensor.cac.gt.000 representation.

Instances in the class are python-native representations of pipe.flow.sensor.cac.gt.000 objects, while the actual pipe.flow.sensor.cac.gt.000 object is the serialized UTF-8 byte string designed for sending in a message.

This method calls the as_dict() method, which differs from the native python dict() in the following key ways: - Enum Values: Translates between the values used locally by the actor to the symbol sent in messages. - - Removes any key-value pairs where the value is None for a clearer message, especially in cases with many optional attributes.

It also applies these changes recursively to sub-types.

Its near-inverse is PipeFlowSensorCacGt.type_to_tuple(). If the type (or any sub-types) includes an enum, then the type_to_tuple will map an unrecognized symbol to the default enum value. This is why these two methods are only 'near' inverses.

Return type
bytes

ComponentAttributeClassId:

- Description: ComponentAttributeClassId. Unique identifier for the device class (aka 'cac' or Component Attribute Class). Authority is maintained by the World Registry.
- Format: UuidCanonicalTextual

MakeModel:

- Description:

DisplayName:

- Description: Sample: Atlas Scientific EZO FLO i2c

CommsMethod:

- Description:

TypeName:

- Description: All GridWorks Versioned Types have a fixed TypeName, which is a string of lowercase alphanumeric words separated by periods, most significant word (on the left) starting with an alphabet character, and final word NOT all Hindu-Arabic numerals.

Version:

- Description: All GridWorks Versioned Types have a fixed version, which is a string of three Hindu-Arabic numerals.

`class gwproto.types.pipe_flow_sensor_cac_gt.check_is_uuid_canonical_textual(v)`

Checks UuidCanonicalTextual format

UuidCanonicalTextual format: A string of hex words separated by hyphens of length 8-4-4-4-12.

Parameters
v (*str*) – the candidate

Raises
ValueError – if v is not UuidCanonicalTextual format

class gwproto.types.**PipeFlowSensorCacGt_Maker**(*component_attribute_class_id*, *make_model*, *display_name*, *comms_method*)

Parameters

- **component_attribute_class_id** (*str*)
- **make_model** (MakeModel)
- **display_name** (*str | None*)
- **comms_method** (*str | None*)

classmethod **dict_to_tuple**(*d*)

Deserialize a dictionary representation of a pipe.flow.sensor.cac.gt.000 message object into a PipeFlowSensorCacGt python object for internal use.

This is the near-inverse of the PipeFlowSensorCacGt.as_dict() method:

- Enums: translates between the symbols sent in messages between actors and

the values used by the actors internally once they've deserialized the messages.

- Types: recursively validates and deserializes sub-types.

Note that if a required attribute with a default value is missing in a dict, this method will raise a SchemaError. This differs from the pydantic BaseModel practice of auto-completing missing attributes with default values when they exist.

Parameters
d (*dict*) – the dictionary resulting from json.loads(t) for a serialized JSON type object t.

Raises
SchemaError – if the dict cannot be turned into a PipeFlowSensorCacGt object.

Returns
PipeFlowSensorCacGt

Return type
PipeFlowSensorCacGt

classmethod **tuple_to_type**(*tpl*)

Given a Python class object, returns the serialized JSON type object.

Parameters
tpl (PipeFlowSensorCacGt)

Return type
bytes

classmethod **type_to_tuple**(*t*)

Given a serialized JSON type object, returns the Python class object.

Parameters
t (*bytes*)

Return type
PipeFlowSensorCacGt

1.5.30 PipeFlowSensorComponentGt

Python pydantic class corresponding to json type *pipe.flow.sensor.component.gt*, version *000*.

class gwproto.types.**PipeFlowSensorComponentGt**(*, *ComponentId*, *ComponentAttributeClassId*, *I2cAddress*, *ConversionFactor*, *DisplayName=None*, *HwUid=None*, *ExpectedMaxGpmTimes100=None*, *TypeName='pipe.flow.sensor.component.gt'*, *Version='000'*)

Type for tracking Pipe Flow Sensor Components.

Designed for Pipe Flow Sensors. It extends the component.gt.000 type. Authority for the attributes of the component.gt.000 (ComponentId, ComponentAttributeClassId, DisplayName, HwUid) belongs to the WorldRegistry. The WorldRegistry is part of the GridWorks 'BackOffice' structure for managing relational device data . Notably, ComponentId and ComponentAttributeClass are both required and immutable. HwUid is optional but once it is set to a non-null value that is also immutable - it is meant to be an immutable identifier associated to a specific physical device, ideally one that can be read remotely by the SCADA and also by the naked eye. The DisplayName is mutable, with its current value in time governed by the WorldRegistry.

[More info](https://gridworks-protocol.readthedocs.io/en/latest/component.html)

Parameters

- **ComponentId** (*str*)
- **ComponentAttributeClassId** (*str*)
- **I2cAddress** (*int*)
- **ConversionFactor** (*float*)
- **DisplayName** (*str | None*)
- **HwUid** (*str | None*)
- **ExpectedMaxGpmTimes100** (*int | None*)
- **TypeName** (*Literal['pipe.flow.sensor.component.gt']*)
- **Version** (*Literal['000']*)

as_dict()

Translate the object into a dictionary representation that can be serialized into a pipe.flow.sensor.component.gt.000 object.

This method prepares the object for serialization by the as_type method, creating a dictionary with key-value pairs that follow the requirements for an instance of the pipe.flow.sensor.component.gt.000 type. Unlike the standard python dict method, it makes the following substantive changes: - Enum Values: Translates between the values used locally by the actor to the symbol sent in messages. - Removes any key-value pairs where the value is None for a clearer message, especially in cases with many optional attributes.

It also applies these changes recursively to sub-types.

Return type
Dict[str, *Any*]

as_type()

Serialize to the pipe.flow.sensor.component.gt.000 representation.

Instances in the class are python-native representations of pipe.flow.sensor.component.gt.000 objects, while the actual pipe.flow.sensor.component.gt.000 object is the serialized UTF-8 byte string designed for sending in a message.

This method calls the as_dict() method, which differs from the native python dict() in the following key ways: - Enum Values: Translates between the values used locally by the actor to the symbol sent in messages. - - Removes any key-value pairs where the value is None for a clearer message, especially in cases with many optional attributes.

It also applies these changes recursively to sub-types.

Its near-inverse is PipeFlowSensorComponentGt.type_to_tuple(). If the type (or any sub-types) includes an enum, then the type_to_tuple will map an unrecognized symbol to the default enum value. This is why these two methods are only 'near' inverses.

Return type
bytes

ComponentId:

- Description: Component Id. Primary GridWorks identifier for a specific physical instance of a PipeFlowSensor, and also as a more generic Component.
- Format: UuidCanonicalTextual

ComponentAttributeClassId:

- Description: ComponentAttributeClassId. Unique identifier for the device class. Authority for these, as well as the relationship between Components and ComponentAttributeClasses (Cacs) is maintained by the World Registry.
- Format: UuidCanonicalTextual

I2cAddress:

- Description:

ConversionFactor:

- Description:

DisplayName:

- Description: Sample: Pipe Flow Meter Component <dist-flow>

HwUid:

- Description: Hardware Unique Id.

ExpectedMaxGpmTimes100:

- Description: Expected Max Flow in Gallons per Minute, times 100.

TypeName:

- Description: All GridWorks Versioned Types have a fixed TypeName, which is a string of lowercase alphanumeric words separated by periods, most significant word (on the left) starting with an alphabet character, and final word NOT all Hindu-Arabic numerals.

Version:

- Description: All GridWorks Versioned Types have a fixed version, which is a string of three Hindu-Arabic numerals.

`class gwproto.types.pipe_flow_sensor_component_gt.check_is_uuid_canonical_textual(v)`

Checks UuidCanonicalTextual format

UuidCanonicalTextual format: A string of hex words separated by hyphens of length 8-4-4-4-12.

Parameters
v (*str*) – the candidate

Raises
ValueError – if v is not UuidCanonicalTextual format

class gwproto.types.**PipeFlowSensorComponentGt_Maker**(*component_id*, *component_attribute_class_id*, *i2c_address*, *conversion_factor*, *display_name*, *hw_uid*, *expected_max_gpm_times100*)

Parameters

- **component_id** (*str*)
- **component_attribute_class_id** (*str*)
- **i2c_address** (*int*)
- **conversion_factor** (*float*)
- **display_name** (*str | None*)
- **hw_uid** (*str | None*)
- **expected_max_gpm_times100** (*int | None*)

classmethod **dict_to_tuple**(*d*)

Deserialize a dictionary representation of a pipe.flow.sensor.component.gt.000 message object into a PipeFlowSensorComponentGt python object for internal use.

This is the near-inverse of the PipeFlowSensorComponentGt.as_dict() method:

- Enums: translates between the symbols sent in messages between actors and

the values used by the actors internally once they've deserialized the messages.

- Types: recursively validates and deserializes sub-types.

Note that if a required attribute with a default value is missing in a dict, this method will raise a SchemaError. This differs from the pydantic BaseModel practice of auto-completing missing attributes with default values when they exist.

Parameters
d (*dict*) – the dictionary resulting from json.loads(t) for a serialized JSON type object t.

Raises
SchemaError – if the dict cannot be turned into a PipeFlowSensorComponentGt object.

Returns
PipeFlowSensorComponentGt

Return type
PipeFlowSensorComponentGt

classmethod **tuple_to_type**(*tpl*)

Given a Python class object, returns the serialized JSON type object.

Parameters
tpl (PipeFlowSensorComponentGt)

Return type
bytes

classmethod **type_to_tuple**(*t*)

Given a serialized JSON type object, returns the Python class object.

Parameters
t (*bytes*)

Return type
PipeFlowSensorComponentGt

1.5.31 PowerWatts

Python pydantic class corresponding to json type *power.watts*, version *000*.

class gwproto.types.**PowerWatts**(*, *Watts*, *TypeName='power.watts'*, *Version='000'*)

Real-time power of TerminalAsset in Watts.

Used by a SCADA -> Atn or Atn -> AggregatedTNode to report real-time power of their TerminalAsset. Positive number means WITHDRAWAL from the grid - so generating electricity creates a negative number. This message is considered worse than useless to send after the first attempt, and does not require an ack. Shares the same purpose as gs.pwr, but is not designed to minimize bytes so comes in JSON format.

Parameters

- **Watts** (*int*)
- **TypeName** (*Literal['power.watts']*)
- **Version** (*Literal['000']*)

as_dict()

Translate the object into a dictionary representation that can be serialized into a power.watts.000 object.

This method prepares the object for serialization by the as_type method, creating a dictionary with key-value pairs that follow the requirements for an instance of the power.watts.000 type. Unlike the standard python dict method, it makes the following substantive changes: - Enum Values: Translates between the values used locally by the actor to the symbol sent in messages. - Removes any key-value pairs where the value is None for a clearer message, especially in cases with many optional attributes.

It also applies these changes recursively to sub-types.

Return type
Dict[str, *Any*]

as_type()

Serialize to the power.watts.000 representation.

Instances in the class are python-native representations of power.watts.000 objects, while the actual power.watts.000 object is the serialized UTF-8 byte string designed for sending in a message.

This method calls the as_dict() method, which differs from the native python dict() in the following key ways: - Enum Values: Translates between the values used locally by the actor to the symbol sent in messages. - - Removes any key-value pairs where the value is None for a clearer message, especially in cases with many optional attributes.

It also applies these changes recursively to sub-types.

Its near-inverse is PowerWatts.type_to_tuple(). If the type (or any sub-types) includes an enum, then the type_to_tuple will map an unrecognized symbol to the default enum value. This is why these two methods are only 'near' inverses.

Return type
bytes

Watts:

- Description: Current Power in Watts.

TypeName:

- Description: All GridWorks Versioned Types have a fixed TypeName, which is a string of lowercase alphanumeric words separated by periods, most significant word (on the left) starting with an alphabet character, and final word NOT all Hindu-Arabic numerals.

Version:

- Description: All GridWorks Versioned Types have a fixed version, which is a string of three Hindu-Arabic numerals.

class gwproto.types.**PowerWatts_Maker**(*watts*)

Parameters
watts (*int*)

classmethod **dict_to_tuple**(*d*)

Deserialize a dictionary representation of a power.watts.000 message object into a PowerWatts python object for internal use.

This is the near-inverse of the PowerWatts.as_dict() method:

- Enums: translates between the symbols sent in messages between actors and

the values used by the actors internally once they've deserialized the messages.

- Types: recursively validates and deserializes sub-types.

Note that if a required attribute with a default value is missing in a dict, this method will raise a SchemaError. This differs from the pydantic BaseModel practice of auto-completing missing attributes with default values when they exist.

Parameters
d (*dict*) – the dictionary resulting from json.loads(t) for a serialized JSON type object t.

Raises
SchemaError – if the dict cannot be turned into a PowerWatts object.

Returns
PowerWatts

Return type
PowerWatts

classmethod **tuple_to_type**(*tpl*)

Given a Python class object, returns the serialized JSON type object.

Parameters
tpl (PowerWatts)

Return type
bytes

classmethod **type_to_tuple**(*t*)

Given a serialized JSON type object, returns the Python class object.

Parameters
t (*bytes*)

Return type
PowerWatts

1.5.32 RelayCacGt

Python pydantic class corresponding to json type *relay.cac.gt*, version *000*.

class gwproto.types.**RelayCacGt**(*, *ComponentAttributeClassId*, *MakeModel*, *DisplayName=None*, *TypicalResponseTimeMs*, *TypeName='relay.cac.gt'*, *Version='000'*)

Type for tracking Relay ComponentAttributeClasses.

GridWorks Spaceheat SCADA uses the GridWorks GNodeRegistry structures and abstractions for managing relational device data. The Cac, or ComponentAttributeClass, is part of this structure.

[More info](https://g-node-registry.readthedocs.io/en/latest/component-attribute-class.html)

Parameters

- **ComponentAttributeClassId** (*str*)
- **MakeModel** (MakeModel)
- **DisplayName** (*str | None*)
- **TypicalResponseTimeMs** (*int*)
- **TypeName** (*Literal['relay.cac.gt']*)
- **Version** (*Literal['000']*)

as_dict()

Translate the object into a dictionary representation that can be serialized into a relay.cac.gt.000 object.

This method prepares the object for serialization by the as_type method, creating a dictionary with key-value pairs that follow the requirements for an instance of the relay.cac.gt.000 type. Unlike the standard python dict method, it makes the following substantive changes: - Enum Values: Translates between the values used locally by the actor to the symbol sent in messages. - Removes any key-value pairs where the value is None for a clearer message, especially in cases with many optional attributes.

It also applies these changes recursively to sub-types.

Return type
Dict[str, *Any*]

as_type()

Serialize to the relay.cac.gt.000 representation.

Instances in the class are python-native representations of relay.cac.gt.000 objects, while the actual relay.cac.gt.000 object is the serialized UTF-8 byte string designed for sending in a message.

This method calls the as_dict() method, which differs from the native python dict() in the following key ways: - Enum Values: Translates between the values used locally by the actor to the symbol sent in messages. - - Removes any key-value pairs where the value is None for a clearer message, especially in cases with many optional attributes.

It also applies these changes recursively to sub-types.

Its near-inverse is RelayCacGt.type_to_tuple(). If the type (or any sub-types) includes an enum, then the type_to_tuple will map an unrecognized symbol to the default enum value. This is why these two methods are only 'near' inverses.

Return type
bytes

ComponentAttributeClassId:

- Description: ComponentAttributeClassId. Unique identifier for the device class (aka 'cac' or Component Attribute Class). Authority is maintained by the World Registry.
- Format: UuidCanonicalTextual

MakeModel:

- Description:

DisplayName:

- Description:

TypicalResponseTimeMs:

- Description:

TypeName:

- Description: All GridWorks Versioned Types have a fixed TypeName, which is a string of lowercase alphanumeric words separated by periods, most significant word (on the left) starting with an alphabet character, and final word NOT all Hindu-Arabic numerals.

Version:

- Description: All GridWorks Versioned Types have a fixed version, which is a string of three Hindu-Arabic numerals.

class gwproto.types.relay_cac_gt.**check_is_uuid_canonical_textual**(*v*)

Checks UuidCanonicalTextual format

UuidCanonicalTextual format: A string of hex words separated by hyphens of length 8-4-4-4-12.

Parameters
v (*str*) – the candidate

Raises
ValueError – if v is not UuidCanonicalTextual format

class gwproto.types.**RelayCacGt_Maker**(*component_attribute_class_id*, *make_model*, *display_name*, *typical_response_time_ms*)

Parameters

- **component_attribute_class_id** (*str*)
- **make_model** (MakeModel)
- **display_name** (*str | None*)
- **typical_response_time_ms** (*int*)

classmethod **dict_to_tuple**(*d*)

Deserialize a dictionary representation of a relay.cac.gt.000 message object into a RelayCacGt python object for internal use.

This is the near-inverse of the RelayCacGt.as_dict() method:

- Enums: translates between the symbols sent in messages between actors and

the values used by the actors internally once they've deserialized the messages.

- Types: recursively validates and deserializes sub-types.

Note that if a required attribute with a default value is missing in a dict, this method will raise a SchemaError. This differs from the pydantic BaseModel practice of auto-completing missing attributes with default values when they exist.

Parameters
d (*dict*) – the dictionary resulting from json.loads(t) for a serialized JSON type object t.

Raises
SchemaError – if the dict cannot be turned into a RelayCacGt object.

Returns
RelayCacGt

Return type
RelayCacGt

classmethod tuple_to_type(*tpl*)

Given a Python class object, returns the serialized JSON type object.

Parameters
tpl (RelayCacGt)

Return type
bytes

classmethod type_to_tuple(*t*)

Given a serialized JSON type object, returns the Python class object.

Parameters
t (*bytes*)

Return type
RelayCacGt

1.5.33 RelayComponentGt

Python pydantic class corresponding to json type *relay.component.gt*, version *000*.

class gwproto.types.**RelayComponentGt**(*, *ComponentId*, *ComponentAttributeClassId*, *DisplayName=None*, *Gpio=None*, *HwUid=None*, *NormallyOpen*, *TypeName='relay.component.gt'*, *Version='000'*)

Type for tracking Relay Components.

Designed for Relays. It extends the component.gt.000 type. Authority for the attributes of the component.gt.000 (ComponentId, ComponentAttributeClassId, DisplayName, HwUid) belongs to the WorldRegistry. The WorldRegistry is part of the GridWorks 'BackOffice' structure for managing relational device data . Notably, ComponentId and ComponentAttributeClass are both required and immutable. HwUid is optional but once it is set to a non-null value that is also immutable - it is meant to be an immutable identifier associated to a specific physical device, ideally one that can be read remotely by the SCADA and also by the naked eye. The DisplayName is mutable, with its current value in time governed by the WorldRegistry.

[More info](https://g-node-registry.readthedocs.io/en/latest/component.html)

Parameters

- **ComponentId** (*str*)
- **ComponentAttributeClassId** (*str*)
- **DisplayName** (*str | None*)
- **Gpio** (*int | None*)
- **HwUid** (*str | None*)
- **NormallyOpen** (*bool*)

- **TypeName** (*Literal['relay.component.gt']*)
- **Version** (*Literal['000']*)

as_dict()

Translate the object into a dictionary representation that can be serialized into a relay.component.gt.000 object.

This method prepares the object for serialization by the as_type method, creating a dictionary with key-value pairs that follow the requirements for an instance of the relay.component.gt.000 type. Unlike the standard python dict method, it makes the following substantive changes: - Enum Values: Translates between the values used locally by the actor to the symbol sent in messages. - Removes any key-value pairs where the value is None for a clearer message, especially in cases with many optional attributes.

It also applies these changes recursively to sub-types.

Return type
Dict[str, *Any*]

as_type()

Serialize to the relay.component.gt.000 representation.

Instances in the class are python-native representations of relay.component.gt.000 objects, while the actual relay.component.gt.000 object is the serialized UTF-8 byte string designed for sending in a message.

This method calls the as_dict() method, which differs from the native python dict() in the following key ways: - Enum Values: Translates between the values used locally by the actor to the symbol sent in messages. - - Removes any key-value pairs where the value is None for a clearer message, especially in cases with many optional attributes.

It also applies these changes recursively to sub-types.

Its near-inverse is RelayComponentGt.type_to_tuple(). If the type (or any sub-types) includes an enum, then the type_to_tuple will map an unrecognized symbol to the default enum value. This is why these two methods are only 'near' inverses.

Return type
bytes

ComponentId:

- Description: Component Id. Primary GridWorks identifier for a specific physical instance of a Relay, and also as a more generic Component.
- Format: UuidCanonicalTextual

ComponentAttributeClassId:

- Description: ComponentAttributeClassId. Unique identifier for the device class. Authority for these, as well as the relationship between Components and ComponentAttributeClasses (Cacs) is maintained by the World Registry.
- Format: UuidCanonicalTextual

DisplayName:

- Description:

Gpio:

- Description:

HwUid:

- Description: Hardware Unique Id.

NormallyOpen:

- Description: Normally Open. Normally open relays default in the open position, meaning that when they're not in use, there is no contact between the circuits. When power is introduced, an electromagnet pulls the first circuit into contact with the second, thereby closing the circuit and allowing power to flow through

TypeName:

- Description: All GridWorks Versioned Types have a fixed TypeName, which is a string of lowercase alphanumeric words separated by periods, most significant word (on the left) starting with an alphabet character, and final word NOT all Hindu-Arabic numerals.

Version:

- Description: All GridWorks Versioned Types have a fixed version, which is a string of three Hindu-Arabic numerals.

class gwproto.types.relay_component_gt.**check_is_uuid_canonical_textual**(*v*)

Checks UuidCanonicalTextual format

UuidCanonicalTextual format: A string of hex words separated by hyphens of length 8-4-4-4-12.

Parameters
v (*str*) – the candidate

Raises
ValueError – if v is not UuidCanonicalTextual format

class gwproto.types.**RelayComponentGt_Maker**(*component_id*, *component_attribute_class_id*, *display_name*, *gpio*, *hw_uid*, *normally_open*)

Parameters

- **component_id** (*str*)
- **component_attribute_class_id** (*str*)
- **display_name** (*str | None*)
- **gpio** (*int | None*)
- **hw_uid** (*str | None*)
- **normally_open** (*bool*)

classmethod **dict_to_tuple**(*d*)

Deserialize a dictionary representation of a relay.component.gt.000 message object into a RelayComponentGt python object for internal use.

This is the near-inverse of the RelayComponentGt.as_dict() method:

- Enums: translates between the symbols sent in messages between actors and

the values used by the actors internally once they've deserialized the messages.

- Types: recursively validates and deserializes sub-types.

Note that if a required attribute with a default value is missing in a dict, this method will raise a SchemaError. This differs from the pydantic BaseModel practice of auto-completing missing attributes with default values when they exist.

Parameters
d (*dict*) – the dictionary resulting from json.loads(t) for a serialized JSON type object t.

Raises
SchemaError – if the dict cannot be turned into a RelayComponentGt object.

Returns
RelayComponentGt

Return type
RelayComponentGt

classmethod tuple_to_type(*tpl*)

Given a Python class object, returns the serialized JSON type object.

Parameters
tpl (RelayComponentGt)

Return type
bytes

classmethod type_to_tuple(*t*)

Given a serialized JSON type object, returns the Python class object.

Parameters
t (*bytes*)

Return type
RelayComponentGt

1.5.34 ResistiveHeaterCacGt

Python pydantic class corresponding to json type *resistive.heater.cac.gt*, version *000*.

class gwproto.types.**ResistiveHeaterCacGt**(*, *ComponentAttributeClassId*, *MakeModel*, *DisplayName=None*, *NameplateMaxPowerW*, *RatedVoltageV*, *TypeName='resistive.heater.cac.gt'*, *Version='000'*)

Type for tracking Resistive Heater ComponentAttributeClasses.

GridWorks Spaceheat SCADA uses the GridWorks GNodeRegistry structures and abstractions for managing relational device data. The Cac, or ComponentAttributeClass, is part of this structure.

[More info](https://g-node-registry.readthedocs.io/en/latest/component-attribute-class.html)

Parameters

- **ComponentAttributeClassId** (*str*)
- **MakeModel** (MakeModel)
- **DisplayName** (*str | None*)
- **NameplateMaxPowerW** (*int*)
- **RatedVoltageV** (*int*)
- **TypeName** (*Literal['resistive.heater.cac.gt']*)
- **Version** (*Literal['000']*)

as_dict()

Translate the object into a dictionary representation that can be serialized into a resistive.heater.cac.gt.000 object.

This method prepares the object for serialization by the as_type method, creating a dictionary with key-value pairs that follow the requirements for an instance of the resistive.heater.cac.gt.000 type. Unlike the standard python dict method, it makes the following substantive changes: - Enum Values: Translates between the

values used locally by the actor to the symbol sent in messages. - Removes any key-value pairs where the value is None for a clearer message, especially in cases with many optional attributes.

It also applies these changes recursively to sub-types.

Return type
Dict[str, *Any*]

`as_type()`

Serialize to the resistive.heater.cac.gt.000 representation.

Instances in the class are python-native representations of resistive.heater.cac.gt.000 objects, while the actual resistive.heater.cac.gt.000 object is the serialized UTF-8 byte string designed for sending in a message.

This method calls the as_dict() method, which differs from the native python dict() in the following key ways: - Enum Values: Translates between the values used locally by the actor to the symbol sent in messages. - - Removes any key-value pairs where the value is None for a clearer message, especially in cases with many optional attributes.

It also applies these changes recursively to sub-types.

Its near-inverse is ResistiveHeaterCacGt.type_to_tuple(). If the type (or any sub-types) includes an enum, then the type_to_tuple will map an unrecognized symbol to the default enum value. This is why these two methods are only 'near' inverses.

Return type
bytes

ComponentAttributeClassId:

- Description: ComponentAttributeClassId. Unique identifier for the device class (aka 'cac' or Component Attribute Class). Authority is maintained by the World Registry.
- Format: UuidCanonicalTextual

MakeModel:

- Description:

DisplayName:

- Description:

NameplateMaxPowerW:

- Description:

RatedVoltageV:

- Description:
- Format: PositiveInteger

TypeName:

- Description: All GridWorks Versioned Types have a fixed TypeName, which is a string of lowercase alphanumeric words separated by periods, most significant word (on the left) starting with an alphabet character, and final word NOT all Hindu-Arabic numerals.

Version:

- Description: All GridWorks Versioned Types have a fixed version, which is a string of three Hindu-Arabic numerals.

class gwproto.types.resistive_heater_cac_gt.**check_is_uuid_canonical_textual**(*v*)

Checks UuidCanonicalTextual format

UuidCanonicalTextual format: A string of hex words separated by hyphens of length 8-4-4-4-12.

Parameters
v (*str*) – the candidate

Raises
ValueError – if v is not UuidCanonicalTextual format

class gwproto.types.resistive_heater_cac_gt.**check_is_positive_integer**(*v*)

Must be positive when interpreted as an integer. Interpretation as an integer follows the pydantic rules for this - which will round down rational numbers. So 1.7 will be interpreted as 1 and is also fine, while 0.5 is interpreted as 0 and will raise an exception.

Parameters
v (*int*) – the candidate

Raises
ValueError – if v < 1

class gwproto.types.**ResistiveHeaterCacGt_Maker**(*component_attribute_class_id*, *make_model*, *display_name*, *nameplate_max_power_w*, *rated_voltage_v*)

Parameters

- **component_attribute_class_id** (*str*)
- **make_model** (MakeModel)
- **display_name** (*str | None*)
- **nameplate_max_power_w** (*int*)
- **rated_voltage_v** (*int*)

classmethod **dict_to_tuple**(*d*)

Deserialize a dictionary representation of a resistive.heater.cac.gt.000 message object into a ResistiveHeaterCacGt python object for internal use.

This is the near-inverse of the ResistiveHeaterCacGt.as_dict() method:

- Enums: translates between the symbols sent in messages between actors and

the values used by the actors internally once they've deserialized the messages.

- Types: recursively validates and deserializes sub-types.

Note that if a required attribute with a default value is missing in a dict, this method will raise a SchemaError. This differs from the pydantic BaseModel practice of auto-completing missing attributes with default values when they exist.

Parameters
d (*dict*) – the dictionary resulting from json.loads(t) for a serialized JSON type object t.

Raises
SchemaError – if the dict cannot be turned into a ResistiveHeaterCacGt object.

Returns
ResistiveHeaterCacGt

Return type
ResistiveHeaterCacGt

classmethod tuple_to_type(*tpl*)

Given a Python class object, returns the serialized JSON type object.

Parameters
tpl (ResistiveHeaterCacGt)

Return type
bytes

classmethod type_to_tuple(*t*)

Given a serialized JSON type object, returns the Python class object.

Parameters
t (*bytes*)

Return type
ResistiveHeaterCacGt

1.5.35 ResistiveHeaterComponentGt

Python pydantic class corresponding to json type *resistive.heater.component.gt*, version *000*.

class gwproto.types.**ResistiveHeaterComponentGt**(*, *ComponentId*, *ComponentAttributeClassId*, *DisplayName=None*, *HwUid=None*, *TestedMaxHotMilliOhms=None*, *TestedMaxColdMilliOhms=None*, *TypeName='resistive.heater.component.gt'*, *Version='000'*)

Type for tracking Resistive Heater Components.

Designed for Resistive Heaters. It extends the component.gt.000 type. Authority for the attributes of the component.gt.000 (ComponentId, ComponentAttributeClassId, DisplayName, HwUid) belongs to the WorldRegistry. The WorldRegistry is part of the GridWorks 'BackOffice' structure for managing relational device data . Notably, ComponentId and ComponentAttributeClass are both required and immutable. HwUid is optional but once it is set to a non-null value that is also immutable - it is meant to be an immutable identifier associated to a specific physical device, ideally one that can be read remotely by the SCADA and also by the naked eye. The DisplayName is mutable, with its current value in time governed by the WorldRegistry.

[More info](https://g-node-registry.readthedocs.io/en/latest/component.html)

Parameters

- **ComponentId** (*str*)
- **ComponentAttributeClassId** (*str*)
- **DisplayName** (*str | None*)
- **HwUid** (*str | None*)
- **TestedMaxHotMilliOhms** (*int | None*)
- **TestedMaxColdMilliOhms** (*int | None*)
- **TypeName** (*Literal['resistive.heater.component.gt']*)
- **Version** (*Literal['000']*)

`as_dict()`

Translate the object into a dictionary representation that can be serialized into a resistive.heater.component.gt.000 object.

This method prepares the object for serialization by the as_type method, creating a dictionary with key-value pairs that follow the requirements for an instance of the resistive.heater.component.gt.000 type. Unlike the standard python dict method, it makes the following substantive changes: - Enum Values: Translates between the values used locally by the actor to the symbol sent in messages. - Removes any key-value pairs where the value is None for a clearer message, especially in cases with many optional attributes.

It also applies these changes recursively to sub-types.

Return type
Dict[str, *Any*]

`as_type()`

Serialize to the resistive.heater.component.gt.000 representation.

Instances in the class are python-native representations of resistive.heater.component.gt.000 objects, while the actual resistive.heater.component.gt.000 object is the serialized UTF-8 byte string designed for sending in a message.

This method calls the as_dict() method, which differs from the native python dict() in the following key ways: - Enum Values: Translates between the values used locally by the actor to the symbol sent in messages. - - Removes any key-value pairs where the value is None for a clearer message, especially in cases with many optional attributes.

It also applies these changes recursively to sub-types.

Its near-inverse is ResistiveHeaterComponentGt.type_to_tuple(). If the type (or any sub-types) includes an enum, then the type_to_tuple will map an unrecognized symbol to the default enum value. This is why these two methods are only 'near' inverses.

Return type
bytes

ComponentId:

- Description: Component Id. Primary GridWorks identifier for a specific physical instance of a Resistive-Heater, and also as a more generic Component.
- Format: UuidCanonicalTextual

ComponentAttributeClassId:

- Description: ComponentAttributeClassId. Unique identifier for the device class. Authority for these, as well as the relationship between Components and ComponentAttributeClasses (Cacs) is maintained by the World Registry.
- Format: UuidCanonicalTextual

DisplayName:

- Description:

HwUid:

- Description: Hardware Unique Id.

TestedMaxHotMilliOhms:

- Description:

TestedMaxColdMilliOhms:

- Description:

TypeName:

- Description: All GridWorks Versioned Types have a fixed TypeName, which is a string of lowercase alphanumeric words separated by periods, most significant word (on the left) starting with an alphabet character, and final word NOT all Hindu-Arabic numerals.

Version:

- Description: All GridWorks Versioned Types have a fixed version, which is a string of three Hindu-Arabic numerals.

class gwproto.types.resistive_heater_component_gt.**check_is_uuid_canonical_textual**(*v*)

Checks UuidCanonicalTextual format

UuidCanonicalTextual format: A string of hex words separated by hyphens of length 8-4-4-4-12.

Parameters
v (*str*) – the candidate

Raises
ValueError – if v is not UuidCanonicalTextual format

class gwproto.types.**ResistiveHeaterComponentGt_Maker**(*component_id*, *component_attribute_class_id*, *display_name*, *hw_uid*, *tested_max_hot_milli_ohms*, *tested_max_cold_milli_ohms*)

Parameters

- **component_id** (*str*)
- **component_attribute_class_id** (*str*)
- **display_name** (*str | None*)
- **hw_uid** (*str | None*)
- **tested_max_hot_milli_ohms** (*int | None*)
- **tested_max_cold_milli_ohms** (*int | None*)

classmethod **dict_to_tuple**(*d*)

Deserialize a dictionary representation of a resistive.heater.component.gt.000 message object into a ResistiveHeaterComponentGt python object for internal use.

This is the near-inverse of the ResistiveHeaterComponentGt.as_dict() method:

- Enums: translates between the symbols sent in messages between actors and

the values used by the actors internally once they've deserialized the messages.

- Types: recursively validates and deserializes sub-types.

Note that if a required attribute with a default value is missing in a dict, this method will raise a SchemaError. This differs from the pydantic BaseModel practice of auto-completing missing attributes with default values when they exist.

Parameters
d (*dict*) – the dictionary resulting from json.loads(t) for a serialized JSON type object t.

Raises
SchemaError – if the dict cannot be turned into a ResistiveHeaterComponentGt object.

Returns
ResistiveHeaterComponentGt

Return type
ResistiveHeaterComponentGt

`classmethod tuple_to_type(tpl)`

Given a Python class object, returns the serialized JSON type object.

Parameters
tpl (ResistiveHeaterComponentGt)

Return type
bytes

`classmethod type_to_tuple(t)`

Given a serialized JSON type object, returns the Python class object.

Parameters
t (*bytes*)

Return type
ResistiveHeaterComponentGt

1.5.36 RestPollerCacGt

Python pydantic class corresponding to json type *rest.poller.cac.gt*, version *000*.

ComponentAttributeClassId:

- Description: ComponentAttributeClassId. Unique identifier for the device class (aka 'cac' or Component Attribute Class). Authority is maintained by the World Registry.
- Format: UuidCanonicalTextual

DisplayName:

- Description:

TypeName:

- Description: All GridWorks Versioned Types have a fixed TypeName, which is a string of lowercase alphanumeric words separated by periods, most significant word (on the left) starting with an alphabet character, and final word NOT all Hindu-Arabic numerals.

Version:

- Description: All GridWorks Versioned Types have a fixed version, which is a string of three Hindu-Arabic numerals.

1.5.37 RestPollerComponentGt

Python pydantic class corresponding to json type *rest.poller.component.gt*, version *000*.

TypeName:

- Description: All GridWorks Versioned Types have a fixed TypeName, which is a string of lowercase alphanumeric words separated by periods, most significant word (on the left) starting with an alphabet character, and final word NOT all Hindu-Arabic numerals.

Version:

- Description: All GridWorks Versioned Types have a fixed version, which is a string of three Hindu-Arabic numerals.

1.5.38 SimpleTempSensorCacGt

Python pydantic class corresponding to json type *simple.temp.sensor.cac.gt*, version *000*.

class gwproto.types.**SimpleTempSensorCacGt**(*, *ComponentAttributeClassId*, *MakeModel*, *TypicalResponseTimeMs*, *Exponent*, *TempUnit*, *TelemetryName*, *DisplayName=None*, *CommsMethod=None*, *TypeName='simple.temp.sensor.cac.gt'*, *Version='000'*)

Type for tracking Simple Temp Sensor ComponentAttributeClasses.

GridWorks Spaceheat SCADA uses the GridWorks GNodeRegistry structures and abstractions for managing relational device data. The Cac, or ComponentAttributeClass, is part of this structure.

[More info](https://g-node-registry.readthedocs.io/en/latest/component-attribute-class.html)

Parameters

- **ComponentAttributeClassId** (*str*)
- **MakeModel** (MakeModel)
- **TypicalResponseTimeMs** (*int*)
- **Exponent** (*int*)
- **TempUnit** (Unit)
- **TelemetryName** (TelemetryName)
- **DisplayName** (*str | None*)
- **CommsMethod** (*str | None*)
- **TypeName** (*Literal['simple.temp.sensor.cac.gt']*)
- **Version** (*Literal['000']*)

as_dict()

Translate the object into a dictionary representation that can be serialized into a simple.temp.sensor.cac.gt.000 object.

This method prepares the object for serialization by the as_type method, creating a dictionary with key-value pairs that follow the requirements for an instance of the simple.temp.sensor.cac.gt.000 type. Unlike the standard python dict method, it makes the following substantive changes: - Enum Values: Translates between the values used locally by the actor to the symbol sent in messages. - Removes any key-value pairs where the value is None for a clearer message, especially in cases with many optional attributes.

It also applies these changes recursively to sub-types.

Return type
Dict[str, *Any*]

as_type()

Serialize to the simple.temp.sensor.cac.gt.000 representation.

Instances in the class are python-native representations of simple.temp.sensor.cac.gt.000 objects, while the actual simple.temp.sensor.cac.gt.000 object is the serialized UTF-8 byte string designed for sending in a message.

This method calls the as_dict() method, which differs from the native python dict() in the following key ways: - Enum Values: Translates between the values used locally by the actor to the symbol sent in messages. - - Removes any key-value pairs where the value is None for a clearer message, especially in cases with many optional attributes.

It also applies these changes recursively to sub-types.

Its near-inverse is SimpleTempSensorCacGt.type_to_tuple(). If the type (or any sub-types) includes an enum, then the type_to_tuple will map an unrecognized symbol to the default enum value. This is why these two methods are only 'near' inverses.

Return type
bytes

ComponentAttributeClassId:

- Description: ComponentAttributeClassId. Unique identifier for the device class (aka 'cac' or Component Attribute Class). Authority is maintained by the World Registry.
- Format: UuidCanonicalTextual

MakeModel:

- Description:

TypicalResponseTimeMs:

- Description:

Exponent:

- Description: Exponent. Say the TelemetryName is WaterTempCTimes1000; this corresponds to units of Celsius. To match the implication in the name, the Exponent should be 3, and a Value of 65300 would indicate 65.3 deg C

TempUnit:

- Description:

TelemetryName:

- Description:

DisplayName:

- Description:

CommsMethod:

- Description:

TypeName:

- Description: All GridWorks Versioned Types have a fixed TypeName, which is a string of lowercase alphanumeric words separated by periods, most significant word (on the left) starting with an alphabet character, and final word NOT all Hindu-Arabic numerals.

Version:

- Description: All GridWorks Versioned Types have a fixed version, which is a string of three Hindu-Arabic numerals.

`class gwproto.types.simple_temp_sensor_cac_gt.check_is_uuid_canonical_textual(v)`

Checks UuidCanonicalTextual format

UuidCanonicalTextual format: A string of hex words separated by hyphens of length 8-4-4-4-12.

Parameters
v (*str*) – the candidate

Raises
ValueError – if v is not UuidCanonicalTextual format

class gwproto.types.**SimpleTempSensorCacGt_Maker**(*component_attribute_class_id*, *make_model*, *typical_response_time_ms*, *exponent*, *temp_unit*, *telemetry_name*, *display_name*, *comms_method*)

Parameters

- **component_attribute_class_id** (*str*)
- **make_model** (MakeModel)
- **typical_response_time_ms** (*int*)
- **exponent** (*int*)
- **temp_unit** (Unit)
- **telemetry_name** (TelemetryName)
- **display_name** (*str | None*)
- **comms_method** (*str | None*)

classmethod **dict_to_tuple**(*d*)

Deserialize a dictionary representation of a simple.temp.sensor.cac.gt.000 message object into a SimpleTempSensorCacGt python object for internal use.

This is the near-inverse of the SimpleTempSensorCacGt.as_dict() method:

- Enums: translates between the symbols sent in messages between actors and

the values used by the actors internally once they've deserialized the messages.

- Types: recursively validates and deserializes sub-types.

Note that if a required attribute with a default value is missing in a dict, this method will raise a SchemaError. This differs from the pydantic BaseModel practice of auto-completing missing attributes with default values when they exist.

Parameters
d (*dict*) – the dictionary resulting from json.loads(t) for a serialized JSON type object t.

Raises
SchemaError – if the dict cannot be turned into a SimpleTempSensorCacGt object.

Returns
SimpleTempSensorCacGt

Return type
SimpleTempSensorCacGt

classmethod **tuple_to_type**(*tpl*)

Given a Python class object, returns the serialized JSON type object.

Parameters
tpl (SimpleTempSensorCacGt)

Return type
bytes

classmethod type_to_tuple(*t*)

Given a serialized JSON type object, returns the Python class object.

Parameters
t (*bytes*)

Return type
SimpleTempSensorCacGt

1.5.39 SimpleTempSensorComponentGt

Python pydantic class corresponding to json type *simple.temp.sensor.component.gt*, version *000*.

class gwproto.types.**SimpleTempSensorComponentGt**(*, *ComponentId*, *ComponentAttributeClassId*, *DisplayName=None*, *HwUid=None*, *Channel=None*, *TypeName='simple.temp.sensor.component.gt'*, *Version='000'*)

Type for tracking Simple Temp Sensor Components.

Designed for simple temp sensors that read only one temp. It extends the component.gt.000 type. Authority for the attributes of the component.gt.000 (ComponentId, ComponentAttributeClassId, DisplayName, HwUid) belongs to the WorldRegistry. The WorldRegistry is part of the GridWorks 'BackOffice' structure for managing relational device data . Notably, ComponentId and ComponentAttributeClass are both required and immutable. HwUid is optional but once it is set to a non-null value that is also immutable - it is meant to be an immutable identifier associated to a specific physical device, ideally one that can be read remotely by the SCADA and also by the naked eye. The DisplayName is mutable, with its current value in time governed by the WorldRegistry.

[More info](https://g-node-registry.readthedocs.io/en/latest/component.html)

Parameters

- **ComponentId** (*str*)
- **ComponentAttributeClassId** (*str*)
- **DisplayName** (*str | None*)
- **HwUid** (*str | None*)
- **Channel** (*int | None*)
- **TypeName** (*Literal['simple.temp.sensor.component.gt']*)
- **Version** (*Literal['000']*)

as_dict()

Translate the object into a dictionary representation that can be serialized into a simple.temp.sensor.component.gt.000 object.

This method prepares the object for serialization by the as_type method, creating a dictionary with key-value pairs that follow the requirements for an instance of the simple.temp.sensor.component.gt.000 type. Unlike the standard python dict method, it makes the following substantive changes: - Enum Values: Translates between the values used locally by the actor to the symbol sent in messages. - Removes any key-value pairs where the value is None for a clearer message, especially in cases with many optional attributes.

It also applies these changes recursively to sub-types.

Return type
Dict[str, *Any*]

as_type()

Serialize to the simple.temp.sensor.component.gt.000 representation.

Instances in the class are python-native representations of simple.temp.sensor.component.gt.000 objects, while the actual simple.temp.sensor.component.gt.000 object is the serialized UTF-8 byte string designed for sending in a message.

This method calls the as_dict() method, which differs from the native python dict() in the following key ways: - Enum Values: Translates between the values used locally by the actor to the symbol sent in messages. - - Removes any key-value pairs where the value is None for a clearer message, especially in cases with many optional attributes.

It also applies these changes recursively to sub-types.

Its near-inverse is SimpleTempSensorComponentGt.type_to_tuple(). If the type (or any sub-types) includes an enum, then the type_to_tuple will map an unrecognized symbol to the default enum value. This is why these two methods are only 'near' inverses.

Return type
bytes

ComponentId:

- Description: Component Id. Primary GridWorks identifier for a specific physical instance of a SimpleTempSensor, and also as a more generic Component.
- Format: UuidCanonicalTextual

ComponentAttributeClassId:

- Description: ComponentAttributeClassId. Unique identifier for the device class. Authority for these, as well as the relationship between Components and ComponentAttributeClasses (Cacs) is maintained by the World Registry.
- Format: UuidCanonicalTextual

DisplayName:

- Description:

HwUid:

- Description: Hardware Unique Id.

Channel:

- Description:

TypeName:

- Description: All GridWorks Versioned Types have a fixed TypeName, which is a string of lowercase alphanumeric words separated by periods, most significant word (on the left) starting with an alphabet character, and final word NOT all Hindu-Arabic numerals.

Version:

- Description: All GridWorks Versioned Types have a fixed version, which is a string of three Hindu-Arabic numerals.

class gwproto.types.simple_temp_sensor_component_gt.check_is_uuid_canonical_textual(*v*)

Checks UuidCanonicalTextual format

UuidCanonicalTextual format: A string of hex words separated by hyphens of length 8-4-4-4-12.

Parameters
v (*str*) – the candidate

Raises
ValueError – if v is not UuidCanonicalTextual format

class gwproto.types.**SimpleTempSensorComponentGt_Maker**(*component_id*, *component_attribute_class_id*, *display_name*, *hw_uid*, *channel*)

Parameters

- **component_id** (*str*)
- **component_attribute_class_id** (*str*)
- **display_name** (*str | None*)
- **hw_uid** (*str | None*)
- **channel** (*int | None*)

classmethod dict_to_tuple(*d*)

Deserialize a dictionary representation of a simple.temp.sensor.component.gt.000 message object into a SimpleTempSensorComponentGt python object for internal use.

This is the near-inverse of the SimpleTempSensorComponentGt.as_dict() method:

- Enums: translates between the symbols sent in messages between actors and

the values used by the actors internally once they've deserialized the messages.

- Types: recursively validates and deserializes sub-types.

Note that if a required attribute with a default value is missing in a dict, this method will raise a SchemaError. This differs from the pydantic BaseModel practice of auto-completing missing attributes with default values when they exist.

Parameters
d (*dict*) – the dictionary resulting from json.loads(t) for a serialized JSON type object t.

Raises
SchemaError – if the dict cannot be turned into a SimpleTempSensorComponentGt object.

Returns
SimpleTempSensorComponentGt

Return type
SimpleTempSensorComponentGt

classmethod tuple_to_type(*tpl*)

Given a Python class object, returns the serialized JSON type object.

Parameters
tpl (SimpleTempSensorComponentGt)

Return type
bytes

classmethod type_to_tuple(*t*)

Given a serialized JSON type object, returns the Python class object.

Parameters
t (*bytes*)

Return type
SimpleTempSensorComponentGt

1.5.40 SnapshotSpaceheat

Python pydantic class corresponding to json type *snapshot.spaceheat*, version *000*.

class gwproto.types.**SnapshotSpaceheat**(*, *FromGNodeAlias*, *FromGNodeInstanceId*, *Snapshot*, *TypeName='snapshot.spaceheat'*, *Version='000'*)

Parameters

- **FromGNodeAlias** (*str*)
- **FromGNodeInstanceId** (*str*)
- **Snapshot** (TelemetrySnapshotSpaceheat)
- **TypeName** (*Literal['snapshot.spaceheat']*)
- **Version** (*Literal['000']*)

as_dict()

Translate the object into a dictionary representation that can be serialized into a snapshot.spaceheat.000 object.

This method prepares the object for serialization by the as_type method, creating a dictionary with key-value pairs that follow the requirements for an instance of the snapshot.spaceheat.000 type. Unlike the standard python dict method, it makes the following substantive changes: - Enum Values: Translates between the values used locally by the actor to the symbol sent in messages. - Removes any key-value pairs where the value is None for a clearer message, especially in cases with many optional attributes.

It also applies these changes recursively to sub-types.

Return type
Dict[str, *Any*]

as_type()

Serialize to the snapshot.spaceheat.000 representation.

Instances in the class are python-native representations of snapshot.spaceheat.000 objects, while the actual snapshot.spaceheat.000 object is the serialized UTF-8 byte string designed for sending in a message.

This method calls the as_dict() method, which differs from the native python dict() in the following key ways: - Enum Values: Translates between the values used locally by the actor to the symbol sent in messages. - - Removes any key-value pairs where the value is None for a clearer message, especially in cases with many optional attributes.

It also applies these changes recursively to sub-types.

Its near-inverse is SnapshotSpaceheat.type_to_tuple(). If the type (or any sub-types) includes an enum, then the type_to_tuple will map an unrecognized symbol to the default enum value. This is why these two methods are only 'near' inverses.

Return type
bytes

FromGNodeAlias:

- Description:
- Format: LeftRightDot

FromGNodeInstanceId:

- Description:
- Format: UuidCanonicalTextual

Snapshot:

- Description:

TypeName:

- Description: All GridWorks Versioned Types have a fixed TypeName, which is a string of lowercase alphanumeric words separated by periods, most significant word (on the left) starting with an alphabet character, and final word NOT all Hindu-Arabic numerals.

Version:

- Description: All GridWorks Versioned Types have a fixed version, which is a string of three Hindu-Arabic numerals.

class gwproto.types.snapshot_spaceheat.**check_is_uuid_canonical_textual**(*v*)

Checks UuidCanonicalTextual format

UuidCanonicalTextual format: A string of hex words separated by hyphens of length 8-4-4-4-12.

Parameters
v (*str*) – the candidate

Raises
ValueError – if v is not UuidCanonicalTextual format

class gwproto.types.snapshot_spaceheat.**check_is_left_right_dot**(*v*)

Checks LeftRightDot Format

LeftRightDot format: Lowercase alphanumeric words separated by periods, with the most significant word (on the left) starting with an alphabet character.

Parameters
v (*str*) – the candidate

Raises
ValueError – if v is not LeftRightDot format

class gwproto.types.**SnapshotSpaceheat_Maker**(*from_g_node_alias*, *from_g_node_instance_id*, *snapshot*)

Parameters

- **from_g_node_alias** (*str*)
- **from_g_node_instance_id** (*str*)
- **snapshot** (TelemetrySnapshotSpaceheat)

classmethod **dict_to_tuple**(*d*)

Deserialize a dictionary representation of a snapshot.spaceheat.000 message object into a SnapshotSpaceheat python object for internal use.

This is the near-inverse of the SnapshotSpaceheat.as_dict() method:

- Enums: translates between the symbols sent in messages between actors and

the values used by the actors internally once they've deserialized the messages.

- Types: recursively validates and deserializes sub-types.

Note that if a required attribute with a default value is missing in a dict, this method will raise a SchemaError. This differs from the pydantic BaseModel practice of auto-completing missing attributes with default values when they exist.

Parameters
d (*dict*) – the dictionary resulting from json.loads(t) for a serialized JSON type object t.

Raises
SchemaError – if the dict cannot be turned into a SnapshotSpaceheat object.

Returns
SnapshotSpaceheat

Return type
SnapshotSpaceheat

classmethod tuple_to_type(*tpl*)

Given a Python class object, returns the serialized JSON type object.

Parameters
tpl (SnapshotSpaceheat)

Return type
bytes

classmethod type_to_tuple(*t*)

Given a serialized JSON type object, returns the Python class object.

Parameters
t (*bytes*)

Return type
SnapshotSpaceheat

1.5.41 SpaceheatNodeGt

Python pydantic class corresponding to json type *spaceheat.node.gt*, version *100*.

class gwproto.types.**SpaceheatNodeGt**(**, ShNodeId, Alias, ActorClass, Role, DisplayName=None, ComponentId=None, ReportingSamplePeriodS=None, RatedVoltageV=None, TypicalVoltageV=None, InPowerMetering=None, TypeName='spaceheat.node.gt', Version='100'*)

Spaceheat Node.

A SpaceheatNode, or ShNode, is an organizing principal for the SCADA software. ShNodes can represent both underlying physical objects (water tank), measurements of these objects (temperature sensing at the top of a water tank), and actors within the code (an actor measuring multiple temperatures, or an actor responsible for filtering/smoothing temperature data for the purposes of thermostatic control).

[More info](https://gridworks-protocol.readthedocs.io/en/latest/spaceheat-node.html)

Parameters

- **ShNodeId** (*str*)
- **Alias** (*str*)
- **ActorClass** (ActorClass)
- **Role** (Role)

- **DisplayName** (*str | None*)
- **ComponentId** (*str | None*)
- **ReportingSamplePeriodS** (*int | None*)
- **RatedVoltageV** (*int | None*)
- **TypicalVoltageV** (*int | None*)
- **InPowerMetering** (*bool | None*)
- **TypeName** (*Literal['spaceheat.node.gt']*)
- **Version** (*Literal['100']*)

as_dict()

Translate the object into a dictionary representation that can be serialized into a spaceheat.node.gt.100 object.

This method prepares the object for serialization by the as_type method, creating a dictionary with key-value pairs that follow the requirements for an instance of the spaceheat.node.gt.100 type. Unlike the standard python dict method, it makes the following substantive changes: - Enum Values: Translates between the values used locally by the actor to the symbol sent in messages. - Removes any key-value pairs where the value is None for a clearer message, especially in cases with many optional attributes.

It also applies these changes recursively to sub-types.

Return type
Dict[str, *Any*]

as_type()

Serialize to the spaceheat.node.gt.100 representation.

Instances in the class are python-native representations of spaceheat.node.gt.100 objects, while the actual spaceheat.node.gt.100 object is the serialized UTF-8 byte string designed for sending in a message.

This method calls the as_dict() method, which differs from the native python dict() in the following key ways: - Enum Values: Translates between the values used locally by the actor to the symbol sent in messages. - - Removes any key-value pairs where the value is None for a clearer message, especially in cases with many optional attributes.

It also applies these changes recursively to sub-types.

Its near-inverse is SpaceheatNodeGt.type_to_tuple(). If the type (or any sub-types) includes an enum, then the type_to_tuple will map an unrecognized symbol to the default enum value. This is why these two methods are only 'near' inverses.

Return type
bytes

ShNodeId:

- Description:
- Format: UuidCanonicalTextual

Alias:

- Description:
- Format: LeftRightDot

ActorClass:

- Description:

Role:

- Description:

DisplayName:

- Description:

ComponentId:

- Description: Unique identifier for Spaceheat Node's Component. Used if a Spaceheat Node is associated with a physical device.
- Format: UuidCanonicalTextual

ReportingSamplePeriodS:

- Description:

RatedVoltageV:

- Description:
- Format: PositiveInteger

TypicalVoltageV:

- Description:
- Format: PositiveInteger

InPowerMetering:

- Description:

TypeName:

- Description: All GridWorks Versioned Types have a fixed TypeName, which is a string of lowercase alphanumeric words separated by periods, most significant word (on the left) starting with an alphabet character, and final word NOT all Hindu-Arabic numerals.

Version:

- Description: All GridWorks Versioned Types have a fixed version, which is a string of three Hindu-Arabic numerals.

`class gwproto.types.spaceheat_node_gt.check_is_uuid_canonical_textual(v)`

Checks UuidCanonicalTextual format

UuidCanonicalTextual format: A string of hex words separated by hyphens of length 8-4-4-4-12.

Parameters
v (*str*) – the candidate

Raises
ValueError – if v is not UuidCanonicalTextual format

`class gwproto.types.spaceheat_node_gt.check_is_positive_integer(v)`

Must be positive when interpreted as an integer. Interpretation as an integer follows the pydantic rules for this - which will round down rational numbers. So 1.7 will be interpreted as 1 and is also fine, while 0.5 is interpreted as 0 and will raise an exception.

Parameters
v (*int*) – the candidate

Raises
ValueError – if v < 1

class gwproto.types.spaceheat_node_gt.**check_is_left_right_dot**(*v*)

Checks LeftRightDot Format

LeftRightDot format: Lowercase alphanumeric words separated by periods, with the most significant word (on the left) starting with an alphabet character.

Parameters
v (*str*) – the candidate

Raises
ValueError – if v is not LeftRightDot format

class gwproto.types.**SpaceheatNodeGt_Maker**(*sh_node_id*, *alias*, *actor_class*, *role*, *display_name*, *component_id*, *reporting_sample_period_s*, *rated_voltage_v*, *typical_voltage_v*, *in_power_metering*)

Parameters

- **sh_node_id** (*str*)
- **alias** (*str*)
- **actor_class** (ActorClass)
- **role** (Role)
- **display_name** (*str | None*)
- **component_id** (*str | None*)
- **reporting_sample_period_s** (*int | None*)
- **rated_voltage_v** (*int | None*)
- **typical_voltage_v** (*int | None*)
- **in_power_metering** (*bool | None*)

classmethod **dict_to_tuple**(*d*)

Deserialize a dictionary representation of a spaceheat.node.gt.100 message object into a SpaceheatNodeGt python object for internal use.

This is the near-inverse of the SpaceheatNodeGt.as_dict() method:

- Enums: translates between the symbols sent in messages between actors and

the values used by the actors internally once they've deserialized the messages.

- Types: recursively validates and deserializes sub-types.

Note that if a required attribute with a default value is missing in a dict, this method will raise a SchemaError. This differs from the pydantic BaseModel practice of auto-completing missing attributes with default values when they exist.

Parameters
d (*dict*) – the dictionary resulting from json.loads(t) for a serialized JSON type object t.

Raises
SchemaError – if the dict cannot be turned into a SpaceheatNodeGt object.

Returns
SpaceheatNodeGt

Return type
SpaceheatNodeGt

classmethod tuple_to_type(*tpl*)

Given a Python class object, returns the serialized JSON type object.

Parameters
tpl (SpaceheatNodeGt)

Return type
bytes

classmethod type_to_tuple(*t*)

Given a serialized JSON type object, returns the Python class object.

Parameters
t (*bytes*)

Return type
SpaceheatNodeGt

1.5.42 TaDataChannels

Python pydantic class corresponding to json type *ta.data.channels*, version *000*.

class gwproto.types.**TaDataChannels**(**, TerminalAssetGNodeAlias, TerminalAssetGNodeId, TimeUnixS, Author, Channels, Identifier, TypeName='ta.data.channels', Version='000'*)

Terminal Asset Data Channels.

A list of data channels associated to a specific Terminal Asset.

Parameters

- **TerminalAssetGNodeAlias** (*str*)
- **TerminalAssetGNodeId** (*str*)
- **TimeUnixS** (*int*)
- **Author** (*str*)
- **Channels** (*List[DataChannel]*)
- **Identifier** (*str*)
- **TypeName** (*Literal['ta.data.channels']*)
- **Version** (*Literal['000']*)

as_dict()

Translate the object into a dictionary representation that can be serialized into a ta.data.channels.000 object.

This method prepares the object for serialization by the as_type method, creating a dictionary with key-value pairs that follow the requirements for an instance of the ta.data.channels.000 type. Unlike the standard python dict method, it makes the following substantive changes: - Enum Values: Translates between the values used locally by the actor to the symbol sent in messages. - Removes any key-value pairs where the value is None for a clearer message, especially in cases with many optional attributes.

It also applies these changes recursively to sub-types.

Return type
Dict[str, *Any*]

`as_type()`

Serialize to the ta.data.channels.000 representation.

Instances in the class are python-native representations of ta.data.channels.000 objects, while the actual ta.data.channels.000 object is the serialized UTF-8 byte string designed for sending in a message.

This method calls the as_dict() method, which differs from the native python dict() in the following key ways: - Enum Values: Translates between the values used locally by the actor to the symbol sent in messages. - - Removes any key-value pairs where the value is None for a clearer message, especially in cases with many optional attributes.

It also applies these changes recursively to sub-types.

Its near-inverse is TaDataChannels.type_to_tuple(). If the type (or any sub-types) includes an enum, then the type_to_tuple will map an unrecognized symbol to the default enum value. This is why these two methods are only 'near' inverses.

Return type
bytes

TerminalAssetGNodeAlias:

- Description: GNodeAlias for the Terminal Asset. The Alias of the Terminal Asset about which the time series data is providing information.
- Format: LeftRightDot

TerminalAssetGNodeId:

- Description: GNodeId for the Terminal Asset. The immutable unique identifier for the Terminal Asset.
- Format: UuidCanonicalTextual

TimeUnixS:

- Description: TimeUnixS. The time that this list of data channels was created
- Format: ReasonableUnixTimeS

Author:

- Description: Author of this list of data channels.

Channels:

- Description: The list of data channels.

Identifier:

- Description: Identifier. Unique identifier for a specific instance of this type that can be used to establish how time series csv's were constructed.
- Format: UuidCanonicalTextual

TypeName:

- Description: All GridWorks Versioned Types have a fixed TypeName, which is a string of lowercase alphanumeric words separated by periods, most significant word (on the left) starting with an alphabet character, and final word NOT all Hindu-Arabic numerals.

Version:

- Description: All GridWorks Versioned Types have a fixed version, which is a string of three Hindu-Arabic numerals.

class gwproto.types.ta_data_channels.**check_is_reasonable_unix_time_s**(*v*)

Checks ReasonableUnixTimeS format

ReasonableUnixTimeS format: unix seconds between Jan 1 2000 and Jan 1 3000

Parameters
v (*int*) – the candidate

Raises
ValueError – if v is not ReasonableUnixTimeS format

class gwproto.types.ta_data_channels.**check_is_uuid_canonical_textual**(*v*)

Checks UuidCanonicalTextual format

UuidCanonicalTextual format: A string of hex words separated by hyphens of length 8-4-4-4-12.

Parameters
v (*str*) – the candidate

Raises
ValueError – if v is not UuidCanonicalTextual format

class gwproto.types.ta_data_channels.**check_is_left_right_dot**(*v*)

Checks LeftRightDot Format

LeftRightDot format: Lowercase alphanumeric words separated by periods, with the most significant word (on the left) starting with an alphabet character.

Parameters
v (*str*) – the candidate

Raises
ValueError – if v is not LeftRightDot format

class gwproto.types.**TaDataChannels_Maker**(*terminal_asset_g_node_alias*, *terminal_asset_g_node_id*, *time_unix_s*, *author*, *channels*, *identifier*)

Parameters

- **terminal_asset_g_node_alias** (*str*)
- **terminal_asset_g_node_id** (*str*)
- **time_unix_s** (*int*)
- **author** (*str*)
- **channels** (*List[DataChannel]*)
- **identifier** (*str*)

classmethod **dict_to_tuple**(*d*)

Deserialize a dictionary representation of a ta.data.channels.000 message object into a TaDataChannels python object for internal use.

This is the near-inverse of the TaDataChannels.as_dict() method:

- Enums: translates between the symbols sent in messages between actors and

the values used by the actors internally once they've deserialized the messages.

- Types: recursively validates and deserializes sub-types.

Note that if a required attribute with a default value is missing in a dict, this method will raise a SchemaError. This differs from the pydantic BaseModel practice of auto-completing missing attributes with default values when they exist.

Parameters
d (*dict*) – the dictionary resulting from json.loads(t) for a serialized JSON type object t.

Raises
SchemaError – if the dict cannot be turned into a TaDataChannels object.

Returns
TaDataChannels

Return type
TaDataChannels

classmethod tuple_to_type(*tpl*)

Given a Python class object, returns the serialized JSON type object.

Parameters
tpl (TaDataChannels)

Return type
bytes

classmethod type_to_tuple(*t*)

Given a serialized JSON type object, returns the Python class object.

Parameters
t (*bytes*)

Return type
TaDataChannels

1.5.43 TelemetryReportingConfig

Python pydantic class corresponding to json type *telemetry.reporting.config*, version *000*.

class gwproto.types.**TelemetryReportingConfig**(*, *TelemetryName*, *AboutNodeName*, *ReportOnChange*, *SamplePeriodS*, *Exponent*, *Unit*, *AsyncReportThreshold=None*, *NameplateMaxValue=None*, *TypeName='telemetry.reporting.config'*, *Version='000'*)

Parameters

- **TelemetryName** (TelemetryName)
- **AboutNodeName** (*str*)
- **ReportOnChange** (*bool*)
- **SamplePeriodS** (*int*)
- **Exponent** (*int*)
- **Unit** (Unit)
- **AsyncReportThreshold** (*float | None*)
- **NameplateMaxValue** (*int | None*)
- **TypeName** (*Literal['telemetry.reporting.config']*)

- **Version** (*Literal['000']*)

as_dict()

Translate the object into a dictionary representation that can be serialized into a telemetry.reporting.config.000 object.

This method prepares the object for serialization by the as_type method, creating a dictionary with key-value pairs that follow the requirements for an instance of the telemetry.reporting.config.000 type. Unlike the standard python dict method, it makes the following substantive changes: - Enum Values: Translates between the values used locally by the actor to the symbol sent in messages. - Removes any key-value pairs where the value is None for a clearer message, especially in cases with many optional attributes.

It also applies these changes recursively to sub-types.

Return type
Dict[str, *Any*]

as_type()

Serialize to the telemetry.reporting.config.000 representation.

Instances in the class are python-native representations of telemetry.reporting.config.000 objects, while the actual telemetry.reporting.config.000 object is the serialized UTF-8 byte string designed for sending in a message.

This method calls the as_dict() method, which differs from the native python dict() in the following key ways: - Enum Values: Translates between the values used locally by the actor to the symbol sent in messages. - - Removes any key-value pairs where the value is None for a clearer message, especially in cases with many optional attributes.

It also applies these changes recursively to sub-types.

Its near-inverse is TelemetryReportingConfig.type_to_tuple(). If the type (or any sub-types) includes an enum, then the type_to_tuple will map an unrecognized symbol to the default enum value. This is why these two methods are only 'near' inverses.

Return type
bytes

classmethod check_axiom_1(*v*)

Axiom 1: Async reporting consistency. If AsyncReportThreshold exists, so does NameplateMaxValue

Parameters
v (*dict*)

Return type
dict

TelemetryName:

- Description:

AboutNodeName:

- Description: The name of the SpaceheatNode whose physical quantity is getting captured.
- Format: LeftRightDot

ReportOnChange:

- Description:

SamplePeriodS:

- Description:

Exponent:

- Description: Exponent. Say the TelemetryName is WaterTempCTimes1000; this corresponds to units of Celsius. To match the implication in the name, the Exponent should be 3, and a Value of 65300 would indicate 65.3 deg C

Unit:

- Description:

AsyncReportThreshold:

- Description:

NameplateMaxValue:

- Description:
- Format: PositiveInteger

TypeName:

- Description: All GridWorks Versioned Types have a fixed TypeName, which is a string of lowercase alphanumeric words separated by periods, most significant word (on the left) starting with an alphabet character, and final word NOT all Hindu-Arabic numerals.

Version:

- Description: All GridWorks Versioned Types have a fixed version, which is a string of three Hindu-Arabic numerals.

class gwproto.types.telemetry_reporting_config.**check_is_positive_integer**(*v*)

Must be positive when interpreted as an integer. Interpretation as an integer follows the pydantic rules for this - which will round down rational numbers. So 1.7 will be interpreted as 1 and is also fine, while 0.5 is interpreted as 0 and will raise an exception.

Parameters
v (*int*) – the candidate

Raises
ValueError – if v < 1

class gwproto.types.telemetry_reporting_config.**check_is_left_right_dot**(*v*)

Checks LeftRightDot Format

LeftRightDot format: Lowercase alphanumeric words separated by periods, with the most significant word (on the left) starting with an alphabet character.

Parameters
v (*str*) – the candidate

Raises
ValueError – if v is not LeftRightDot format

class gwproto.types.**TelemetryReportingConfig_Maker**(*telemetry_name*, *about_node_name*, *report_on_change*, *sample_period_s*, *exponent*, *unit*, *async_report_threshold*, *nameplate_max_value*)

Parameters

- **telemetry_name** (TelemetryName)
- **about_node_name** (*str*)

- **report_on_change** (*bool*)
- **sample_period_s** (*int*)
- **exponent** (*int*)
- **unit** (Unit)
- **async_report_threshold** (*float | None*)
- **nameplate_max_value** (*int | None*)

classmethod dict_to_tuple(*d*)

Deserialize a dictionary representation of a telemetry.reporting.config.000 message object into a TelemetryReportingConfig python object for internal use.

This is the near-inverse of the TelemetryReportingConfig.as_dict() method:

- Enums: translates between the symbols sent in messages between actors and

the values used by the actors internally once they've deserialized the messages.

- Types: recursively validates and deserializes sub-types.

Note that if a required attribute with a default value is missing in a dict, this method will raise a SchemaError. This differs from the pydantic BaseModel practice of auto-completing missing attributes with default values when they exist.

Parameters
d (*dict*) – the dictionary resulting from json.loads(t) for a serialized JSON type object t.

Raises
SchemaError – if the dict cannot be turned into a TelemetryReportingConfig object.

Returns
TelemetryReportingConfig

Return type
TelemetryReportingConfig

classmethod tuple_to_type(*tpl*)

Given a Python class object, returns the serialized JSON type object.

Parameters
tpl (TelemetryReportingConfig)

Return type
bytes

classmethod type_to_tuple(*t*)

Given a serialized JSON type object, returns the Python class object.

Parameters
t (*bytes*)

Return type
TelemetryReportingConfig

1.5.44 TelemetrySnapshotSpaceheat

Python pydantic class corresponding to json type *telemetry.snapshot.spaceheat*, version *000*.

class gwproto.types.**TelemetrySnapshotSpaceheat**(*, *ReportTimeUnixMs*, *AboutNodeAliasList*, *ValueList*, *TelemetryNameList*, *TypeName='telemetry.snapshot.spaceheat'*, *Version='000'*)

Snapshot of Telemetry Data from a SpaceHeat SCADA.

A snapshot of all current sensed states, sent from a spaceheat SCADA to its AtomicTNode. The nth element of each of the three lists refer to the same reading (i.e., what is getting read, what the value is, what the TelemetryNames are.)

[More info](https://gridworks-protocol.readthedocs.io/en/latest/spaceheat-node.html)

Parameters

- **ReportTimeUnixMs** (*int*)
- **AboutNodeAliasList** (*List[str]*)
- **ValueList** (*List[int]*)
- **TelemetryNameList** (*List[TelemetryName]*)
- **TypeName** (*Literal['telemetry.snapshot.spaceheat']*)
- **Version** (*Literal['000']*)

as_dict()

Translate the object into a dictionary representation that can be serialized into a telemetry.snapshot.spaceheat.000 object.

This method prepares the object for serialization by the as_type method, creating a dictionary with key-value pairs that follow the requirements for an instance of the telemetry.snapshot.spaceheat.000 type. Unlike the standard python dict method, it makes the following substantive changes: - Enum Values: Translates between the values used locally by the actor to the symbol sent in messages. - Removes any key-value pairs where the value is None for a clearer message, especially in cases with many optional attributes.

It also applies these changes recursively to sub-types.

Return type
Dict[str, *Any*]

as_type()

Serialize to the telemetry.snapshot.spaceheat.000 representation.

Instances in the class are python-native representations of telemetry.snapshot.spaceheat.000 objects, while the actual telemetry.snapshot.spaceheat.000 object is the serialized UTF-8 byte string designed for sending in a message.

This method calls the as_dict() method, which differs from the native python dict() in the following key ways: - Enum Values: Translates between the values used locally by the actor to the symbol sent in messages. - - Removes any key-value pairs where the value is None for a clearer message, especially in cases with many optional attributes.

It also applies these changes recursively to sub-types.

Its near-inverse is TelemetrySnapshotSpaceheat.type_to_tuple(). If the type (or any sub-types) includes an enum, then the type_to_tuple will map an unrecognized symbol to the default enum value. This is why these two methods are only 'near' inverses.

Return type
bytes

classmethod check_axiom_1(*v*)

Axiom 1: ListLengthConsistency. AboutNodeAliasList, ValueList and TelemetryNameList must all have the same length.

Parameters
v (*dict*)

Return type
dict

ReportTimeUnixMs:

- Description: ReportTimeUnixMs. The time, in unix ms, that the SCADA creates this type. It may not be when the SCADA sends the type to the atn (for example if Internet is down).
- Format: ReasonableUnixTimeMs

AboutNodeAliasList:

- Description: AboutNodeAliases. The list of Spaceheat nodes in the snapshot.
- Format: LeftRightDot

ValueList:

- Description: ValueList.

TelemetryNameList:

- Description:

TypeName:

- Description: All GridWorks Versioned Types have a fixed TypeName, which is a string of lowercase alphanumeric words separated by periods, most significant word (on the left) starting with an alphabet character, and final word NOT all Hindu-Arabic numerals.

Version:

- Description: All GridWorks Versioned Types have a fixed version, which is a string of three Hindu-Arabic numerals.

class gwproto.types.telemetry_snapshot_spaceheat.check_is_left_right_dot(*v*)

Checks LeftRightDot Format

LeftRightDot format: Lowercase alphanumeric words separated by periods, with the most significant word (on the left) starting with an alphabet character.

Parameters
v (*str*) – the candidate

Raises
ValueError – if v is not LeftRightDot format

class gwproto.types.telemetry_snapshot_spaceheat.check_is_reasonable_unix_time_ms(*v*)

Checks ReasonableUnixTimeMs format

ReasonableUnixTimeMs format: unix milliseconds between Jan 1 2000 and Jan 1 3000

Parameters
v (*int*) – the candidate

Raises
ValueError – if v is not ReasonableUnixTimeMs format

class gwproto.types.**TelemetrySnapshotSpaceheat_Maker**(*report_time_unix_ms*, *about_node_alias_list*, *value_list*, *telemetry_name_list*)

Parameters

- **report_time_unix_ms** (*int*)
- **about_node_alias_list** (*List[str]*)
- **value_list** (*List[int]*)
- **telemetry_name_list** (*List[TelemetryName]*)

classmethod **dict_to_tuple**(*d*)

Deserialize a dictionary representation of a telemetry.snapshot.spaceheat.000 message object into a TelemetrySnapshotSpaceheat python object for internal use.

This is the near-inverse of the TelemetrySnapshotSpaceheat.as_dict() method:

- Enums: translates between the symbols sent in messages between actors and

the values used by the actors internally once they've deserialized the messages.

- Types: recursively validates and deserializes sub-types.

Note that if a required attribute with a default value is missing in a dict, this method will raise a SchemaError. This differs from the pydantic BaseModel practice of auto-completing missing attributes with default values when they exist.

Parameters
d (*dict*) – the dictionary resulting from json.loads(t) for a serialized JSON type object t.

Raises
SchemaError – if the dict cannot be turned into a TelemetrySnapshotSpaceheat object.

Returns
TelemetrySnapshotSpaceheat

Return type
TelemetrySnapshotSpaceheat

classmethod **tuple_to_type**(*tpl*)

Given a Python class object, returns the serialized JSON type object.

Parameters
tpl (TelemetrySnapshotSpaceheat)

Return type
bytes

classmethod **type_to_tuple**(*t*)

Given a serialized JSON type object, returns the Python class object.

Parameters
t (*bytes*)

Return type
TelemetrySnapshotSpaceheat

1.6 Contributor Guide

Thank you for your interest in improving this project. This project is open-source under the MIT license and welcomes contributions in the form of bug reports, feature requests, and pull requests.

Here is a list of important resources for contributors:

- Source Code
- Documentation
- Issue Tracker
- *Code of Conduct*

1.6.1 How to report a bug

Report bugs on the Issue Tracker.

When filing an issue, make sure to answer these questions:

- Which operating system and Python version are you using?
- Which version of this project are you using?
- What did you do?
- What did you expect to see?
- What did you see instead?

The best way to get your bug fixed is to provide a test case, and/or steps to reproduce the issue.

1.6.2 How to request a feature

Request features on the Issue Tracker.

1.6.3 How to set up your development environment

You need Python 3.7+ and the following tools:

- Poetry
- Nox
- nox-poetry

Install the package with development requirements:

```
$ poetry install
```

You can now run an interactive Python session, or the command-line interface:

```
$ poetry run python
$ poetry run gridworks-protocol
```

1.6.4 How to test the project

Run the full test suite:

```
$ nox
```

List the available Nox sessions:

```
$ nox --list-sessions
```

You can also run a specific Nox session. For example, invoke the unit test suite like this:

```
$ nox --session=tests
```

Unit tests are located in the *tests* directory, and are written using the pytest testing framework.

1.6.5 How to submit changes

Open a pull request to submit changes to this project.

Your pull request needs to meet the following guidelines for acceptance:

- The Nox test suite must pass without errors and warnings.
- Include unit tests. This project maintains 100% code coverage.
- If your changes add functionality, update the documentation accordingly.

Feel free to submit early, though—we can always iterate on this.

To run linting and code formatting checks before committing your change, you can install pre-commit as a Git hook by running the following command:

```
$ nox --session=pre-commit -- install
```

It is recommended to open an issue before starting work on anything. This will allow a chance to talk it over with the owners and validate your approach.

1.7 GridWorks Energy Consulting Code of Conduct

1.7.1 Basic Truth

All humans are worthy.

1.7.2 Scope

This Code of Conduct applies to moderation of comments, issues and commits within this repository to support its alignment to the above basic truth.

1.7.3 Enforcement Responsibilities

GridWorks Energy Consulting LLC (gridworks@gridworks-consulting.com) owns and administers this repository, and is ultimately responsible for enforcement of standards of behavior. They are responsible for merges to dev and main branches, and maintain the right and responsibility to remove, edit, or reject comments, commits, code, docuentation edits, issues, and other contributions that are not aligned to this Code of Conduct, and will communicate reasons for moderation decisions when appropriate.

If you read something in this repo that you want GridWorks to consider moderating, please send an email to them at gridworks@gridworks-consulting.com. All complaints will be reviewed and investigated, and GridWorks will respect the privacy and security of the reporter of any incident.

1.7.4 What not to add to this repo

Ways to trigger GridWorks moderation enforcement:

- Publish others' private information, such as a physical or email address, without their explicit permission
- Use of sexualized language or imagery, or make sexual advances
- Troll

1.7.5 Suggestions

- Empathize
- Recognize you are worthy of contributing, and do so in the face of confusion and doubt; you can help clarify things for everyone
- Be interested in differing opinions, viewpoints, and experiences
- Give and accept constructive feedback
- Accept responsibility for your mistakes and learn from them
- Recognize everybody makes mistakes, and forgive
- Focus on the highest good for all

1.7.6 Enforcement Escalation

1. Correction

A private, written request from GridWorks to change or edit a comment, commit, or issue.

2. Warning

With a warning, GridWorks may remove your comments, commits or issues. They may also freeze a conversation.

3. Temporary Ban

A temporary ban from any sort of interaction or public communication within the repository for a specified period of time. No public or private interaction with the people involved, including unsolicited interaction with those enforcing the Code of Conduct, is allowed during this period. Violating these terms may lead to a permanent ban.

4. Permanent Ban

A permanent ban from any sort of interaction within the repository.

1.7.7 Attribution

This Code of Conduct is loosely adapted from the Contributor Covenant, version 2.1, available at https://www.contributor-covenant.org/version/2/1/code_of_conduct.html.

Community Impact Guidelines were inspired by Mozilla's code of conduct enforcement ladder.

For answers to common questions about this code of conduct, see the FAQ at https://www.contributor-covenant.org/faq. Translations are available at https://www.contributor-covenant.org/translations.

1.8 License

PYTHON MODULE INDEX

g

INDEX

A

C

D

E

F

G

H

L

M

P

R

S

T

U

V