Gridworks

GridWorks

Apr 24, 2024

MILLINOCKET, ME

GridWorks uses distributed actors to balance the electric grid. What does this mean? In today's world, more power comes from highly variable power sources such as wind and solar. And yet, the number of electrons going into the grid must match the number coming out. This is where GridWorks comes in. GridWorks technology enables electrical devices with some embedded storage or with flexibility to provide grid balancing. Furthermore, GridWorks allows these appliances to be more thrifty, using electricity when it is cheap and green.

To learn how using and contributing to GridWorks can support a cost-effective and rapid transition to a sustainable future:

- Try some simple Hello World examples;
- Read the Millinocket Story to learn how to exploit the synergy between wind power and space heating;
- Go through the partner Millinocket Tutorial.

CHAPTER

ONE

BLOCKCHAIN MECHANICS

Gridworks runs markets between distributed actors acting as avatars for physical devices on the grid. It needs a foundation of trustless, secure, scalable validation and authentication. It heavily uses the Algorand blockchain. If you want to undestand more about how and why this is, go here.

Despite the current negative view of crypto technology, blockchain is actually a valuable technology which we use for validating the location and metering accuracy of Transactive Devices, and for creating a scalable mechanism for auditing energy transactions.

CHAPTER

TWO

GRIDWORKS SDKS

- **gridworks**: package provides basic shared mechanics for communication and GNode structure. It is used as a package in all of our other repos.
- **gridworks-atn**: package and associated documentation for the GridWorks Python AtomicTNodes SDK. AtomicTNodes are the GridWorks actors that make electrical devices *transactive*. This SDK is a great place to learn about blockchain mechanics, as it introduces some of the simpler structures (NFTs, stateless contracts, and then some simple stateful smart contracts constructed using beaker) required for establishing the link between physical reality on the electric grid and the actors that play their avatars in GridWorks.
- **gridworks-marketmaker**: package and associated documentation for our Python MarketMaker SDK. GridWorks uses distributed actors to balance the electric grid, and MarketMakers are the actors brokering this grid balancing via the markets they run for energy and balancing services.

There are several other open source GridWorks repos to explore on our github page, including the code running on the SCADA systems that Efficiency Maine is deploying in Millinocket this winter. The GNodeFactory currently hosts the demo, and does most of the heavy lifting in terms of identity management and authentication in GridWorks. Finally, since the demo is a distributed simulation, it needs a method of handling time. That's done by a TimeCoordinator GNode.

CHAPTER

THREE

INSTALLATION

Note: gridworks requires python 3.10 or higher.

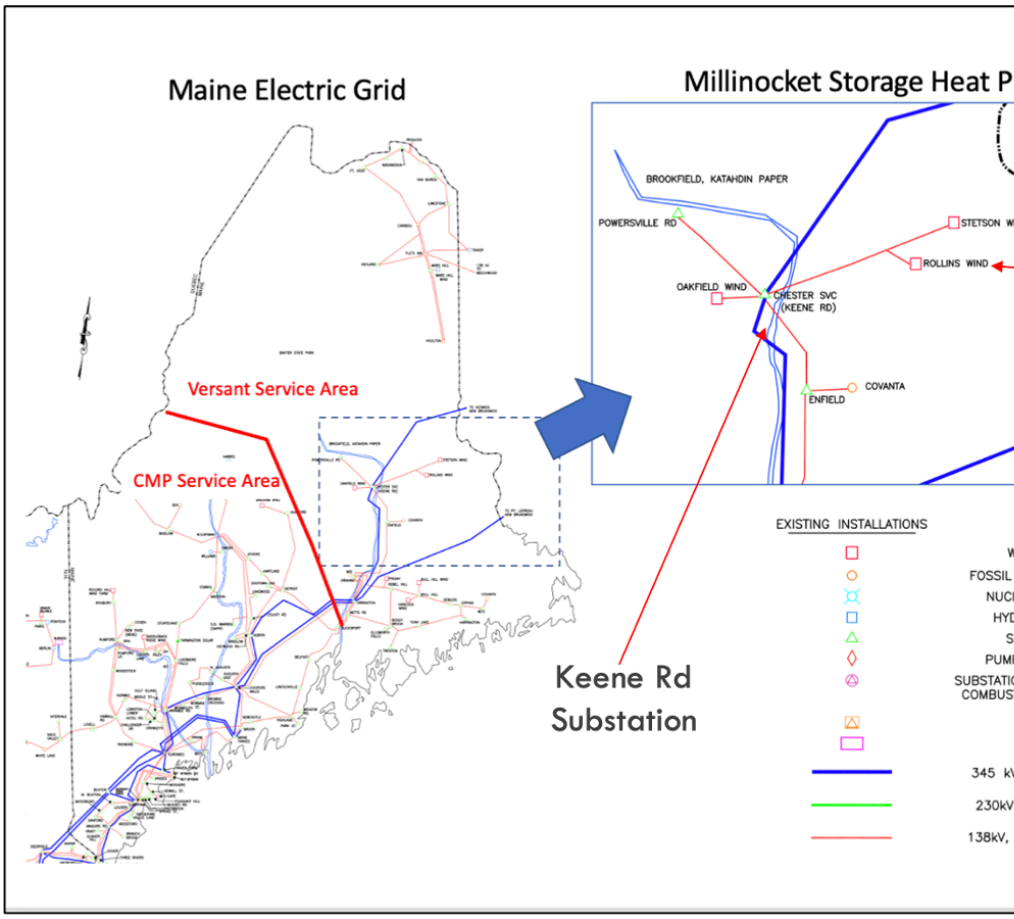
```
(venv)$ pip install gridworks
```

3.1 Millinocket Demo

This is an explanation and summary of the results of a 10 MW Transactive Heating demo, simulating 400 heat pump thermal storage Transactive Devices in Millinocket Maine.

3.1.1 Background

In the hills east of Millinocket Maine are two of the largest wind farms in New England. The electric grid is not very robust in this remote part of the state, and as a result on windy days, the grid cannot handle all of the energy being generated by the Stetson and Rollins Wind Projects.

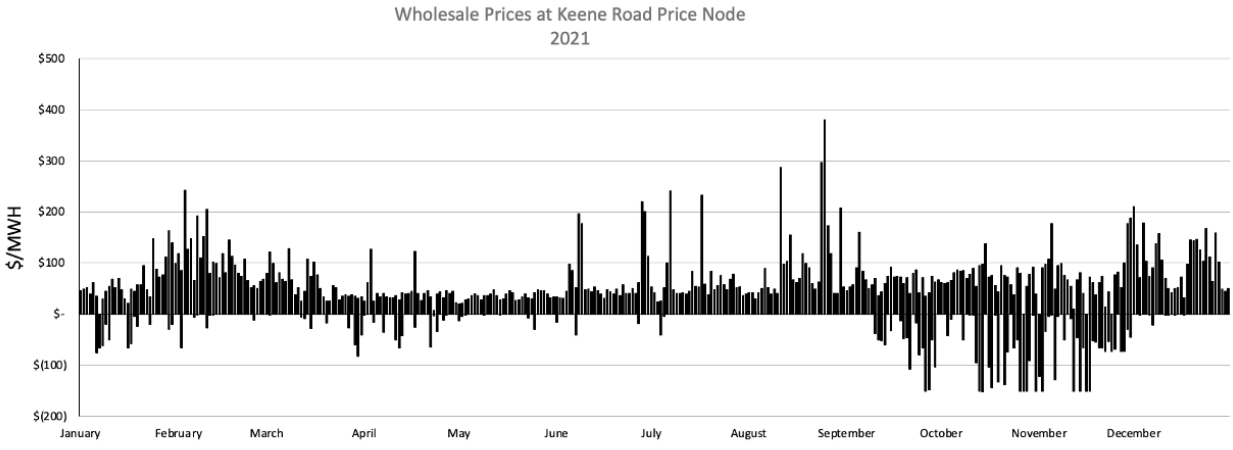


Electric Grid Around Millinocket Maine

When this happens the regional grid operator, ISO New England, sets the price of electricity very low (often -$40 or lower) in order to induce the two wind farms to “curtail” their output so as not to overload the substation at Keene Road.

Keene Road Substation

These curtailment events are not rare. In 2021, the wholesale price of electricity (called the Locational Marginal Price) at Keene Road was negative 30% of the hours during the windy winter months.



Meanwhile, people around Millinocket are shivering, turning down their thermostats to reduce their fuel oil bill.

3.1.2 The Millinocket Storage Heat Pilot

GridWorks is working with Efficient Maine Trust and other partners to replace oil boilers in homes around Millinocket with heat pumps with thermal storage. The systems use air-to-water heat pumps (located outside), with hot water tanks as thermal stores located in the basement.

Heat Pump and Storage Tank, currently being installed in Freedom ME

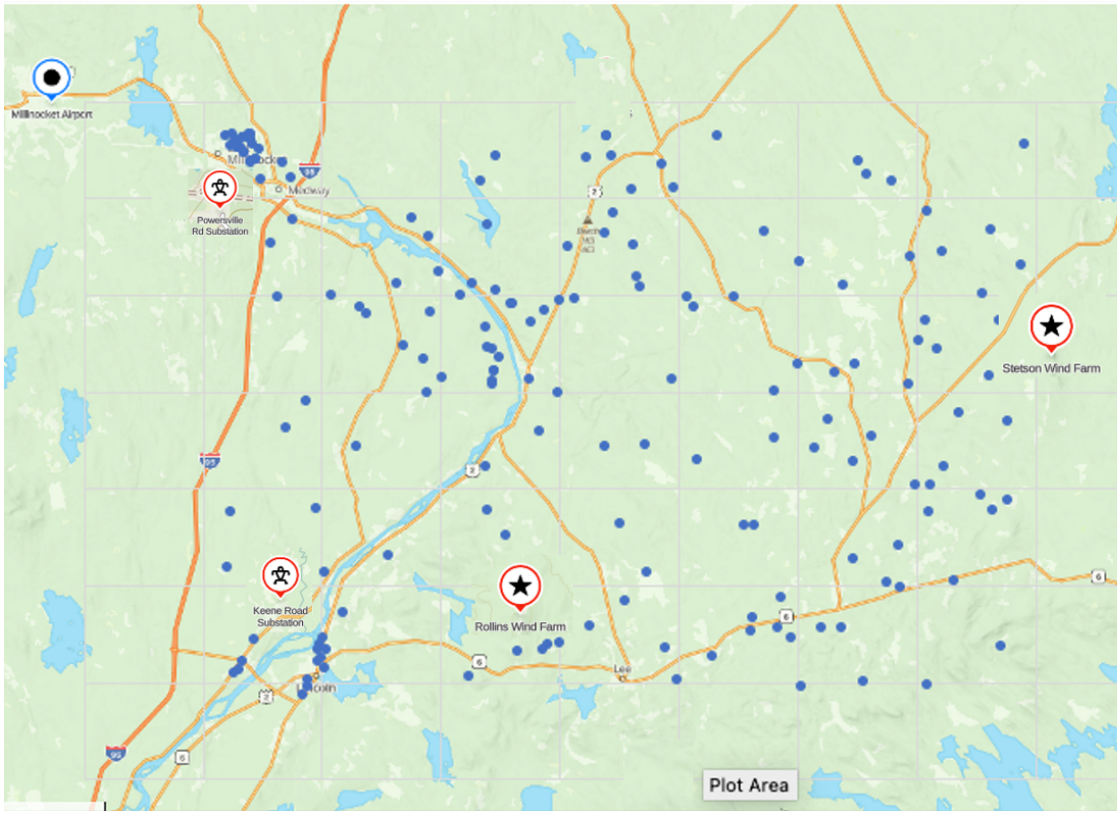
These heating systems will allow homes to heat with otherwise curtailed wind power, heating the thermal stores when electricity prices are low or negative, and then using this stored heat to keep people warm 24-7.

GridWorks is providing the sophisticated Forward-Looking Optimization software that is necessary to control these heating systems, as well as demonstrating the power of blockchain technology to help manage what will eventually be tens of thousands of decentralized, distributed energy storage devices working together to cost-effectively reduce curtailment and keep people warm at a cost substantially below the cost of oil heat.

3.1.3 A 10 MW Simulation

The simulaiton that GridWorks has delivered as part of this Algorand grant is designed to demonstrate that an aggregation of 400 homes around Millinocket could appreciably reduce the amount of curtailment from the Rollins and Stetson Wind Farms, as well as reduce the cost of heating for these homes. In addition, as will become clear in this presentation, the control problem presented by the presence of thousands of distributed load and storage assets all making decentralized decisions about when to buy energy is a near perfect application of blockchain technology.

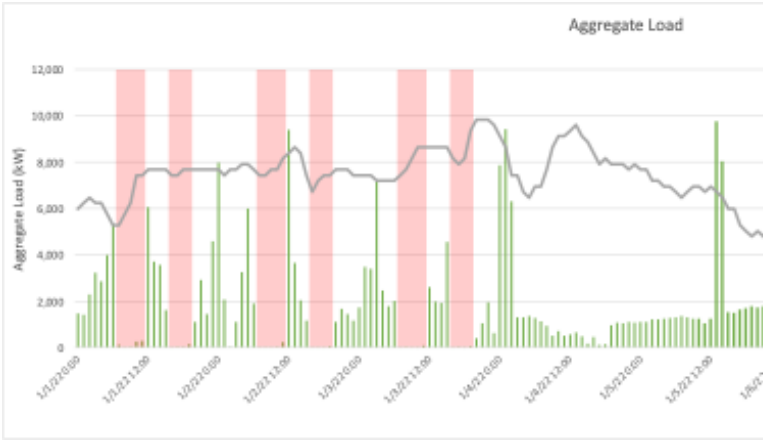
The simulation faithfully models 400 homes, of 18 different types, located in the area around the Keene Road substation.



Locations of Simulated Homes

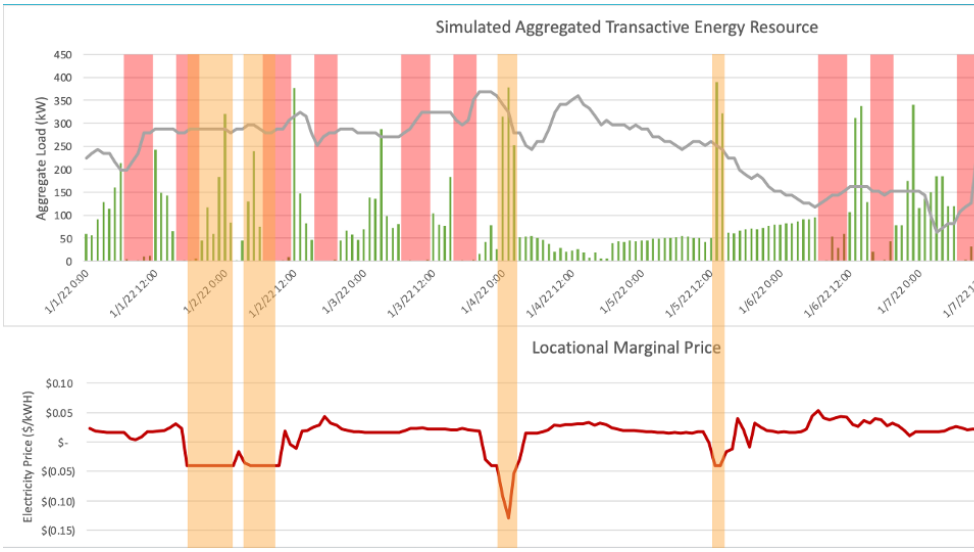
The homes differ in size, heating requirements, insulation, and the sizes of their heating systems. The average heating system draws a maximum of over 25 kW, creating an aggregate electric load of slightly above 10 MW.

The graph below shows the aggregate behavior of the homes over one week. As can be seen, the Forward-Looking Optimization algorithm keeps the heat pumps from coming on during “on-peak” periods except on the coldest days.



Performance of Aggregated Heat Pump Thermal Storage Heaters

The most interesting aspect of the performance of the aggregated load however is seeing what happens during curtailment periods at the two wind farms. Using information about the local price of electricity, we can infer when curtailment occurs. The graph below shows the curtailment periods, as well as the behavior of the aggregate load. As can be seen, the heating systems come on at nearly full power during cutailment events.



Aggregate Load During Curtailment Periods

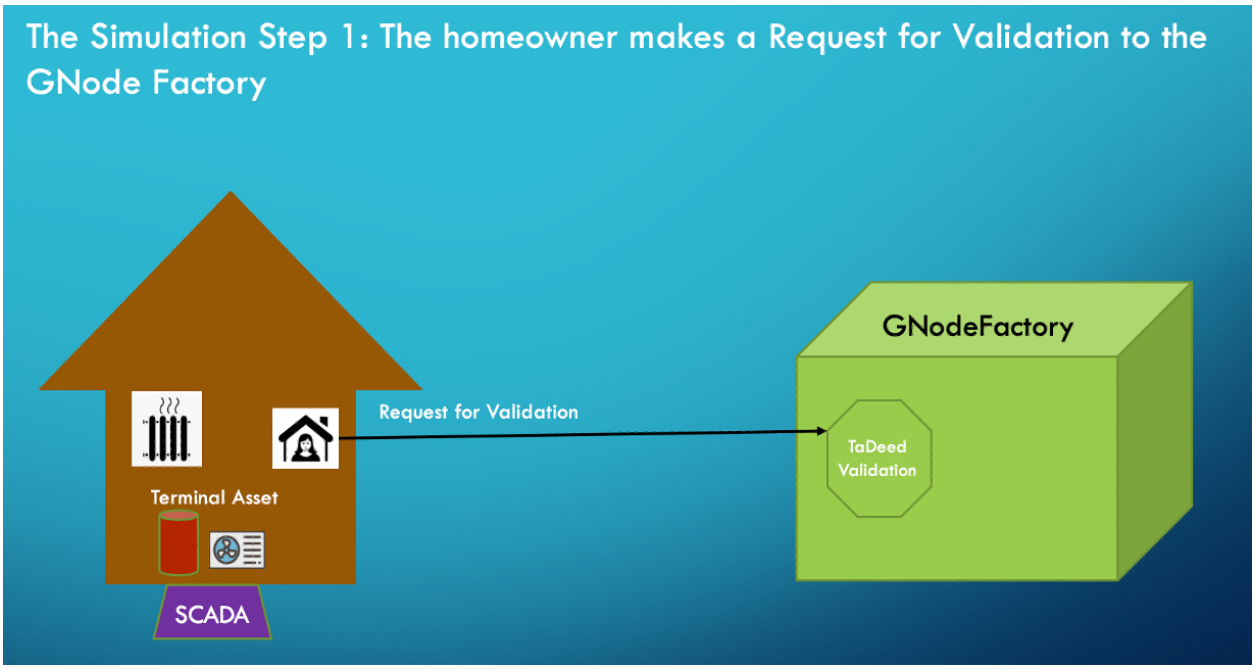
Aggregate load is nearly twice as high during curtailment periods as during the average hours in this week. ISO-New England has told us that as little at 10 MW of aggregate load located behind the Keene Road constraint would have a noticable effect on the amount of curtailment required at Stetson and Rollins Wind Farms.

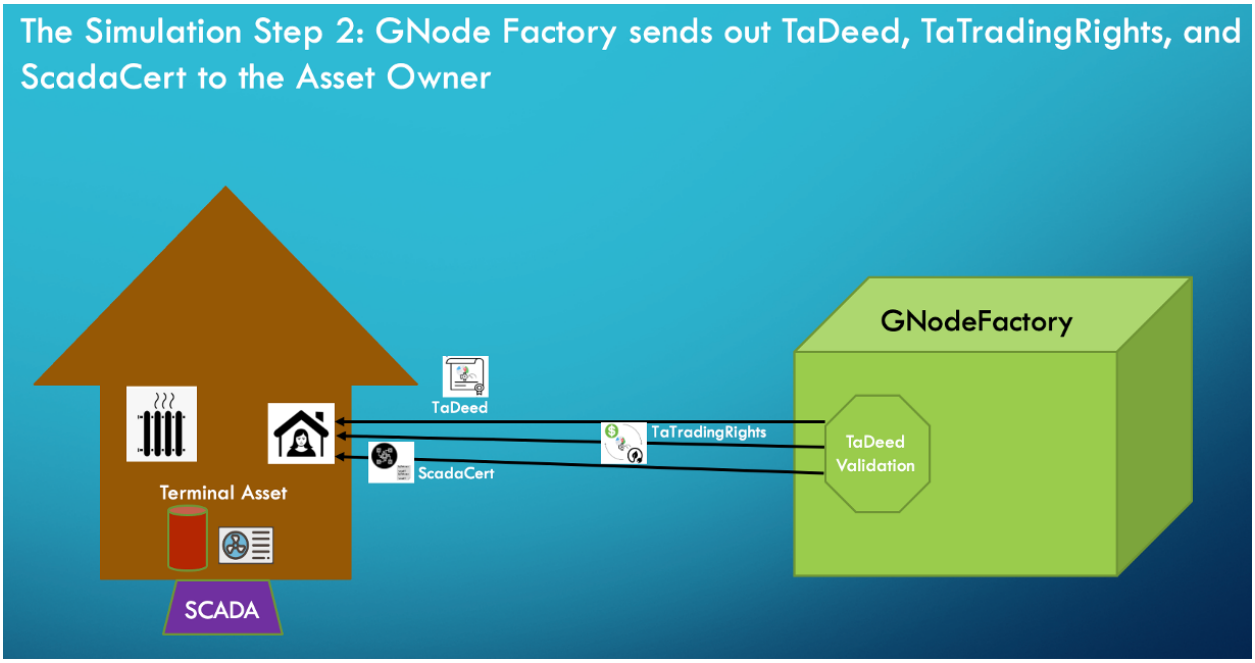
3.1.4 Why Blockchain?

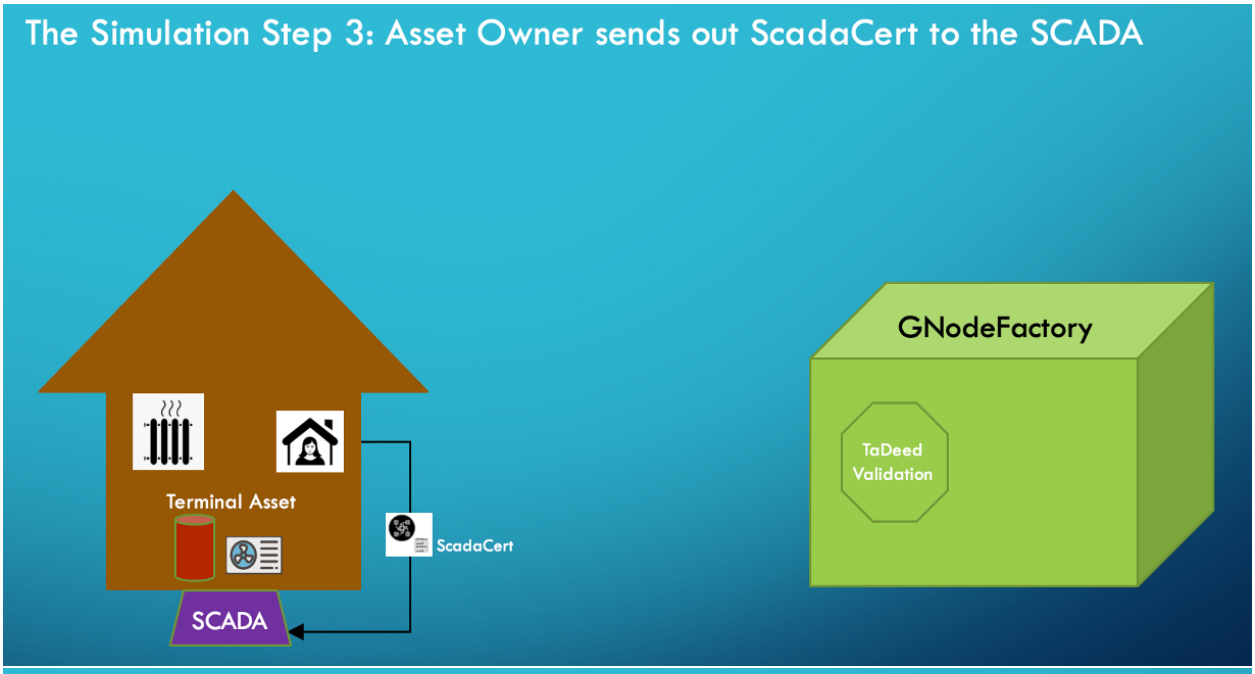
For over a century, the electric grid has been kept in balance by careful control of a relatively small number of large generators being "dispatched" to serve the load on the grid. As discussed in the *economics* and *physics* sections, load was expected to behave in a predictable way, to change relatively slowly, and to be entirely unresponsive to changes in the wholesale price of electricity.

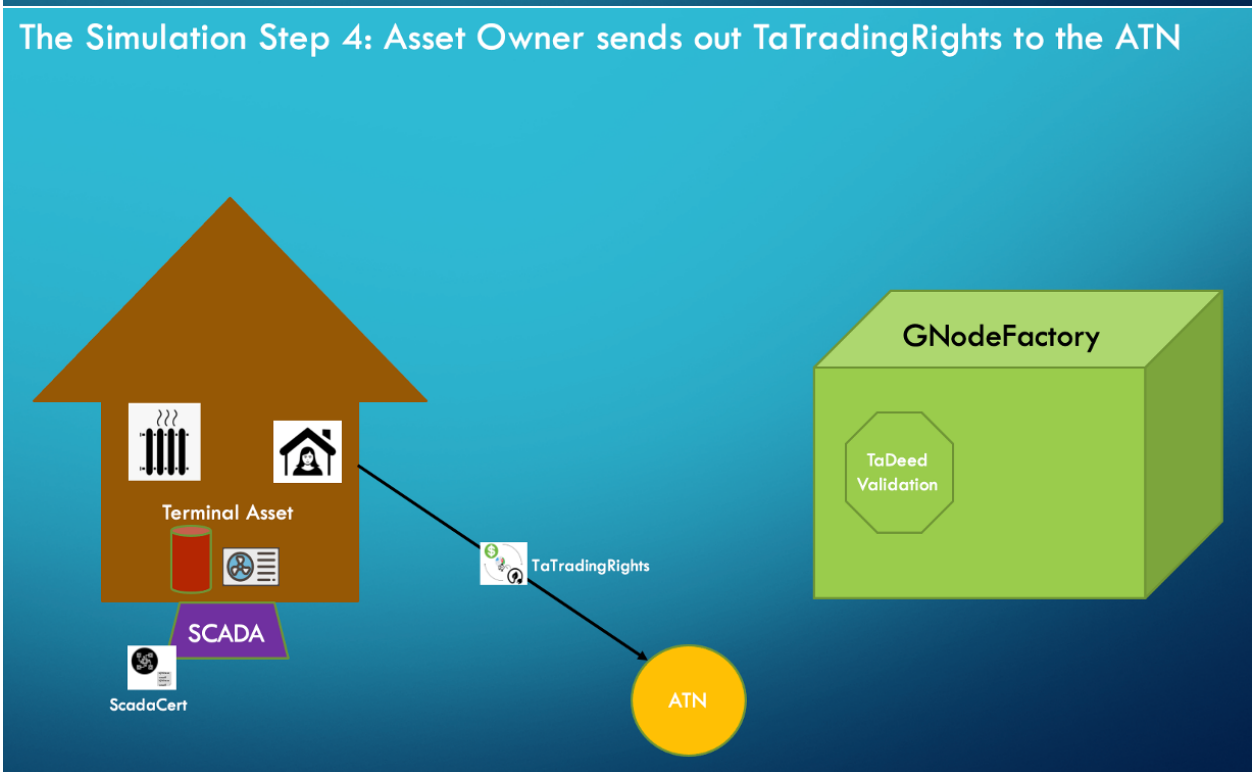
With the advent of price responsive Transactive Load, this will change. Tens of thousands – eventually millions – of geographically distributed *TerminalAssets* will be behaving in decentralized yet coordinated way, with both physical and financial consequences for energy suppliers, renewables generators, aggregators, and consumers. We believe that this is a perfect applicaton for blockchain technology.

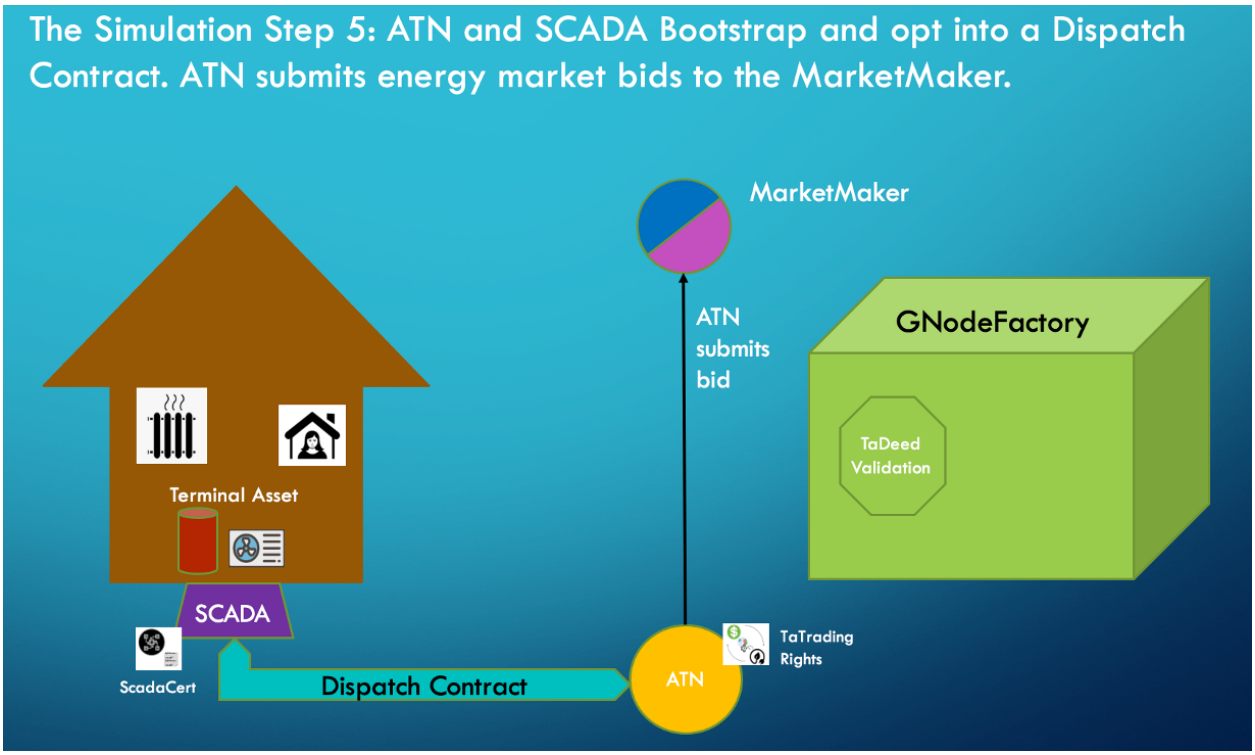
As part of this simulation, therefore, a set of actors and transactions are created that use Algorand's blockchain tools to validate and verify actions and transactions as they relate to the operation of these *Transctive Energy Resources*. This involves a 5-step process shown below:


The Simulation Step 1: The homeowner makes a Request for Validation to the GNode Factory
Request for Validation
Terminal Asset
SCADA
GNodeFactory
TaDeed Validation


The Simulation Step 2: GNode Factory sends out TaDeed, TaTradingRights, and ScadaCert to the Asset Owner
TaDeed
TaTradingRights
ScadaCert
Terminal Asset
SCADA
GNodeFactory
TaDeed Validation


The Simulation Step 3: Asset Owner sends out ScadaCert to the SCADA
Terminal Asset
SCADA
ScadaCert
GNodeFactory
TaDeed
Validation


The Simulation Step 4: Asset Owner sends out TaTradingRights to the ATN
Terminal Asset
SCADA
ScadaCert
TaTradingRights
ATN
GNodeFactory
TaDeed
Validation



Technical details about this certification process can be found in the next section, the Millinocket Tutorial.

3.2 Millinocket Demo Tutorial

First, here is a non-technical video about Millinocket that is geared towards developers that can serve as a motication and introduction to this section. Below you will find a series of code snippets and videos designed to walk you through the Millinocket Demo. It is designed to bring you up to speed on the infrastructure and domain-specific concepts embedded in GridWorks APIs, SDKs, and repos.

The tutorial is mostly sequential - you must implement earlier steps to complete later steps.

3.2.1 Demo Prep

1. You will need Python 3.10 or higher, and docker.
2. Make a demo folder and clone these 4 repositories inside it:

- gridworks-atn
- g-node-factory
- gridworks-marketmaker
- Algorand sandbox

```
- YourDemoFolder
  |
  -- gridworks-atn
  -- g-node-factory
```

(continues on next page)

(continued from previous page)

```
  -- gridworks-marketmaker
  -- sandbox
```

3. Install Python environments. At the top level of each of the 3 GridWorks repos, do this:

```
python -m venv venv
source venv/bin/activate
pip install -e .
```

4. Start the Algorand sandbox in dev mode. In the sandbox repo:

```
./sandbox up dev
```

This takes a few minutes to initiate.

5. Start the RabbitMQ broker. In the gridworks-atn repo:

```
./x86.sh  # if you have an x86 chip
./arm.sh # if you have an arm chip
```

The broker takes a few minutes to initiate. Check that it is up at the admin web page http://0.0.0.0:15672/#/, password and username both *smqPublic*

You can now either continue step by step through what is happening directly below, or skip forward to the end to run the full demo.

3.2.2 TaValidator Certification

TaValidator Certification Step 1

In order to run the simulation, TerminalAssets need to be created by the GNodeFactory. This involves a TaValidator issuing TaDeeds. So the first step in the simulation is to create a TaValidator. That is what we walk through in the first couple sections of this demo, with a simulated entity called MollyMetermaid.

Step 1 Prepare a request to send to the GNodeFactory's RestAPI.

Molly does this by creating a tavalidatorcert.algo.create JSON object, using the corresponding Python class TaValidatorcertAlgoCreate. We will walk through this step by step, introducing patterns and methods used in GridWorks packages as they come up for the first time. Do this all at the top level of the gridworks-atn repo, in a Python REPL.

Listing 1: Make a tavalidatorcert.algo.create JSON object

```
import gwatn.config as config

settings = config.ValidatorSettings()
assert settings.cert_name == 'Molly Metermaid'
assert settings.sk.get_secret_value() == 'FCLmrvflibLD6Deu3NNiUQCC9LOWpXLsbMR/
↪cP2oJzH8IT4Zu8vBAhRNsXoWF+2i6q2KyBZrPhmbDCKJD7rBBg=='
```

GridWorks uses the dotenv package and pydantic BaseSettings to handle environment variables. Default dev settings will be defined in the **src/[pckg-name]/config.py** file. In this example, ValidatorSettings (for the dev TaValidator MollyMetermaid) can be found in **gridworks-atn/src/gwatn/config.py**.

Listing 2: (cont from above)

```
import gridworks.algo_utils as algo_utils
import gwatn.config as config
from gwatn.dev_utils import DevValidator

molly = DevValidator(config.ValidatorSettings())
assert molly.acct.addr == "7QQT4GN3ZPAQEFCNWF5BMF7NULVK3CWICZVT4GM3BQRISD52YEDLWJ4MII"

multi_addr = molly.validator_multi.addr
assert multi_addr == "Y5TRQXIJHWJ4OHCZSWP4PZTCES5VWOF2KDTNYSMU5HLAUXBFQQDX6IR5KM"
assert algo_utils.algos(molly.acct.addr) > 0
assert algo_utils.algos(multi_addr) == 100
```

On initialization, the DevValidator actor funds its account using a randomly chosen genesis acct from the sandbox. After that, the DevValidator funds the 2-signature MultiAccount as described here. Examine the DevValidator init here.

AlgodClient is the algosdk wrapper for sending messages to the blockchain. GridWorks Python actors objects will typically have a **self.client** object of type AlgodClient. For example, after running the above:

Listing 3: (cont from above)

```
molly.client.account_info(multi_addr)

# RETURNS

{'address': 'Y5TRQXIJHWJ4OHCZSWP4PZTCES5VWOF2KDTNYSMU5HLAUXBFQQDX6IR5KM',
'amount': 100000000,
'amount-without-pending-rewards': 100000000,
'apps-local-state': [],
'apps-total-schema': {'num-byte-slice': 0, 'num-uint': 0},
'assets': [],
'created-apps': [],
'created-assets': [],
'min-balance': 100000,
'pending-rewards': 0,
'reward-base': 0,
'rewards': 0,
'round': 3,
'status': 'Offline',
'total-apps-opted-in': 0,
'total-assets-opted-in': 0,
'total-created-apps': 0,
'total-created-assets': 0}
```

Returning now to the objective of this session, which is having Molly create a tavalidatorcert.algo.create JSON object, using the corresponding Python class TaValidatorcertAlgoCreate.

Listing 4: Make a tavalidatorcert.algo.create JSON object

```
import gridworks.algo_utils as algo_utils
import gridworks.api_utils as api_utils
from algosdk import encoding
```

(continues on next page)

(continued from previous page)

```
from algosdk.future import transaction

import gwatn.config as config
from gwatn.dev_utils import DevValidator
from gwatn.types import TavalidatorcertAlgoCreate


molly = DevValidator(config.ValidatorSettings())

txn = transaction.AssetCreateTxn(
    sender=molly.validator_multi.address(),
    total=1,
    decimals=0,
    default_frozen=False,
    manager=molly.settings.public.gnf_admin_addr,
    asset_name=molly.settings.cert_name,
    unit_name="VLDTR",
    note=molly.settings.name,
    sp=molly.client.suggested_params(),
)

mtx = molly.validator_multi.create_mtx(txn)
mtx.sign(molly.acct.sk)

payload = TavalidatorcertAlgoCreate(
    ValidatorAddr=molly.acct.addr,
    HalfSignedCertCreationMtx=encoding.msgpack_encode(mtx),
)

print(payload.as_type())
```

In the above, Molly:

- Makes an transaction for creating an Algorand Standard Asset that meets the criterion for being a TaValidatorCert;
- Uses her 2-signature MultiAccount [GnfAdminAddr, molly.acct.addr] to make a MultiTransaction (or mtx) out of that transaction;
- Adds her signature to the mtx; and
- Finally, makes the JSON payload that she plans to send to the GNodeFactory API.

Your printed statement should look something like the example provided at the end of the tavalidatorcert.algo.create API specification.

Concrete Types Example

We use the output of the previous section to go into more detail about the mechanics of GridWorks Types.

The `TavalidatorcertAlgoCreate` Python class runs a number of validation checks. It does the basics of checking all required properties exist and have the correct Python type. It then does checks on the format of the properties and also checks additional axioms. The axioms can involve a single property (for example, checking the type of the ALgorand transaction encoded in the `HalfSignedCertCreationMtx` property) or multiple properties (for example, that the `ValidatorAddr` signed the `HalfSignedCertCreationMtx`). These formats and axioms are described in the API:

Listing 5: Formats and axioms in tavalidatorcert.algo.create

```
"formats": {
    "AlgoAddressStringFormat": {
    "type": "string",
    "description": "String of length 32, characters are all base32 digits.",
    "example": "RNMHG32VTIHTC7W3LZOEPTDGREL5IQGK46HKD3KBLZHYQUCAKLMT4G5ALI"
    },
    "AlgoMsgPackEncoded": {
    "type": "string",
    "description": "Error is not thrown with algosdk.encoding.future_msg_
↪decode(candidate)",
    "example":
↪"gqRtc2lng6ZzdWJzaWeSgaJwa8Qgi1hzb1WaDzF+215cR8xmiRfUQMrnjqHtQV5PiFBAUtmConBrxCD8IT4Zu8vBAhRNsXoWF+2i
↪mCnNHKfhkdYMCD5jxWejHRmPCrR8U9z/FBVsoCGbjDTTk2L1k7n/eVlumEk/
↪M1KSe48Jo3RocgKhdgGjdHhuiaRhcGFyhaJhbq9Nb2xseSBNZXRlcm1haWSiYXXZKWh0dHA6Ly9sb2NhbGhvc3Q6NTAwMC9tb2xse
↪iF+bI4LU6UTgG4SIxyD10PS0/
↪vNAEa93OC5SVRFn6omx2zQQ5pG5vdGXEK01vbGx5IEluYyBUZWxlbWV0cnkgU3VydmV5b3JzIGFuZCBQdXJ2ZXlvcnOjc25kxCDHZ
↪H5mIku1s4ulDm3EmU6dYKXCWEB6R0eXBlpGFjZmc="
    }
},
"axioms": {
    "Axiom1": {
    "title": "Is correct Multisig",
    "description": "Decoded HalfSignedCertCreationMtx must have type MultisigTransaction␣
↪from the 2-sig MultiAccount [GnfAdminAddr, ValidatorAddr], signed by ValidatorAddr.",
    "url": "https://gridworks.readthedocs.io/en/latest/g-node-factory.html#gnfadminaddr"
    },
    "Axiom2": {
    "title": "Is AssetConfigTxn",
    "description": "The transaction must have type AssetConfigTxn."
    },
    "Axiom3": {
    "title": "Is TaValidatorCert",
    "description": "For the asset getting created: Total is 1, Decimals is 0, UnitName␣
↪is VLDTR, Manager is GnfAdminAddr, AssetName is not blank.",
    "url": "https://gridworks.readthedocs.io/en/latest/ta-validator.html#tavalidator-
↪certificate"
    },
    "Axiom5": {
    "title": "Uniqueness",
    "description": "There must not already be a TaValidatorCert belonging to the 2-sig␣
↪[GnfAdminAddr, ValidatorAddr] address."
```

(continues on next page)

(continued from previous page)

```
    }
}
```

Perhaps the simplest way to understand this is to make a failing example. Axiom 3 states the requirements for a TaValidatorCert. Let's see what happens if we set `total=2` for the Algorand Standard Asset proposed for co-creation with the GNodeFactory (which means it would not be a Non-fungible Token).

Listing 6: (continued from above)

```
txn = transaction.AssetCreateTxn(
    sender=molly.validator_multi.address(),
    total=2,
    decimals=0,
    default_frozen=False,
    manager=molly.settings.public.gnf_admin_addr,
    # asset_name=molly.settings.cert_name,
    unit_name="VLDTR",
    note=molly.settings.name,
    sp=molly.client.suggested_params(),
)

mtx = molly.validator_multi.create_mtx(txn)
mtx.sign(molly.acct.sk)

payload = TavalidatorcertAlgoCreate(
    ValidatorAddr=molly.acct.addr,
    HalfSignedCertCreationMtx=encoding.msgpack_encode(mtx),
)
```

Listing 7: Results in ...

```
ValidationError: 1 validation error for TavalidatorcertAlgoCreate
HalfSignedCertCreationMtx
Axiom 3: TaValidatorCert ASA Total must be 1,  got 2. See https://gridworks.readthedocs.
↪io/en/latest/ta-validator.html#tavalidator-certificate (type=value_error)
```

TaValidator Certification Step 2

In this section, MollyMetermaid sends the payload she built in Step 1 to the GNodeFactory API. Start up the GNodeFactory by going to the **g-node-factory** repo (top level) and running `./uvi.sh`. This uses uvicorn to start the GNodeFactory and its RestAPI. Check http://0.0.0.0:8000/settings/ to see that it is working.

Start with this codeblock from section 1 above, and then add the following:

Listing 8: Post tavalidatorcert.algo.create JSON object to GNodeFactory RestAPI

```
import requests
api_endpoint = (
        f"{molly.settings.public.gnf_api_root}/tavalidatorcert-algo-create/"
    )
r = requests.post(url=api_endpoint, json=payload.as_dict())
```

(continues on next page)

(continued from previous page)

```
print(r.json())
```

Listing 9: Results in ...

```
{
    'Note': 'ValidatorCert for ..WJ4MII created, asset_idx 4 \n tx_id␣
↪CIJ6VVCNY5OK5INJ2443ZRZAUISBHW756BFMFFVSE7V5MHY6J7LQ',
    'HttpStatusCode': 200,
    'PayloadTypeName': 'int',
    'PayloadAsDict': {'Value': 4}
}
```

If the posted json is not valid, the GNodeFactory will return a 422 response with details about the error. Under the hood, this is made easy by the fact that all GridWorks APIs are built with FastAPI, and both FastAPI and the GridWorks Type SDKs use pydantic. We test this out using the encoded Multitransaction from above for making an ASA with total=2.

Listing 10: Posting an invalid type to the GNodeFactory API

```
d = {
"ValidatorAddr": "7QQT4GN3ZPAQEFCNWF5BMF7NULVK3CWICZVT4GM3BQRISD52YEDLWJ4MII",
"HalfSignedCertCreationMtx":
↪"gqRtc2lng6ZzdWJzaWeSgaJwa8Qgi1hzb1WaDzF+215cR8xmiRfUQMrnjqHtQV5PiFBAUtmConBrxCD8IT4Zu8vBAhRNsXoWF+2i
↪ca4yuXu+sP2nK2u+dh2OZEKeB6mpps91gyM6UxiJSFDCwPKrGTv0tbEiT64zrEHZgeQ+MJsBeJiUw0ng2L8Io3RocgKhdgGjdHhui
↪",
"TypeName": "tavalidatorcert.algo.create",
"Version": "000"
}
r = requests.post(url=api_endpoint, json=d)

print(r.json())
```

Listing 11: Results in ...

```
{
    'detail': [{'loc': ['body', 'HalfSignedCertCreationMtx'],
    'msg': 'Axiom 3: TaValidatorCert ASA Total must be 1,  got 2. See https://gridworks.
↪readthedocs.io/en/latest/ta-validator.html#tavalidator-certificate',
    'type': 'value_error'}]
}
```

TaValidator Certification Step 3

What the GNodeFactory did after validating the type was:

- Sign the mtx so it now has its requried set of 2 signatures;
- Submit the mtx to the blockchain using gridworks.algo_utils send_signed_mtx,
- Return the Id of the newly created TaValidatorCertificate in the Post response (*Note that ids for Algorand Standard Assets are integers.*)

If you want to inspect the created asset using algosdk, do this:

Listing 12: Inspect created asset

```
molly.client.account_info(molly.validator_multi.addr)
```

Molly has two more steps before she is a TaValidator. The first is opting into her certificate:

Listing 13: Molly opts into her TaValidatorCert

```
opt_in_txn = transaction.AssetOptInTxn(
        sender=molly.acct.addr,
        index=4,
        sp=molly.client.suggested_params(),
)
signed_txn = opt_in_txn.sign(molly.acct.sk)
molly.client.send_transaction(signed_txn)

response = algo_utils.wait_for_transaction(molly.client, signed_txn.get_txid())
print(response)
```

You will need to substitute `index=4` for the ASA Id returned by your GNodeFactory API.

(*Why does she need to opt in? Because Account Algo requirements increase with the number of ASAs they hold.*)

Molly completes the process by generating another POST to the GNodeFactory API, this time for co-signing a *transfer* of the TaValidatorCert from the 2-signature MultiAccount that created the TaValidatorCert (and currently owns it) to her Algorand address. This process is almost identical to Step 1 above, creating a tavalidatorcert.algo.transfer JSON object, using the corresponding Python class TaValidatorcertAlgoTransfer, and posting it to the api endpoint `0.0.0.0:8000/tavalidatorcert-algo-transfer`.

One thing to note about the validation of these blockchain-related GridWorks types is the depend on the state of the Blockchain and thus on in particular on *time() Go to gridworks-atn/src/gwatn/dev_utils/dev_validator.py to see the above implemented in one place.

3.2.3 TaDeed and TaTradingRights

The TaDeeds are interesting in two ways: they are made by a Multi-Account tying local and global authority together, and in the future they can *either* be Algorand Standard Assets or Smart Signatures. What is more interesting than the technicalitiies around the TaDeed and the TaTradingRights is their ramificactions on the rest of GridWorks. You can start reading about that here and by generally looking through the lexicon as well as the sections on physics, , economics ,, and Transactive Energy.

3.2.4 GridWorks Contracts On-Chain

The big work of bringing the GridWorks Contracts On-Chain was actually preparing the ground with enough certificates for the Smart Contracts to have a foundation and connection to the real, physical grid and all the transcactive electric devices connected to it.

3.2.5 The Dispatch Contract

Establishing Communication

This tutorial discusses the graphics from the end of the Millinocket Demo section, honing in on the relationship between two central actors in GridWorks: the AtomicTNode and the Scada. In a nutshell, the AtomicTNode tells the SCADA when to flip relays to turn the power on and off for the underlying Transactive Device that is sensed and controlled by the SCADA. Saving money and making sure the Transactive Device is honoring its service level agreement (keeping people warm, in the case of an electric thermal storage heater) is the responsibility of the AtomicTNode. **Except**, of course, that the SCADA must take over responsibility for controlling the device if the Internet goes down.

What this means is that there are responsibility (money, comfort) that are *shared through time* between the AtomicTNode and the SCADA, where at any point in time the responsibility belongs to exactly one of the two. Determining which of these two is responsible at any point in time hinges on the state of communicating between these two actors. The primary purpose of the Dispatch Contract, therefore, is to serve as a third-party objective umpire about when these two actors are **Talking With** each other. The AtomicTNode and SCADA send heartbeats to each other as RabbitMQ messages. Each time one sends a heartbeat to its partner, it also sends a record of the heartbeat to their Dispatch Contract. The Dispatch Contract then stores that record in a box (BOX STORAGE IS STILL GETTING DEBUGGED)

The Dispatch Contract is also (1) involved in the initial handshake between the SCADA and the AtomicTNode, and (2) will be used for auditing energy and power transactions.

A note on time stamps. When the AtomicTNode or SCADA sends a record of a heartbeat they just sent, they include the timestamp of sending. Even though the Dispatch Contract is arbitrating the state of a variable (TalkingWith) through time, it does not rely on blockchain time in any way. This is a requirement, since blockchain time can vary. It also works because the state of TalkingWith does not need to be determined in the present moment but instead is only required for auditing purposes. This means eventual consistency is good enough.

Beaker and the Dispatch Contract

GridWorks Type APIs and our work connecting ABIs with SDKs in order to accurately articulate and formalize the boundaries between distributed systems (with both semantics and syntax) dovetails very significantly with the work that Algorand has done with ABIs and additional layers of abstraction. I would like to highlight beaker for how it supports the evolution of abstraction on the Algorand, with a particular shout-out to its Fight Club boxes example and its nicely commented and written AMM demo .

In this video I go over how the Dispatch Contract works by walking through code. The sample code is in this directory of the **gridworks-atn** package. Follow set-up instructions at the beginning of this section.

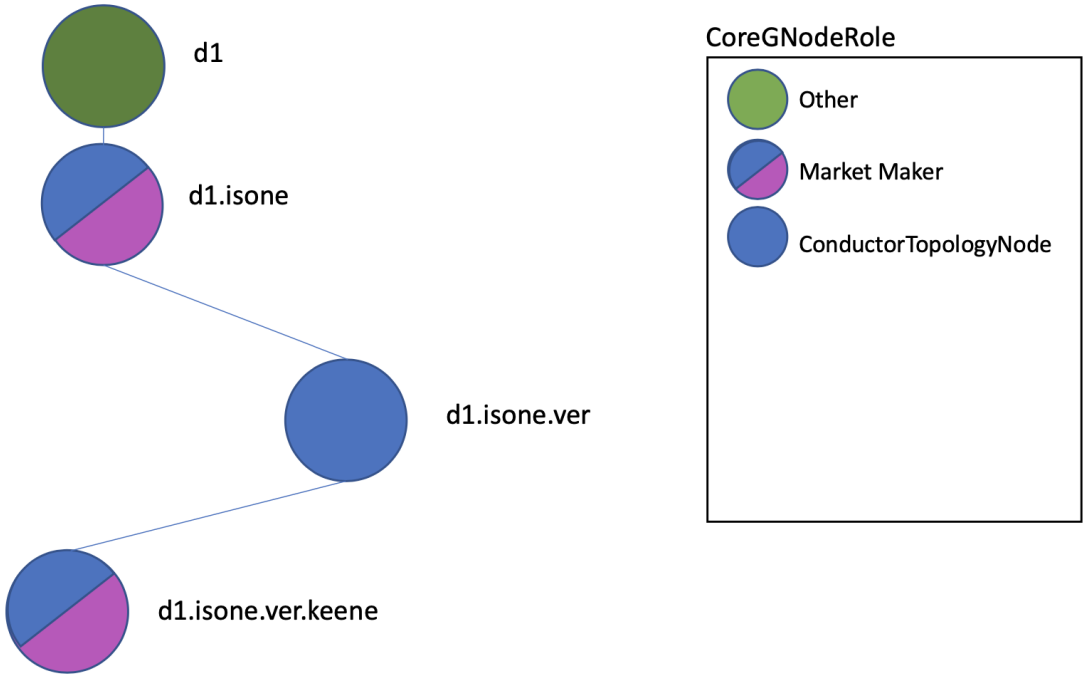
3.2.6 Running the Demo

The demo requires a fair amount of preparation. We are working on simplifying this. Right now this is the dispatch contract demo. Will be adding the full simulation.

1. Start with the instructions from demo prep above.
2. In the **g-node-factory** repo, run:

```
python millinocket.py
```

This will reset the Algorand sandbox, flush the GNodeFactory database, and then start up the GNodeFactory API with the following GNodes pre-loaded:



(Go here to read more about the Core GNode Roles)

Check for success

- Check http://0.0.0.0:8000/base-g-nodes/ to confirm these 4 BaseGNodes are loaded.
- **Check http://0.0.0.0:15672/#/queues and look for a queue named d1-Fxxx to confirm a GNodeFactory queue showed up on your local rabbit broker. If that queue is not showing up:**
  - do a *git pull* to update the default rabbit_url for GnfSettings in config to point to localhost
  - If you have a *.env* file overriding default values in GnfSettings, look for a GNF_RABBIT__URL that is not pointing to localhost.

3. In **gridworks-atn**, set up the TaValidator web page

Listing 14: Set up the TA Validator API

```
./validator.sh
```

Check for success

- Check http://localhost:8001/docs for Molly Metermaid's (TA Validator) FastAPI

4. Also in **gridworks-atn**, in another window, run the Dispatch Contract demo

Listing 15: Dispatch contract demo

```
python millinocket.py
```

3.2.7 Full Simulation

Flavor 1: 1 minute time steps In the original milestone 2, there were no simulated SCADA actors. Obviously, to add the Dispatch Contracts between Atns and SCADA to the simulation, the SCADA must be added to the simulation. In addition, the time steps must be on the same order as the heartbeats back and forth between the Atn and the SCADA.

Both of these additions slow down the simulation significantly. This simulation is useful in helping to evaluating the real life performance of the platform. However, to see what happens with the demand curves, the original flavor should be used

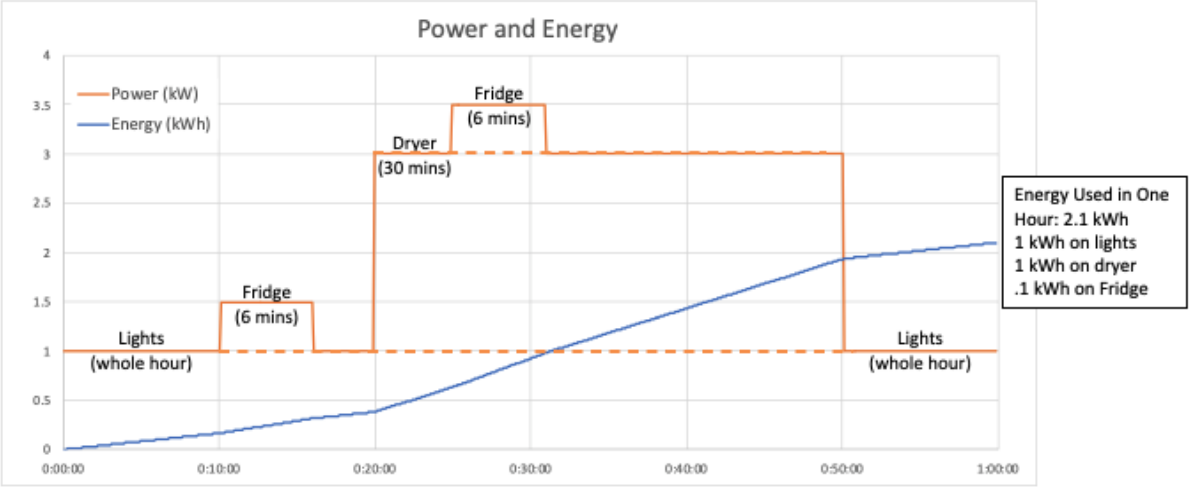
Flavor 2: 1 hour time steps (no SCADA actors, one hour time steps)

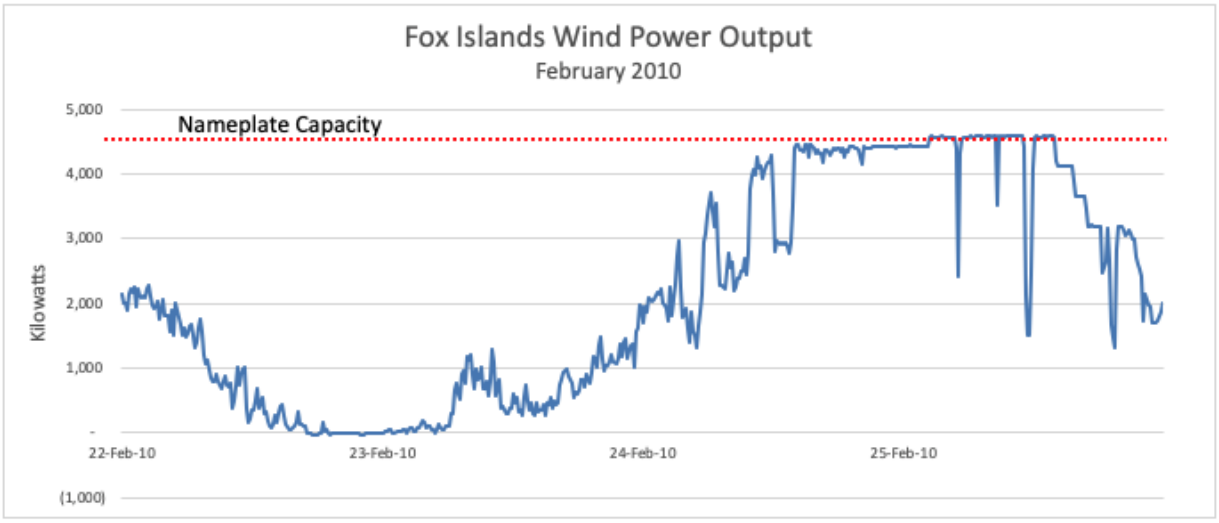
3.3 Physics of the Electric Grid

3.3.1 Power and Energy

- **Power** is the instantaneous rate of flow of energy
  - Units: Watts, Horsepower, BTUs/hour, Ergs/sec, etc.
  - How powerful is your car? How large is this powerplant?
- **Energy** is power through time
  - Units: Watt-hours, Calories, BTUs, Joules, Foot-pounds, Barrels of Oil Equivalent (BOE), etc.
  - How much energy does it take to drive to New York? How much energy does this powerplant produce in a year?

- Power (Watts) vs. Energy (Watt-Hours)
- Energy = Power through time





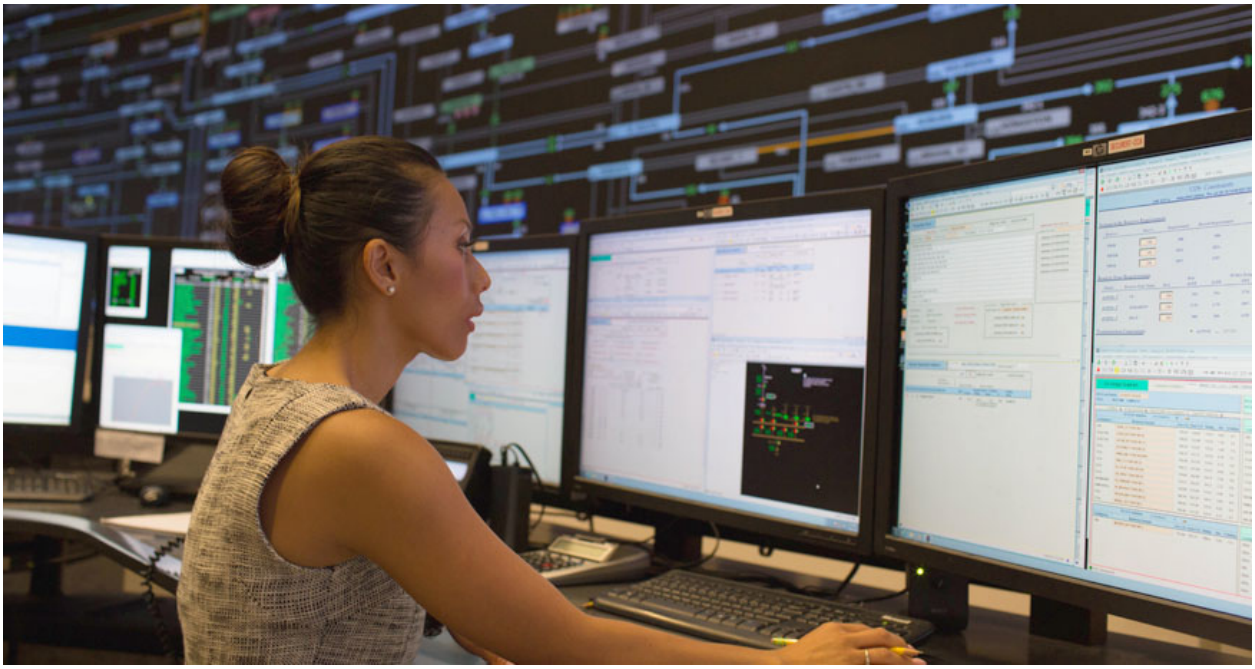
How much energy was produced over these four days?

3.3.2 What does "grid balancing" mean?

- The grid must be "balanced"
- This means that the instantaneous power of all electric *loads* connected to the system must exactly equal the instantaneous electric power of all *generators*.
- This also means, of course, that the amount of *energy consumed* over any period of time must be equal to the amount of *energy generated* over that time.
  - This is only true if we label the electricity put into storage (e.g. into a battery, an EV, or a hot water heater) as "consumption" and the electricity taken out of storage as "generation."

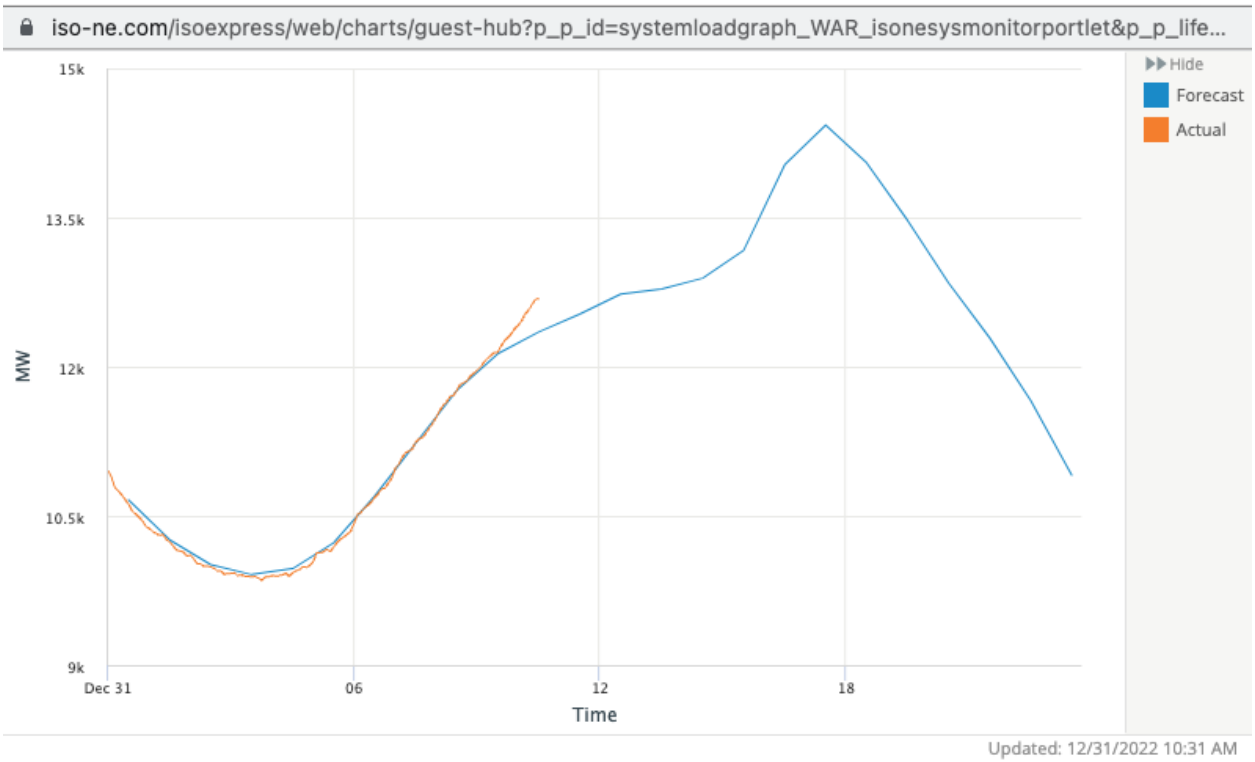
How is the grid kept in balance?

On a traditional electric grid (that is, one without large amounts of intermittent renewable generation) the control rooms of grid operators "dispatch" generators – that is instruct them to turn on and off, and to modulate their power up and down – so that the amount of electricity generated exactly matches the amount of electricity being consumed: in every second, every minute, every hour, every day, etc.



Grid Operators have elaborate models that help them forecast load hours and even days in advance, and they plan how they will meet this load with precisely the correct amount of generation.

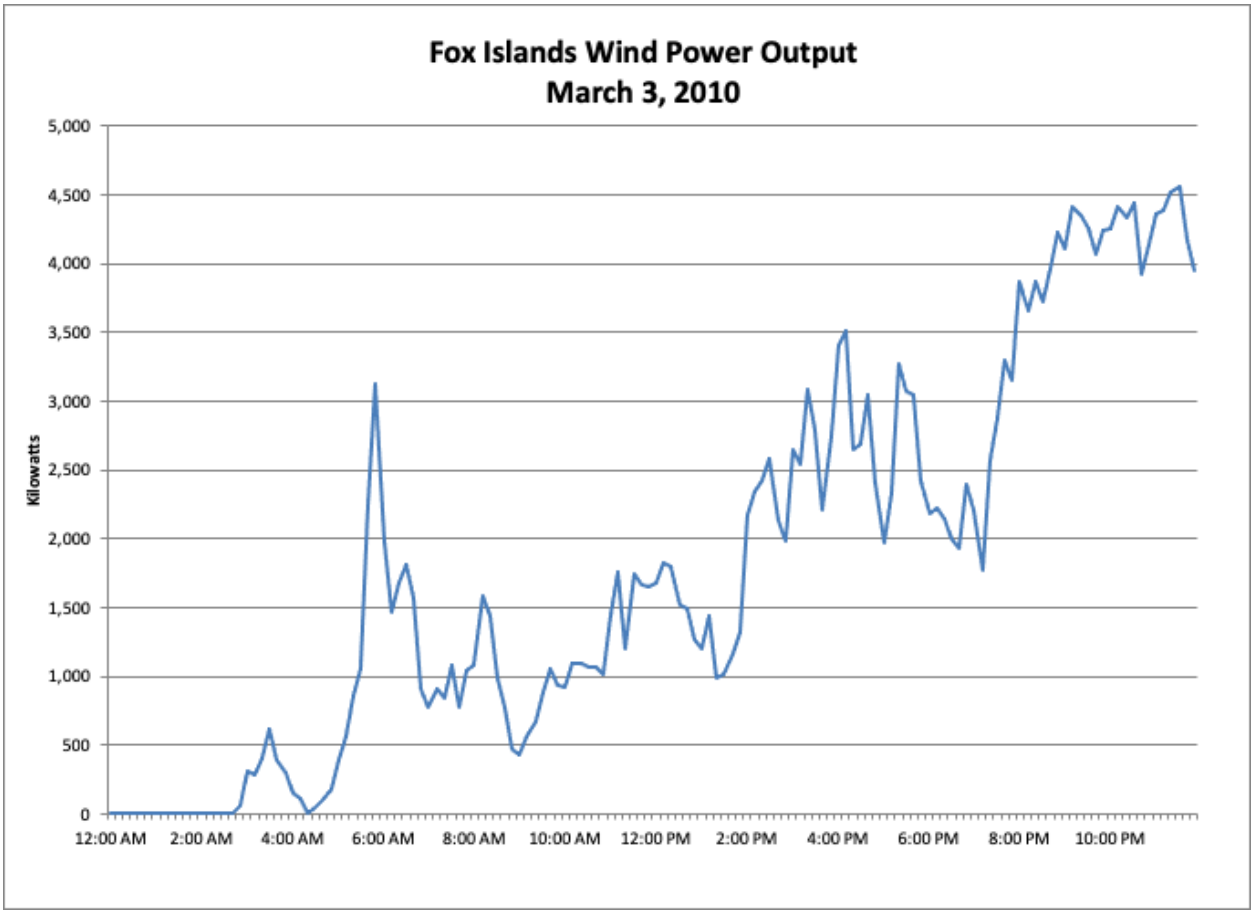
They then react, in real time, when load is not exactly what they planned for, by dispatching more or less generation on a second-by-second basis.



The difficulty of this grid balancing task is exacerbated by the fact that there are several complex feedback loops in grid control that make the system inherently unstable. As Christopher DeMarco, Gridworks Energy Consulting principal and professor of Electrical Engineering at the University of Wisconsin and one of the country's leading experts on the stability of synchronous electric grids says, "Balancing the grid is a bit like balancing a broom on its handle."

This process of balancing the grid becomes significantly more complex and difficult when there is a large amount of intermittent renewable generation happening. A wind or solar farm cannot be "turned up" when the electric load increases, and turning down a generator whose output is free seems like an enormous waste of energy.

In addition, neither wind nor solar generate a steady level of power, the way that fossil fuel plants do. When the sun goes behind a cloud, the output of a solar farm can be cut by 80% in seconds. On gusty days, the output of wind farms may fluctuate wildly.



Most experts agree that the only way to keep a grid with a high fraction of intermittent renewable generation balanced is to make the load side of the grid more flexible and responsive: what we call Transactive Energy Management.

3.4 Economics of the Electric Grid

3.4.1 Supply and Demand

Load on the electic grid is an overloaded word: it can mean devices that consume electricity, it can mean instantaneous power withdrawn from the grid, and it can mean energy withdrawn from the grid over a period of time.

In this section, we are going to focus on the economics of the electric grid and of energy markets, and redefine generation as "supply" and load as "demand," in the classic economic sense of supply and demand in a market.

This is a felicitous definition, because just like in the physics of the grid where generation must equal load, in economics supply must equal demand.

Prior to the deregulation of electricity markets in the US in the 1980s, large electric utilities were vertically integrated, owning the entire grid from generators (and sometimes even coal mines) to the electric meter attached to the outside of your house. These utilities kept the grid in balance by dispatching their own generators so that generation = load.

With deregulation, which required the electric utilities to divest themselves of their generation assets, new entities had to be created to keep the grid in balance. These entities – variously called Independent System Operators (ISOs), Regional Transmission Organizations (RTOs) or simply Grid Operators – make sure that generation = load using different mechanisms depending on the timescale over which the grid balancing needs to occur. On all timescales at or above 5

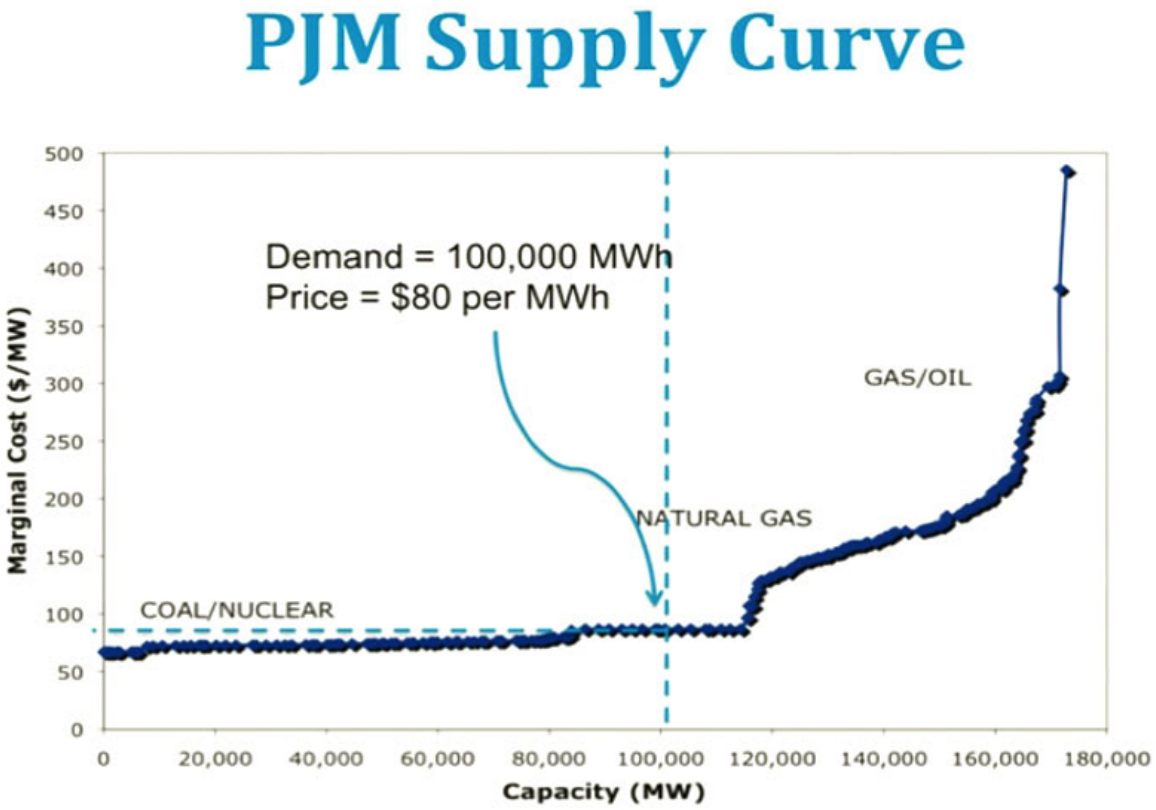
minutes, these Grid Operators run wholesale markets that insure that generation = load by making sure that supply = demand.

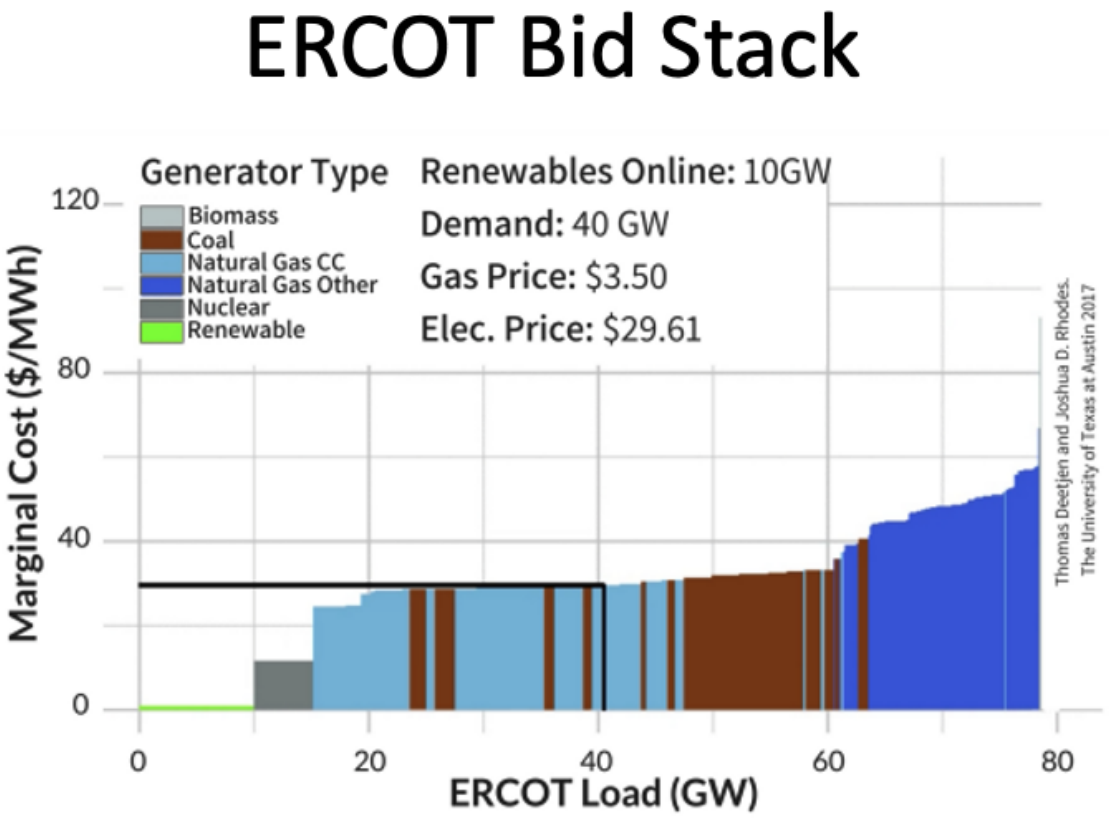
Generation and existing markets

The way that Grid Operators run these wholesale markets is as follows. Each Generator submits bids to the Grid Operator in advance committing to deliver a certain quantity of generation at any given price. This might be a simple as "If the price goes above $75 per MWh, I will deliver my full output of 150 MW. Below this price I will generate nothing" or it might be more complex: "At any price below $50 per MWh, I will generate nothing; between $50 and $75, I will generate 100 MW, above $75 I will generate my full output of 150 MW."

In general the bids that Generators submit depend on what type of fuel they use, the price of this fuel, and the type of plant that the Generator operates. For reasons that will be explained below, Generators have incentives to make bids that reflect their true marginal cost of generation.

The Grid Operator gets all of these bids and sorts them by price, and then "stacks them up" to create a "Bid Stack" or aggregate supply curve. These supply curves look like this:





This Bid Stack represents the relationship between the price of electricity set by the Grid Operator and aggregate amount of energy that Generators will be willing to supply in any hour. Using this bid stack, and using its forecast for Load in each hour, the Grid Operator can set a price for electricity in each hour that will exactly match Supply with Demand, or Generation with Load.

The result is a time-varying price for electricity that is higher when demand is high, and lower when demand is low. This is because in hours with high demand, grid operators need to set a higher price in order to induce higher-cost generators to come online.

- It is important to to note that all generators get paid the price that is paid to the generator with the highest bid accepted into the market, regardless of their own bid.
- It is for this reason that bidders will bid based on their true marginal costs: their bid is only used to determine whether they get accepted into the market, not how much they get paid.

Load and existing markets

Note that nowhere in this story is there any notion that the amount of energy demanded in an hour – that is, the amount of Load – might depend on the price of electricity. Virtually every grid in the world assumes that demand for electricity is entirely "inelastic."

This assumption is largely correct: few electricity consumers pay the time-varying cost for electricity. Rather this volatility is "averaged out" by their Energy Supplier, allowing them not to worry about what electricity might cost this hour, and robbing them of the opportunity, which they have with almost everything else that they buy, to economize on electricity use when it is expensive, and use more of it when it is cheap.

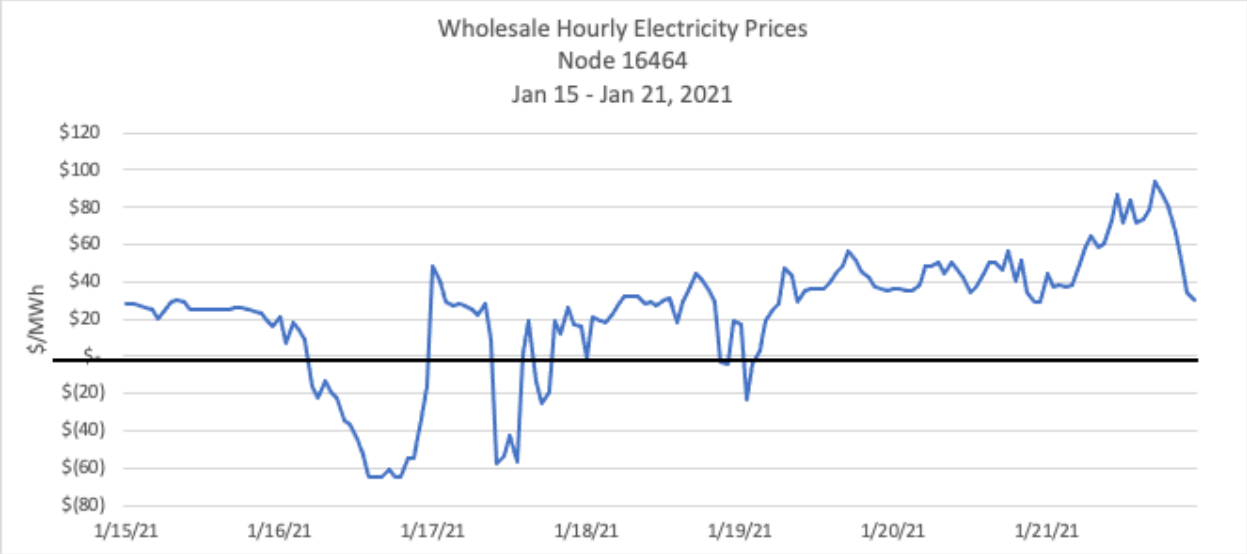
3.4.2 Zero marginal cost generation

What Happens When the Marginal Cost of Generation is Zero? How will a Wind Generator, whose marginal cost of generation is zero, bid into energy markets?

The simple answer is, they will agree to turn on their wind turbines whenever the offered electricity price is greater than zero. Same with a solar farm.

The actual answer is not quite this simple. Because renewable energy is subsidized by state and federal governments in the US (and many other places in the world) on a per kWh basis regardless of how much the energy is worth on the grid, renewables generators have an incentive to keep on generating even if the price of electricity paid by the grid operator is zero. In fact, in order for the grid operator to get a renewables generator to turn off their plants, they must charge the generators (that is set a negative price for electricity) an amount larger than the size of the government subsidy.

The result of this is that, in places where there is a lot of renewable generation wholesale, prices for electricity routinely fall below zero when renewable generation is high. The graph below shows hourly wholesale electricity prices for one week at a location east of Millinocket Maine, near two of the largest wind farms in New England.



Those hours on the 16th, 17th, and 19th of January when prices go negative are periods when wind energy is being shut down – curtailed – by the New England Grid Operator. The amount of locally-generated wind energy exceeds the load during those hours, and the wind farms need to be curtailed to protect equipment from being overloaded.

Note that, as long as load on the electric grid is entirely price inelastic, there will be no response to this cheap, abundant, and wasted renewable energy.

3.5 Transactive Energy and the Grid

3.5.1 Transactive Energy Resources

We start with definition from the *lexicon*:

A **Transactive Energy Resource** (or TER) is a physical resource capable of 24-7, real-time, geographically localized response to grid conditions that can significantly shift and adapt its pattern of electric power use and/or generation with negligible negative consequences on its primary use.

Aggregations of Transactive Load have the potential to significantly mitigate the problems that are encountered on an electric grid with significant amounts of renewable generation.

- They do this by fundamentally altering the behavior of electric load on the grid, turning it from a fixed burden that must be served by some generator into a powerful and flexible resource that can be used to balance the grid on timescales varying from days to milliseconds.

What is needed?

Several things need to change for load to become Transactive and begin to solve the problem of renewables integration on the world's electric grids.

- The current idea of Demand Response, which involves reducing load during discrete "peak" events, needs to be redefined to mean 24/7 adaptation and adjustment to conditions on the grid.
- Load assets need to be able to communicate in real time with the Grid Operator, weather service, and other sources of information required to operate efficiently and effectively.
- Forward-looking optimization techniques will be required to ensure that load assets can meet the needs of the grid while also fulfilling their primary function.
- A mechanism for distributed, decentralized, secure, and trustless transactions must be developed so that potentially millions of Transactive Energy Resources can interact with each other, the Grid Operator, and end users. This will include the development of new Service Level Agreements between TER Aggregators and end users.

By `trustless`, what we mean is that counterparties do not need to rely on some central third party to broker their contract. We use this word in the context of already having established the rights of a GNode to act as an avatar for a physical device. This is established using a trust mechanism that bridges the gap between evaluating a real power meter and the blockchain. For more information on this, read about why we call a TaDeed a Link of Trust.

3.5.2 Service Level Agreements

What is a Service Level Agreement (SLA), and why is it important for transactive energy?

Your current SLA with your electricity supplier is simple: whenever you turn on a switch, or plug in your EV, or the thermostat in your fridge tells the compressor to turn on, or the temperature in your hot water tank goes below some setpoint, you can consume electricity at full power until you turn off the switch, your car is full, your fridge is cold, or your water is hot.

Even though there may be significant flexibility in when your car needs to charge, or your fridge needs to come on, or when you next need hot water, under the current SLA your energy supplier has no flexibility in when it draws power from the grid and delivers it to you.

For traditional Demand Response, you get paid by an Aggregator for occasionally inconveniencing you. This works because it is occasional. But with transactive energy, the grid interaction is happening 24-7: *all* decisions about when to consume electricity are made in the context if it is the cheapest time to do it.

But cheapest compared to what?

From the perspective of the developers writing code for the AtomicTNode, the Service Level Agreement contains a *contract* defining the operating bounds of the AtomicTNode's bidding strategy.

From the perspective of the person paying the bills, the Service Level Agreement articulates what they can expect in terms of the performance of their device.

The new Service Level Agreements for TERs will allow end users to "sell" the inherent flexibility in many electric devices and appliances to the grid in return for lower energy bills. An SLA for an electric clothes dryer might be something like: when you push the start button, your TER Aggregator guarantees that your clothes will be dry by 7:00 the next morning. If you need them dried right now, you push an override button and for 50¢, your dryer will come on immediately. In return for agreeing to this SLA, you will receive a 25% rebate on all energy consumed by your dryer. A similar SLA for your hot water heater might be that you are guaranteed a full tank at 5:00 AM and 5:00 PM every day; once again if you need to reheat your tank right away, you push an override button and for a small fee you get immediate power.

The thing to recognize is that rarely do consumers care about electric power per se; they care about the services that electrical devices provide: dry clothes; hot water; cold food; transportation when I need it, etc. On the other hand, what matters to the grid is the precise timing of when power is drawn. By allowing Transactive Energy Resources to separate the consumption of electricity from the consumption of the services that electricity provides, these new Service Level Agreements will allow load to become Transactive.

- Note that not all electric loads are good candidates for becoming TERs. Lighting, for instance, is something that you need when you need it and not when you don't. Nothing but cheap electric storage (a rechargeable battery) will turn lighting into a Transactive Load.

3.5.3 Transactive Heat

There are many electric loads that are attractive potential TERs, but we believe that the most promising is space heating (both commercial and residential) using heat pumps with thermal storage.

These systems can be designed so that they can deliver enormous flexibility to the grid without sacrificing end-user comfort. The SLA for Transactife space heat looks pretty much like it looks now: the aggregator guarantees that the heating system will hit the thermostat's setpoint 24/7.

This is possible through the use of thermal storage, either in the form of hot water tanks in the basement, or Phase-Change Materials (which allow for the storage of large amounts of thermal energy in a much smaller volume).

The thermal storage in these systems allow them to buy energy from the grid when it is plentiful and cheap, and keep the building warm for up to 12 hours (on most days) without buying any more energy.

- This type of system requires sophisticated Forward-Looking Optimization techniques, using weather and price forecasts to plan future energy purchases and react in real time to unexpected weather and/or market conditions.

The *Transactive Heat Pilot in Maine* will demonstrate the ability of the GridWorks TER platform to keep houses warm, reduce curtailment of wind farms, and lower the cost of home heating for thousands of people in and around Millinocket.

3.5.4 The Millocket Demo

3.5.5 In the field

Efficiency Maine is a quasi-governmental organization with a mandate to install 100,000 heat pumps across the state. They are concerned that they do not have a solution that is cost effective and that can keep up with the coldest days. Therefore, they have funded a pilot project to test out the capabilities of transactive thermal storage heat systems. Over the next year, this will likely be on the scale of 20 homes. If the solution proves to keep people warm and is cost effective, they are poised to roll out rapidly at scale after that.

Freedom ME

To support this real demonstration, GridWorks has developed open-source SCADA code (based on our previous experience at VCharge, where we designed and deploying SCADA systems as retrofits for thousands of ceramic brick thermal stoage heaters). We have also agreed to build prototype SCADA boxes (built with off-the-shelf parts, with a raspberry Pi 4 as the brain) for the first several dozen installations.

Dec 30 2022 The first SCADA is deployed in Freedom Maine! The thermal storage and heat pump have also been installed, waiting on some upgrades to the heat distribution system before going live.

3.6 The Algorand Blockchain

Gridworks is building a platform enabling distributed, decentralized, secure, and trustless transactions so that potentially millions of Transactive Energy Resources can interact with each other, the Grid Operator, and end users.

By `trustless`, what we mean is that counterparties do not need to rely on some central third party to broker their contract. In fact, Gridworks foundations for a scalable, decentralized trustless market structure is based in establishing *links of trust*: blockchain ASAs and/or SmartSigs establishing the rights of a GNode Actor to act as an avatar for a physical device. For example, a) (and its relationship to a TaValidatorCert) bridges the gap between a third-party inspector evaluating the accuracy of an electricity meter at a house, and the *TerminalAsset GNode* avatar of the *Transactive Device* behind that meter. For more information on this, read about why we call a TaDeed a Link of Trust.

Working knowledge of some basic Algorand tech is necessary to start developing with GridWorks:

- py-algorand-sdk, the Algorand Python SDK
- Algorand Standard Assets
- Algorand Smart Signatures

These are the tools used for establishing the link between GridWorks and the real world. For example, in order to use an AtomicTNode it a larger GridWorks simulation (including anytime you are outside of the *dev environment*), the AtomicTNode's GNode actor will need to own *TaTradingRights* for its associated TerminalAsset. These foundational GridWorks certificates and their creation/transfer process are interlinked and have an initiatic sequencing. For example, TaValidatorCert -> TaDeed -> TaTradingRights: (The TaValidatorCert for a TaValidator account must be created prior to the creation of a TaDeed, which must be created prior to the creation of TaTradingRights.) The first couple sections of the *Millinocket tutorial* walk through this process.

If you have never worked with blockchain before, it may take some time to get used to the blockchain mechanics that show up in the first part of the tutorial. But once you get the hang of it, they become simple to understand and use, and you can realize that these certificates are actually just the welcome mat into the house.

Although these initial certificates are simple, they are also the single most essential part of blockchain used by GridWOrks. Without something like the above for establishing proof of *when*, *where* and *how much* a device uses, Transactive resources do not work at scale.

Once this trust is established, one can operate within the world of code: a system of distributed actors, their blockchain holdings and/or identities, the rules governing how these actors and blockchain entities communicate, and the data they create. This is what we typically mean by `GridWorks`. Even before using blockchain, GridWorks was built on a system of contracts. The initial, most important contract is the *DispatchContract* between an AtomicTNode and a SCADA both serving the same TerminalAsset. Note that this contract truly is an agreement between two code entities. In the *Millinocket demo*, the DispatchContract is monitoring and establishing the state of comms between two actors working together to keep a house warm, where one of these actors lives behind a residential router in a rural area of Maine. Built on top of the DispatchContract is the *RepresentationContract*, a multi-party contract used to establish the provision of a service to a human or business customer (warmth as a service, in Millinocket) as well as the financial terms going along with this. Unlike the DispatchContract, the RepresentationContract will involve *both* code actors (AtomicTNode, MarketMaker) *and* humans/human businesses (HomeOwner, Aggregator, Grid Operator).

DispatchContract and the RepresentationContract you will need to learn about Algorand Smart Contracts, written in Pyteal using Beaker. This is the technology used in GridWorks for operating simple foundational contracts between GNodes and the entities (human or otherwise) that own or use them:

- TaDaemon App
- DispatchContract
- RepresentationContract

In addition, GridWorks is working on:

- **A Smart Contract implementation of MarketMaker GNodes** These second-generation MarketMakers will behave somewhat like the blockchain Automated Market Makers, or AMMs (see this demo Algo example in beaker) to solve the Optimal Power Flow problems traditionally solved in centralized ways by Grid Operators to determine market clearing prices. The catch is that, unlike the liquidity pools of AMMs, this tree-like hierarchy of GNode MarketMakers must coordinate with each other, since they are solving a problem that is globally connected by the laws of physics (in particular, Kirchoff's laws).
- **A Smart Contract implementation of the GNodeFactory** The GNodeFactory is the Single Source of Truth in GridWorks for the known topology of the electric grid. At present, this information is stored in a database belonging to a centralized application. At scale, this topology and some the data used to back it up needs to be on-chain.

3.6.1 Creating a TaDeed

3.7 Hello World

Note: All GridWorks repos assume Python 3.10 or higher, docker, and the Algorand sandbox. Also, For these `hello world` examples, please clone this repo in order to spin up a local rabbit broker.

GridWorks is message-driven, following the reactive manifesto. The basic building blocks are GNodes ('G' stands for the electric grid). GNode actors communicate via:

1) Asynchronous message-passing on a RabbitMQ Broker;

2) API calls via FastAPI interfaces; and

3) Calls to the Algorand blockchain ledger, typically usually smart contract ABI method calls.

3.7.1 Hello Rabbit

Here is a simple example of the first. Before running, start a development world rabbit broker. From the top level of this repo:

1) **./arm.sh** (*if your computer has an arm chip*)

2) **./x86.sh** (*x86 chip*)

Wait for the rabbit admin page to load (username/passwd smqPublic)

```
from gridworks.actor_base import ActorBase
from gridworks.enums import GNodeRole
from gridworks.enums import MessageCategory
from gridworks.gw_config import GNodeSettings
from gridworks.types import HeartbeatA_Maker


class HelloGNode(ActorBase):
    def __init__(self, settings: GNodeSettings):
        super().__init__(settings=settings)
        self.settings: GNodeSettings = settings

    def prepare_for_death(self) -> None:
        self.actor_main_stopped = True


def demo():
    settings = GNodeSettings()

    settings.g_node_alias = "d1.hello"
    settings.g_node_role_value = "GNode"

    gn = HelloGNode(settings=settings)
    gn.start()

    input(
        f"Go to http://0.0.0.0:15672/#/queues and wait for the d1.hello-Fxxxx queue to␣
↪appear."
    )
    assert gn.g_node_role == GNodeRole.GNode
    hb = HeartbeatA_Maker(my_hex=0, your_last_hex="a").tuple

    print("Broadcasting a heartbeat on rabbitmq")
    gn.send_message(payload=hb, message_category=MessageCategory.RabbitJsonBroadcast)

    print("Inpsect the dummy ear queue to examine the message (click on GetMessage)")
    input("http://0.0.0.0:15672/#/queues/d1__1/dummy_ear_q")
    input(f"Hit return to tear down the GNode rabbit actor")
    gn.stop()
```

(continues on next page)

(continued from previous page)

```
if __name__ == "__main__":
    demo()
```

3.7.2 Hello Algorand

Before you start, clone the Algorand sandbox and start it up in dev mode by running *./sandbox up dev* at its top level directory (you will need docker installed).

In order for an Algorand Account or Smart Contract to do any actions on-chain, it must be funded. The dev sandbox is running a local dev Algorand blockchain on your computer, which comes with a couple pre-funded genesis accounts. The **gridworks** package has a method called dev_fund_to_min which can be called in the dev environment to fund an account from one of the sandbox genesis accounts.

Listing 16: Create and fund a dev account

```
import gridworks.algo_utils as algo_utils
from gridworks.algo_utils import BasicAccount
import gridworks.dev_utils.algo_setup as algo_setup

acct = BasicAccount()
assert algo_utils.algos(acct.addr) == 0
algo_setup.dev_fund_to_min(addr=acct.addr, min_algos=3)
assert algo_utils.algos(acct.addr) == 3
algo_setup.dev_fund_to_min(addr=acct.addr, min_algos=2)
assert algo_utils.algos(acct.addr) == 3
```

The first Algorand transaction that occurs in any full simulation is the creation of a TaValidator Certificate. (For quick context, a TaValidator is an entity involved in establishing the link between GridWorks avatars for real-world devices attached to the electric grid.)

This certificate is an example of an Algorand Non-fungible Token (NFT). In particular, the certificate is an Algorand Standard Asset whose total is 1 (this is how uniqueness in enforced). You may have heard about NFTs in the context of art. We are using NFTs in a similar way - s an identifiable object that can only be owned by a single entity.

All ASAs have *creators* - identified by the Algorand address that pays the fee for the creation transaction (aka the sender). One of the criterion for an ASA being a TaValidator Certificate is that the creator's address must be a 2-signature MultiSig address (examine all of the criteria here, and also note that this is enforced in the validation of a tavalidator.algo.create type, in Axiom 3).

Gridworks has a MultiAccount object used for this purpose:

Listing 17: Creating a 2-sig MultiAccount[GnfAdminAddr, ValidatorAddr]

```
from gridworks.gw_config import Public
from gridworks.algo_utils import BasicAccount
from gridworks.algo_utils import MultisigAccount

validator_acct = BasicAccount()
gnf_admin_addr = Public().gnf_admin_addr
multi = MultisigAccount(
```

(continues on next page)

(continued from previous page)

```
    version=1,
    threshold=2,
    addresses = [gnf_admin_addr, validator_acct.addr]
)

print(multi.addr)
```

GridWorks always uses its MultiAccount instead of the **algosdk.futures.transaction.Multisig** object. The algosdk Multisig object is not designed for multiple transactions, as it stores transaction signatures. Gotcha note. Sometimes Gridworks methods duck-type BasicAccount and MultiAccount. This only works if the method only accesses their public address (e.g. **acct.addr**). MultiAccounts do not have a secret key, since it does not store the private information of their signatories.

To continue with more tutorial-type instructions, please go to the Millinocket tutorial.

3.8 APIs,SDKs, and ABIs

The foundational GridWorks package gridworks, which is the source of this documentation, is designed to help you set up a sample GridWorks API and its corresponding SDK in a development environment, to help learn the ropes for core GridWorks communication mechanics.

Imagine a GridWorks application (say, a GNode actor) as a cell in a larger organism. It has ways of communicating, and this is captured in its cell membrane.

- **APIs** The Application Programming Interface specifies how communication happens at and beyond its cell membranes.
- **SDKs** A GridWorks Software Development Kit, on the other hand, is like an internal organelle capable of generating valid messages to send out to other cells.
- **ABIs** The application might be an Algorand Smart Contract - for example, a [DispatchContract](dispatch-contract). Then it has an Algorand Application Binary Interface instead of an API.

More on Algorand ABIs

We describe the historical precedent for Application Binary Interfaces. Continuing the cell analogy, an ABI is a specification for a cell communicating _inwardly_ to, say, its mitochondria. ABIs are familiar to c programmers as a way of arranging memory layout.

The Algorand blockchain is like a smallish, Turing-complete computer whose guts are spread out across the Internet, whose every internal action in compiler code is visible for the world to see. A method call to an Algorand Smart Contract is much more like a C ABI than a Restful API with a url endpoint.

For more information on Algorand ABIs:

- A 40 minute youtube discussing the launch of Algorand ABIs
- ARC-4, the source of truth for implementation details
- pyteal ABI support docs
- AlgoBank ABI example

3.8.1 GridWorks APIs

In order to explore how GridWorks APIs work, please start up a version of the `gridworks` package API:

` uvicorn gridworks.rest_api:app --reload --port 8000 `

navigate to

` http://127.0.0.1:8001/docs# `

You can then try out POSTS to various API endpoints. These endpoints will return a valid response if the GridWorks type provided has no errors, and will otherwise return a list of one or more errors.

GridWorks types are the building blocks for GridWorks APIs: they articulate *what* is getting sent without specifying *where and how* (i.e., a Post to a Restful API endpoint, or a message published to an exchange on a rabbit broker).

As a first pass, you can make the slightly incorrect assumption that all Gridworks types must be serialized JSON - the lingua franca of APIs. Note that JSON is NOT a programming language. It is a multi-dimensional graph structure, and a greatest common denominator for articulating the WHAT of data that programming languages use and manipulate in separate but interacting applications. For example, a JSON "int"does not articulate anything about memory - it is just an arbitrarily long sequence of *0123456789* characters.

The Type SDK objects provide a Pythonic method of creating valid instances of these types, and for interpretting payloads as natural Python objects.

Examples of sending valid payloads to GridWorks APIS

COMING SOON

3.9 Code Generation

We use the aicapture open source command line package manager to translate information we store in airtable into automatically generated code.

Basic flow is:

- Add new data to the airtable tables for the
- run `aic -build` from `CodeHomeDir/CodeGenerationTools/`

3.9.1

TODO: Update with information about how to update table structure via the effortless API

The ssot.me site has documentation with a list of transpilers available for the ssotme tool. For example, I added an odxml-to-entities transpiler by:

1. looking at the syntax on that site
2. running the command `ssotme odxml-to-entities-json -i ODXML/DataSchema.odxml -o SSoT/Entities.json -install`
3. moving the additional block in aicapture.json to just above the `airtable-to-xml` transpiler (which requires Entities.json to be updated)

3.9.2 More about types

When working with the GridWorks ecosystem, a "type" means a set of validations for a serialized message getting passed between software agents/applications within the ecosystem. The authority for these validations is the Gridworks Type registry (not yet built). Typically these messages are JSON but the registry does allow for different serializations.

Some of these validations are simple and syntactical in nature - for example a JSON attribute "Description" may have type "string." Others capture embedded semantic meaning. This can involve axioms that involve multiple attributes within the type. It also involves the use of GridWorks enums across multiple types whose meanings are conveyed both in how they are used across multiple software actors and also in their corresponding documentation. One way to think about this is that Domain Driven Design requires a mechanism for maintaining evolution and clarity about the *meaning* of messages, not just their syntax. The place where this starts is in a mechanism for managing enums.

The GridWorks registry manages authority for three types of objects, in increasing order of complexity:

- **PropertyFormats** Examples:

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

```
GwVersion (defn): A three-character string, where each character is a numeric digit, and
the first character must not be 0.
```

- **Enums**
  - Each enum is defined by a `LeftRightDot` formatted Name and a `GwVersion` formatted Version. The combination of these components is represented as VersionedName f"{name}.{version}".
  - Once an element belongs to an Enum version, it must belong to all future versions. For example if the enum `rochambeau.throw.000` includes "Rock" then "Rock" must belong to `rochambeau.throw.001` as well.
  - Each enum also has a default value. If an inbound message has an unrecognized enum value, the SDK will interpret that enum value as the default.
  - An Enum's elements are represented as 8-digit hex strings, known as `EnumSymbols`, when transmitted as serial messages. In addition, each `EnumSymbol` is associated with a `LocalValue`, which is a human-readable string. The purpose of `LocalValues` is to convey the meaning of the enum element and is intended to be interpreted by software agents utilizing native enums. For instance, "Rock" might serve as the `LocalValue`, while its corresponding `EnumSymbol` could be "1ea112b9". For a more comprehensive explanation, please refer to the documentation provided below [TODO: ADD LINK]
- **Types**
  - As with enums, each type is defined by a `LeftRightDot` formatted TypeName and a `GwVersion` formatted Version. The combination of these components is represented as VersionedName f"{TypeName}.{Version}".

This registry will eventually have an open-source tool command line tool (like ssotme itself) which will take the place of the xslt tools in `CodeGenerationTools/GridworksCore that can be accessed by any repository using GridWorks Types in order to build or start using the GridWorks types. But until that happens there is near-replication in the CodeGenerationTools/GridworksCore folders.

All instances of GridWorks types must include a TypeName which is recognized by the GridWorks Type Registry. The format of the TypeName is LeftRightDot

Every type comes in versions. These versions are strings of three numerals and increment in numerical order. Starting with "000", then "001" etc.

If you ask query the Gridworks Type Registry for information about a TypeName, it will either tell you there is no such TypeName or it will return information about how to evaluate whether a serial message is an example of that type

(message validation). . This is specifically intended for situations where bandwidth is limited and the messages have pretty minimal content.

The message validation includes:

1. attribute-specific validations 1a. type (string, integer, boolean, float, another gridworks type) 1b. format (various simple formats like left-right-dot format, or a GridWorks Enum) 1c. additional simple pydantic checks (greater than 0)
2. more complex validations called axioms that are expected to be hand-coded. Some of these involve multiple attributes.

3.9.3 New type built from airtable

3.9.4 New type built from scratch

This is instructions for if you want to build a type with minimal interaction with the code derivation machinery. It assumes you are adding a new type with TypeName `your.type.name` to the `gridworks-protocol` repository

1. In the airtable Types table add a new row:

TypeName	CurrentVersion	Status	ProtocolCategory
your.type.name	000	Active	Json

```
[ProtocolTypes](https://airtable.com/appgibWM6WZW20bBx/tblnyHLmbF5ihZoM1/
↪viwyweGPAEhKuNpBB?blocks=hide)
```

2. In the VersionedTypes table add:

TypeName	Version	PreStatus
your.type.name	000	Active

3. In the ProtocolTypes table add:

Protocol	VersionedType
gwproto	your.type.name.000

4. From the cli in `gridworks-protocol/CodeGenerationTools/GridworksCore`:

```
aic -build
```

Note: aic and ssotme are both aliases for the aicapture command

5. From `gridworks-protocol`, in poetry, run `nox -s pre-commit` a few times.

3.9.5 Why make the EnumSymbols unreadable?

3.10 GridWorks lexicon

3.10.1 AggregatedTNode

An **AggregatedTNode** ("T" for Transactive) is an aggregation of *AtomicTNodes*.

The individual AtomicTNodes within the AggregatedTNode must share the same *MarketMaker GNode*. The MarketMaker for an AtomicTNode is the first MarketMaker encountered when walking up the GNodeTree branch from the AtomicTNode - that is, the youngest MarketMaker ancestor of the AtomicTNode).

An AtomicTNode cannot participate in a market run by the MarketMaker if it is part of an AggregatedTNode that is also participating in that market. That is, it can't buy (or sell) energy twice.

When an AggregatedTNode takes over the market participation, it must handle the dispatch of its AtomicTNodes out of the market.

AggregatedTNodes may be created by businesses that prefer to have a coarser-grained bidding strategy (although they will still be responsible for providing the granular, metered consumption data for settlement).

Another reason to create an AggregatedTNode is if a MarketMaker has a minimum size requirement for market participation. While the default MarketMakers provided by GridWorks will not have this restriction, the design of the system allows for it.

Back to Lexicon

3.10.2 AtomicMeteringNode

AtomicMeteringNode is a helper *GNodeRole* and *CoreGNodeRole*. Essentially, it is the larval form of an *AtomicTNode*.

This GNode for a pending `AtomicTNode` must *exist* at the time of creation of its `TerminalAsset`, since it is the *parent* of the `TerminalAsset`. However, it cannot become an `AtomicTNode` until it owns the *TaTradingRights* for the `TerminalAsset`.

Back to Lexicon

3.10.3 AtomicTNode

At a high level, an **AtomicTNode** is the GNode responsible for the behavior of a *TerminalAsset* most of the time. It meets the primary needs of the `TerminalAsset` (like providing heat, if the asset is a heating system), and it looks at future price and weather forecasts in order to optimize the participation of the TerminalAsset in electricity markets.

The AtomicTNode has a good internal model of how the TerminalAsset works, adjusting and double-checking this getting real-time data from the SCADA.

The owner of the TerminalAsset (TaOwner) chooses the AtomicTNode by entering into a multi-party Representation Contract.

The AtomicTNode has a close working relationship with the SCADA. In a nutshell, whenever the two are successfully communicating, the SCADA lets the AtomicTNode call all the shots on how the TerminalAsset uses (or provides) electrical power.

AtomicTNode is a fundamental *GNodeRole* and *CoreGNodeRole*.

Back to Lexicon

3.10.4 Conductor Topology Node

The interior nodes of the copper spanning tree are places where multiple lines come together. Often, there will also be some sort of voltage transformation. These can be circuit breaker panels in houses, transformers on poles, or substations. These are all examples of **ConductorTopologyNodes**, or `CTN` for short. When a `CTN` becomes the focal point of a constraint on power flow - like Keene Rd in our demo - the corresponding GNode can get its role upgraded by the *GNodeFactory* from ConductorTopologyNode to *MarketMaker*.

GNodes with this role are **copper GNodes**, and they typically are *passive* - that is, they do not have actors, but instead serve the function of helping track the grid topology.

Back to Lexicon

3.10.5 CoreGNodeRole

The CoreGNodeRole are a pared-down version of GNodeRole, restricted to:

- TerminalAsset
- AtomicTNode
- MarketMaker
- AtomicMeteringNode
- ConductorTopologyNode
- InterconnectionComponent

and Other (to be used for all other GNodeRoles). These are the roles that perform crucial functions in establishing the grid topology and running the markets. In order to be assigned any of these roles, the GNodeFactory must authorize the assignment.

For other roles, the GNodeFactory only needs to authorize the *creation* of the GNode (and assigns it a default CoreGNodeRole of **other**).

GridWorks is designed to have a single GNodeFactory, which concerns itself with mapping out the copper grid and the validation of metering.

However, GridWorks encourages the formation of *multiple* GNodeRegistries, each of which can create their own breakdown of GNodeRoles.

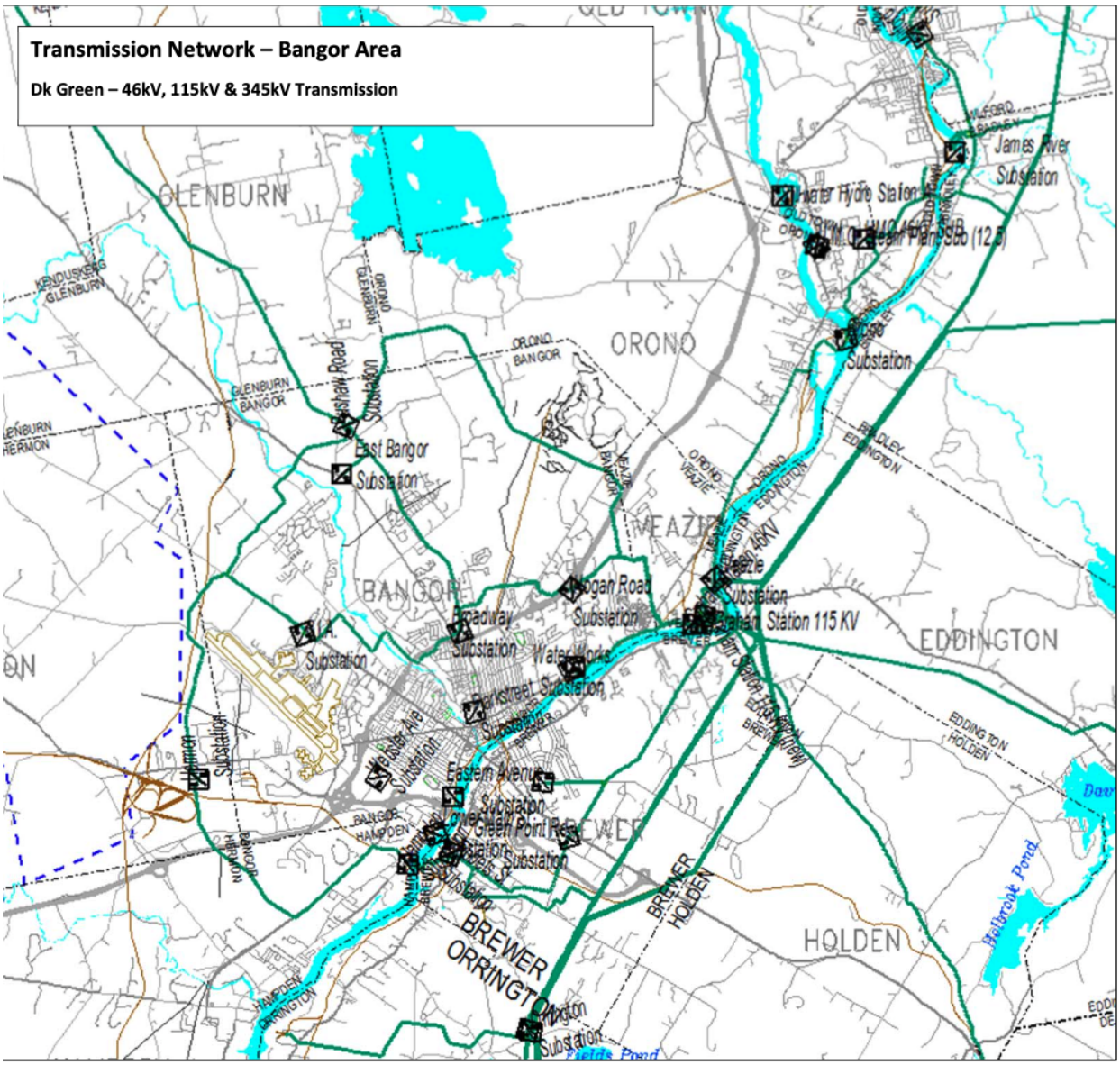
Back to Lexicon

3.10.6 Discoverer

A **Discoverer** is an entity with an Algorand address who solicits the creation of a new ConductorTopologyNode from the GNodeFactory.

The mechanism for adding new ConductorTopologyNodes to the GNodeFactory is still a work in progress. It will likely involve providing data (for example, a utility map like the one below, with annotations) about the location of a new ConductorTopologyNode in relation to known existing ConductorTopologyNodes, as well as a period where this information can be reviewed and/or challenged.

The Discoverer will likely be issued a DiscoveryCertificate to acknowledge their contribution to mapping out the grid.



Back to Lexicon

3.10.7 DispatchContract

The DispatchContract is an Algorand Smart Contract between the AtomicTNode and Scada GNode Actors serving the same Transactive Device.

It:

- Helps the SCADA initially determines *what* credentials an AtomicTNode must provide

in messages in order for the SCADA to accept dispatch commands - Is the *eventual* source of authority on the state of its global True/False variable *AtnInCommand* - Is the primary source of audit data for power and energy injection/withdrawal by the TransactiveDevice.

Back to `Lexicon <lexicon.html> `_

3.10.8 GNode

A **GNode** is the fundamental building block of GridWorks. The *G* stands for grid.

Many of the lead roles in the Millinocket demo are GNodes: the SCADA, the TerminalAsset, the MarketMaker and the AtomicTNodes.

Copper GNodes

The electric grid can be reasonably well modeled as a graph, where an edge in the graph represent a copper wire and cable with relatively uniform electrical characteristics (mainly no sudden intentional change in voltage via a transformer) and the nodes in the graph represent places where two or more edges meet. Note that an electrical device attached to the electric grid can also usually meet the characteristics of an *edge*, where one endpoint is electrical ground.

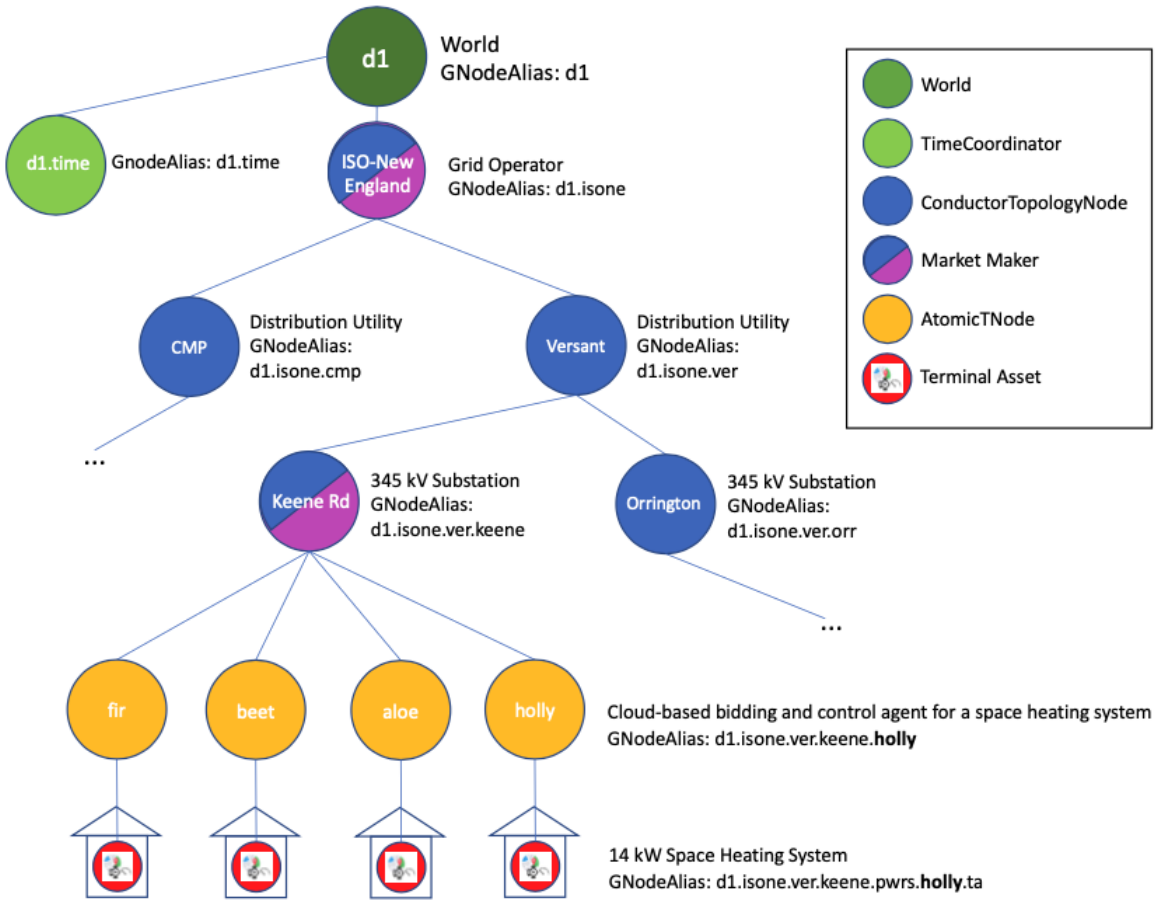
Edison originally made a bunch of local DC powerplants, and transmit low-voltage DC to businesses and homes. This is what he made in New York City.

Tesla proposed an alternative system that involved alternating current (AC) generation and transformation. This allowed for high-voltage transmission.

Edison's grid had generators everywhere, Tesla's (which is what we have implemented) has substations and other transformers everywhere.

What this means is there is a natural hierarchy on our AC grid based on voltage.

We start out with a set of what we call **Copper GNodes** which represent a subset of the nodes in this copper graph. We then create a **spanning tree** that tracks the natural hierarchy of voltage on the grid. While the electric grid does have loops, it doesn't actually have that many - put another way, the edges of this spanning tree capture *most* of the edges in the graph. We capture the remaining edges by turning them into additional GNodes with GNodeRole InterconnectionComponent.



In SDK:

- TODO: hyperlink to GNode dataclass inside the SDK Data Classes
- TODO: hyperlink to GNodeGt inside the SDK Types

BaseGNodes

The Copper GNodes have the GNodeFactory as their global authority. The remaining GNodes have various GNodeRegistries as their global authority.

The GNodeFactory does not maintain or track information about the devices themselves - this is done in the GNodeRegistry.

This level of abstraction is implemented by creating BaseGNodes, which are the objects that the GNodeFactory authorizes and maintains, and creating GNodes, which are a small extension of BaseGNodes.

Back to Lexicon

3.10.9 GNodeAlias

GNodes have structured, mutable identifiers called GNodeAliases that serve several important functions in GridWorks:

- taken together, they help define the topology of the electric grid
- they provide organizational structure to the communication between actors, as well as to the time-series state data generated.

Note for most developers, `alias` usually means an ephemeral or temporary moniker. `GNodeAlias` is an exception to this rule.

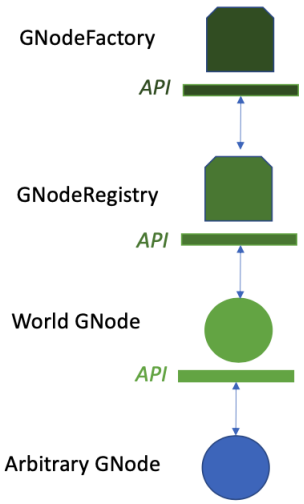
Back to Lexicon

3.10.10 GNodeFactory

The GNodeFactory is responsible for the topology of the GNodeTree - and importantly its copper sub-tree. It also managing the process of creating and updating Algorand certificates (TaDeeds, TaTradingRights, DiscoveryCertificates). This provides the foundation for scalable, decentralized verification of the bijection between online avatars and their real-life partners (e.g. TerminalAssets <-> Transactive Device). This validation process uses the Algorand blockchain.

There are several layers of indirection between an arbitrary GNode and the GNodeFactory.

A unique World GNode sitting at the top of a World's GNodeTree and – for its descendants – acts as the Single Source of Truth (SSoT) for updates related to the GNodeTree and associated data. Under the hood, the World GNode's actor is responsible for managing the applications representing the GNodes in that World (captured by the concept of GNodeInstance).



The GNodeFactory also manages WorldInstances. This allows for multiple simulations to run that share the same GNodeTree. Scenarios where actual money changes hands (e.g., using the Algorand Mainnet instead of TestNet or sandbox) occur in the production universe. To prevent the obvious potential issue of double-counting, there is only one WorldInstance in the production universe.

Although the GNodeFactory spawns WorldInstances, a World's GNode actor does not directly communicate with the GNodeFactory. This is because the GNodeFactory does not track all of the relevant information about GNodes. For example, detailed information about a GNode's associated Physical Device is not managed by the GNodeFactory.

In order to get its information about a specific GNode, therefore, the World GNode actor communicates with the GNodeRegistry for that GNode. The GNodeRegistry is the authority for certain kinds of information that the GNodeFactory

does not track. Under the hood, that GNodeRegistry communicates with the (unique) GNodeFactory in that universe for key topological and identity information.

The open-souce repository for the GNodeFactory can be found here.

GNfAdminAddr

GnfAdminAddr is the primary Algorand address associated to the GNodeFactory. This address is used in numerous validation checks for GridWorks types. It is available as an environment variable.

Listing 18: Inspect the dev universe GnfAdminAddr

```
from gridworks.gw_config import Public

gnf_admin_addr = Public().gnf_admin_addr
assert gnf_admin_addr ==  "RNMHG32VTIHTC7W3LZOEPTDGREL5IQGK46HKD3KBLZHYQUCAKLMT4G5ALI"
```

Note that the GNodeFactory is designed to be unique. However, as a developer, you can run GridWorks simulations on your laptop, with no Internet. GridWorks has the concept of Universe, which can be Dev, Hybrid or Production. Each of these has a unique GnfAdminAddr.

TaValidatorFundingThresholdAlgos

In order to be certified as a TaValidator, an entity must put enough skin in the game to show that they are serious about doing TerminalAsset validations. They are required to fund their 2-sig Multi [GnfAdminAddr, TaValidatorAddr] with this amount. In the Dev Universe, this is set to 100 Algos.

Listing 19: Inspect dev TaValidatorFundingThesholdAlgos

```
from gridworks.gw_config import Public

threshold = Public().ta_validator_funding_threshold_algos
assert threshold ==  100
```

TaDeedConsiderationAlgos

A TaOwner must fund their TaDaemonAddr with an initial consideration in order to receive their initial TaDeed and TaTradingRights (or more accurately, for the TaDaemonAddr to receive these). This account will be actively involved in transactions with an AtomicTNode as the AtomicTNode transacts on behalf of the TerminalAsset in markets. Prior to entering into markets, initial requirements may be made about the funding level of the TaDaemonAddr - roughly equivalent to the energy costs incurred by a month of trading activity.

In the Dev Universe, the TaDeedConsiderationAlgos is set to 50 Algos.

Listing 20: Inspect dev TaValidatorFundingThesholdAlgos

```
from gridworks.gw_config import Public

threshold = Public().ta_deed_consideration_algos
assert threshold ==  50
```

Back to Lexicon

3.10.11 GNodeInstance

A **GNodeInstance** is the one of the layers of abstraction connecting a GNode with a running app in a Docker container. At any point in time, a GNode can be represented by only one GNodeInstance.

Every GNodeInstance is subordinate to a Supervisor GNodeInstance which runs a SupervisorContainer - typically a docker container running in the cloud. The SupervisorContainers are managed and spawned by a World GNodeInstance, which sits at the top of the GNodeTree and is responsible for staying in touch in with the GNodeFactory for lifecycle status and private keys of its GNodes.

The Supervisor monitors the health of its subordinates with heartbeats, and is responsible for killing and re-spawning any subordinate that fails to receive or send messages in a reasonable timeframe.

GNodeInstanceId

One of the primary ways GNode actors communicate with each other is via RabbitMQ messages. The actors share their GNodeInstanceId in these messages in order to establish their credentials.

At the initiation of a conversation, a GNode's conversation partner can check in with the World Actor about the credentials of the inbound message by getting a verification that the GNodeInstanceId matches the GNodeAlias.

Most actors find their GNodeInstance because they are on a docker instance that was started by the World Actor, and the World Actor has placed a file with the relevant GNodeInstance data in the container. If and when that container dies, a new GNodeInstance will be created for the GNode.

Scada GNodeInstanceId

A Scada actor live on a Scada device that is sensing and controlling a Transactive Device. The Scada's process for getting assigned to a GNodeInstance is therefore slightly different.

At the time of its installation, a SCADA device's GNodeAlias is naturally determined by the GNodeAlias of its TerminalAsset (by appending *scada*).

But the system needs to proof that a SCADA actor showing up with that GNodeAlias is in fact running on the physical device.

Note that the TaOwner holds a TaDeed establishing their ownership of the associated TerminalAsset (in GridWorks) and Transactive Device (in the real world). This means if the TaOwner signs a message with their private key, anybody can check that the entity that signed that message owned the TerminalAsset.

The TaOwner therefore creates an Algorand account for the Scada actor, puts that account's secret key on the Scada device, and then sends a signed request to the GNodeFactory to create the Scada GNode with an associated ScadaAlgoAddr (the public address of the Scada's Algorand account). When the Scada is establishing its identity with the DispatchContract it signs its messages with this key.

Back to Lexicon

3.10.12 GNodeRegistry

While there is a single *GNodeFactory*, GridWorks encourages the formation of multiple GNodeRegistries. Upon creation by the GNodeFactory, each GNode is assigned to a unique GNodeRegistry.

That GNodeRegistry is the authority for information about the GNode that is beyond the concern of the GNodeFactory.

GNodeRole Assignment

If the GNodeFactory has assigned a *CoreGNodeRole* different than the default `Other` to a GNode (such as *TerminalAsset*, *AtomicTNode*, etc) then the GNodeRegistry's *GNodeRole* for the GNode must match that CoreGNodeRole. These are the Roles critical for tracking the copper of the electric grid. However, the GNodeRegistry may assign a GNodeRole of its choosing (such as *TimeCoordinator*) to any GNode whose CoreGNodeRole is `Other`.

Device data

AtomicTNode GNode actors need to maintain realistic models of their Transactive Devices in order to do a good job of buying energy and meeting the Service Level Agreement of their device. For example, if a heating system adds additional storage capacity or has a significant change in its heat distribution system, the AtomicTNode needs some way of tracking this. Certain changes to the underlying Transactive Device will actually provoke the retirement of that AtomicTNode/TerminalAsset pair (for example, if the heating system is replaced by a heating system with 3 times the power capacity). Others will just require adjustments to how the AtomicTNode operates.

The SCADA GNode actors have additional requirements, since they run on their associated physical device. For example, a SCADA needs to know *how* it is sensing power, and this depends of course on its device (e.g. does it have an embedded power meter or is it connected to an eGauge meter using modbus over TCP with the eGauge protocol).

Organizing and maintaining this information is the job of the GNodeRegistry. There is significant discretion in this organization process, and different GNodeRegistries may choose how they do it. GridWorks Energy Consulting has designed one, focused initially on serving thermal storage space heat. A company designing AtomicTNodes and/or SCADA systems for an entirely different class of Transactive Device may choose to collaborate with GridWorks Energy Consulting to expand the scope of the GridWorks GNodeRegistry, or design their own.

Back to Lexicon

3.10.13 GNodeRole

The GNodeRole is an enum categorizing GNodes by function. The Role will determine, for example, whether the GNode has a code actor representing it (for example, an AtomicTNode) or whether it is a passive object that is discussed by actors (typically, the GNodes that represent copper on the Grid or physical devices attached to the grid). If the GNode has an actor, its GNodeRole will also typically determine a set of basic messages sent and received by the GNode. A GNode's role is determined by the GNodeFactory if it represents copper on the grid (or if it is the World GNode), and otherwise is determined by the GNode's GNodeRegistry.

- **GNode** This is a default role when more information is not available
- TerminalAsset **CoreGNodeRole**
- AtomicTNode **CoreGNodeRole**
- MarketMaker **CoreGNodeRole**
- AtomicMeteringNode **CoreGNodeRole**
- ConductorTopologyNode **CoreGNodeRole**
- InterconnectionComponent **CoreGNodeRole**
- World
- TimeCoordinator

- Supervisor
- Scada
- PriceService
- WeatherService
- AggregatedTNode

Back to Lexicon

3.10.14 GNodeStatus

GNodeStatus is an Enum for managing GNode lifecycle.

Pending

The GNode exists but cannot be used yet. The GNodeId and GNodeAlias are reserved at this point.

Used for example when a TerminalAsset has been created but does not yet have its TaDeed.

Active

For passive GNodes that represent physical objects, like TerminalAssets, a Status of active means there is evidence supporting the existence of the TerminalAsset. A GNodeStatus of active is required in order for any contract to be entered to that requires trading rights for and/or a deed for a passive GNode.

For GNodes with actors, a Status of active is required for any authorized actor to send a message representing itself as that GNode.

If a GNode is Active, its parent GNode MUST ALSO be active.

PermanentalyDeactivated

For passive GNodes, like TerminalAssets, this designation means no contracts may be entered into and no deed may be held for this GNode. For active GNodes, no message may be sent from this GNode. Once a GNode is PermanentlyDeactivated it can have no other status in the future.

Suspended

The GNode cannot be used, but may become active in the future. This is used, for example, if certification for a TerminalAsset has lapsed.

Back to Lexicon

3.10.15 InterconnectionComponent

The copper spanning tree inside the GNode Tree captures many of the cables and lines on the grid. However, since the electric grid does have loops, we need to introduce a GNodeRole specifically for cables and wires that are not part of the spanning tree. This is the **InteconnectionComponent**.

Back to Lexicon

3.10.16 Market Bid

Bidders in MarketMaker markets can be GNodes with the following roles:

- AtomicTNode
- MarketMaker
- AggregatedTNode

To start understanding how this works, you can assume that the bidder is an AtomicTNode. (The other roles allow for different ways of aggregation and scaling).

In most commodity exchanges, participants can provide *bids* (participants who want to buy the commodity) or they can provide *offers* (participants who want to sell the commodity).

For GridWorks, both sides submit the same kind of bid, in the form of an *atn.bid* object.

The fundamental part of the *atn.bid* is a BidList, which is a set of unitless price/quantity pairs. The bid also includes the units in question, for example price in USDPer Mwh and quantity in AvgMW. Since there is not a clear convention about whether positive power means *injecting* or *withdrawing*, there is also a boolean toggle for this (InjectionIsPositive).

We describe the convention and meaning assigned to these messages, and give a few examples. TODO: add!

Back to Lexicon

3.10.17 MarketMaker

A **MarketMaker** is a Copper GNode that runs an electricity market. MarketMaker is an essential *GNodeRoles* in GridWorks: a hierarchy of MarketMakers is responsible for coordinating Transactive Devices in a way that respects the copper constraints on the grid and turns these transactive devices into balancing resources. A simple open-source MarketMaker available on github is used in the *Millinocket tutorial*.

How AtomicTNodes bid

If you are "bidding into an the energy market" - whether run by the New England Grid Operator or a GridWorks MarketMaker - the bid is about energy consumed or provided over a *specific interval of time*. An AtomicTNode providing a bid to a MarketMaker is bidding into a MarketSlot. The MarketSlot specifies:

- The start time
- The MarketType (determines properties shared with other MarketSlots of the same MarketType)
- The Alias of the MarketMaker GNode

The Atn provides its bid via the GridWorks type atn.bid. We explain what is going on in this bid. We will work an example below with the MarketMaker for the Keene Rd constraint in the *Millinocket demo*. This MarketMaker has a *single* MarketType, which means for example that the question "what is current Keene Rd price?" is well-formed, since at any time there is only one MarketSlot containing that time.

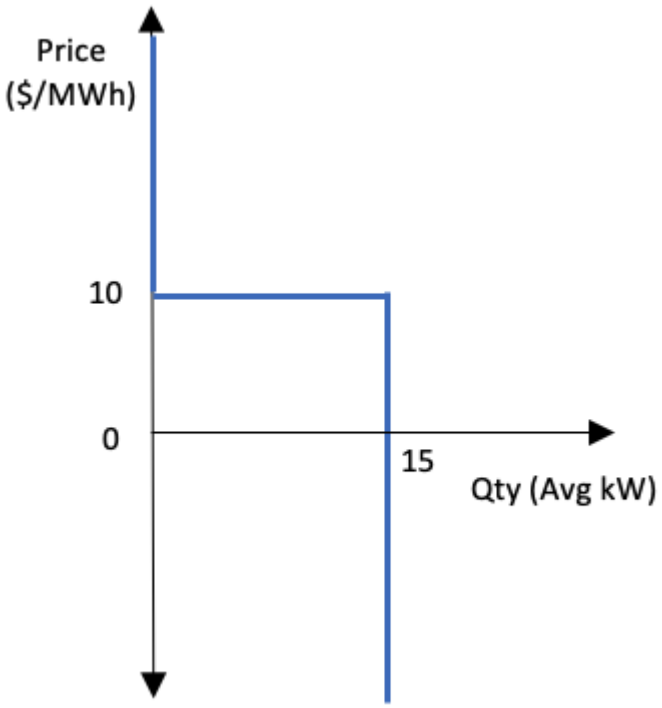
The `d1.isone.me.ver.keene` MarketType is for hourly real-time energy. Bids to a MarketSlot stop being accepted 5 minutes before the start of the MarketSlot. The price units are UsdPerMWh and the quantity units are AvgkW. Lastly, there is a global upper bound of $10,000 UsdPerMwh on price. It is identified globally within GridWorks by the string `rt60gate30b`.

Let's examine a Millinocket *AtomicTNode* (ATN) bidding into the Keene Rd MarketMaker's `rt60gate30b` MarketSlot starting at 5 am Jan 2020 UTC.

PqPairs in AtnBid	
Price	Qty
$10,000/MWh	0 Avg kW
$10/MWh	15 Avg kW

This list of pairs defines quantity as a function of price. For all pairs [P,Q] in the PqPairs list:

- [**Price p**]: For any price p below P (in $ per MWh), down to the next listed price (if any)
- [**Quantity as fn of Price p**]: I will consume (or demand) on average Q kWs in this MarketSlot.



Here is a graph of this function:

Each MarketType has a maximum price PMax; `rt60gate30b` has a PMax of $10,000/MWh. While this price is arbitrary, it is necessary to specify a maximum. Specifically, the Price in the *first pair* of PqPairs must be PMax.

This, therefore, is the bid that an electric heating system would submit to express that it will want to turn on full (15 kW) if the price is at or below $10/MWh, and otherwise it will turn off.

Back to Lexicon

3.10.18 MarketSlot

A MarketMaker can run several types of Markets. For example, it can run an hourly real-time market and also an ancillary services market for Regulation.

This is captured by the concept of MarketType.

A given MarketType partitions time into MarketSlots. For example, an hourly market will have a new MarketSlot that starts at the top of each hour.

MarketSlotName

The MarketSlotName encodes:

- The MarketType (the first word)
- The start of that MarketSlot (last word)
- The GNodeAlias of its MarketMaker (middle words)

Example: rt60gate5.d1.isone.ver.keene.1673539200

Back to Lexicon

3.10.19 MarketType

A MarketMaker can run several types of Markets. For example, it can run an hourly real-time market and also an ancillary services market for Regulation.

This is captured by the concept of **MarketType**. Each MarketType has a unique name, and the list of possible MarketTypes is encoded in the GridWorks enum MarketTypeName.

Additional information about the Market - the duration of the market slots, the duration of gate closing (the time before the start of a MarketSlot by which all bids must be received) - is encapsulated in the MarketType dataclass.

3.10.20 PriceService (GNodeRole)

A **PriceService** is an actor responsible for providing price forecasts for the markets run by *MarketMakers*. These services are typically provided via RabbitMQ using a pubsub pattern. *AtomicTNodes* will subscribe to price services.

Given the importance of good forecasting for the financial performance of AtomicTNodes, there is very likely a business opportunity in providing high-quality, localized forecasting.

Back to Lexicon

3.10.21 Representation Contract

Back to Lexicon

3.10.22 SCADA

In the context of GridWorks, SCADA refers to:

1. A device, and software, that monitors and controls a TerminalAsset
2. A GNode representing the above.

SCADA is also a more generalized industry term for any device capable of supervisory control and data acquisition.

The functions of the SCADA are pretty straightforward and obvious, given the goals of transactive energy. A SCADA needs to:

1. Sense and report information relevant to the performance of the TerminalAsset, including but not limited to the power and energy metering;
2. Let its AtomicTNode call the shots on how the TerminalAsset uses (or provides) electrical power, whenever the two are communicating; and
3. Do a decent job of running the TerminalAsset on its own when it loses communications with its AtomicTNode.

Go here to examine an example of SCADA software for a thermal storage heating system.

Back to Lexicon

3.10.23 Supervisor (GNodeRole)

A **Supervisor** is responsible for the actors running in a GridWorks container. For more details on these mechanics go here.

Back to Lexicon

3.10.24 TaDaemon

A TaDaemon (Ta for *TerminalAsset*) is a piece of code devoted to serving its *TaOwner* (usually a human). You can think of it as a combined butler, money manager and personal lawyer. The TaDaemon has an Algorand address (TaDaemonAddr). The TaOwner's ownership of the TerminalAsset is established by the ownership of a *TaDeed* by the TaDaemonAddr.

The TaDaemon has two main functions.

- It keeps the TaDeed and *TaTradingRights* up to date
- It enters into a representation contract at the behest of its TaOwner.

The first task is managed in coordination with the *GNodeFactory*, and is fairly simple: when the GNodeFactory makes a new TaDeed for its TerminalAsset, the TaDaemon returns the old TaDeed and opts into the new TaDeed.

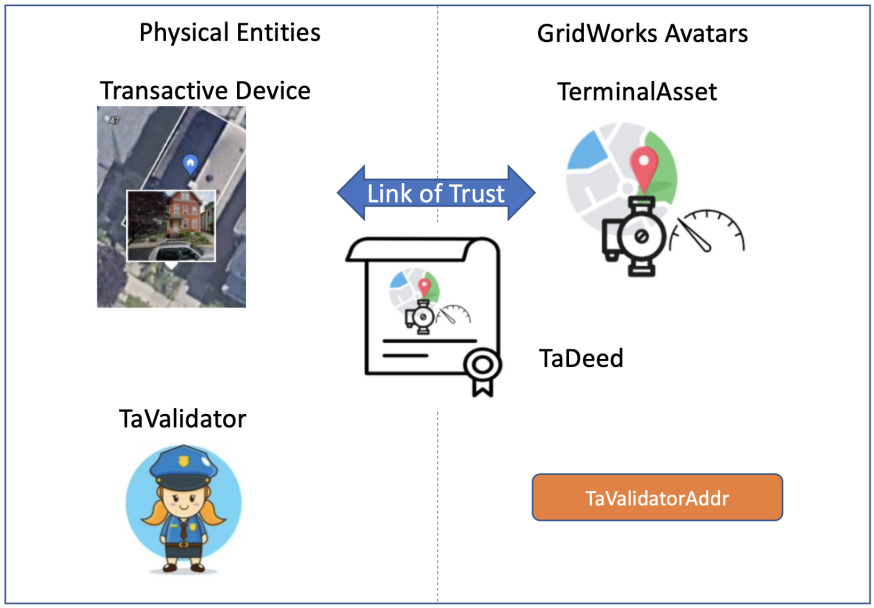
The second task is more complex. It involves providing the TaTradingRights to an *AtomicTNode*, and entering into a long-term financial agreement that involves both receiving and sending money in a triangle involving the AtomicTNode and its *MarketMaker*.

Back to Lexicon

3.10.25 TaDeed

A TaDeed is a blockchain object associating a Transactive Device to its GridWorks avatar, which is a TerminalAsset. It is analagous in some ways to the deed of a house, in the way that it establishes ownership of a physical asset. However, it goes beyond establishing ownership, since it also provides a mechanism for reducing the counterparty risk in trusting the physical meaning of energy transactions.

Link of trust



If you search *blockchain* and *trustless*, you can find a lot of information about how blockchains enable transactions that do not require the counterparties to trust each other or a centralized third party. Indeed, this is a foundational feature of blockchains.

But blockchains *do* require trust. Operating with blockchain requires trust that *collective agreement about the meaning of symbols* will continue to exist. This is true with any form of currency. In addition, a person must have *faith* that the underlying chain itself will remain viable. Finally, as the Ethereum hard fork demonstrates, collective trust of the integrity and competence of a single individual within crypto can have profound ramifications.

For GridWorks, there is a global physical entity – the electric grid, with all of the electrical devices hanging off of it – that requires faithful modeling for GridWorks to support and coordinate the real-time operations of the grid. The decentralized, hierararchical structure of MarketMakers designed to coordinate the flow of real electrons on real copper need some reason to **trust** its blockchain transactions of energy for money. More specifically, when a MarketMaker examines an AtomicTNode's DispatchContract for the audit trail of actual energy and power consumed, there must be some reason for it to trust that these are are accurate representations of *what*, *when*, and *where* the corresponding Transactive Device consumed or provided electricity.

The GridWorks solution to this is a framework allowing for decentralized entities (TaValidators) establishing their credentials within a localized area - the geographical locations served by the grid in the sub-tree under a MarketMaker GNode. In order to be effective in their role, TaValidators will need to have a strong and positive reputation both

- within the community and/or communities living in that geographical area; and
- with any other counterparties required to make the MarketMakers run correctly in the existing regulatory environment (typically incumbent companies in the electricity sector: grid operators, utilities, and/or energy suppliers).

These TaValidators establish the **links of trust** between Transactive Devices and their TerminalAssets. Note that a centralized registry must be involved in this process in order to ensure various uniqueness constraints (bijection between Transactive Devices and TerminalAssets, unique GNode for any GNodeAlias across time, and unique unique GNodeAlias for any GNode at any instant of time). This centralized registry is the GNodeFactory, and this is why the creation of a TaDeed is an act of co-creation (requiring both signatures) between the GNodeFactory and a TaValidator.

When possible, GridWorks reduces or removes mechanisms that could force or encourage unhelpful patterns of taking on faith the word and expertise of centralized authorities. Indeed, the grand GridWorks ambition is to move the heavy lifting of grid control and coordination *out* of the purview of centralized Grid Operators and into the purview of a decentralized hierarchy of MarketMakers. As another example, *any* entity can become a TaValidator in GridWorks. This is in contrast to a design decision that, say, would only allow existing utilities or other large energy companies to authorize Transactive Devices.

That being said, we ignore the physical reality of the copper to our own collective sorrow, a fact that is dawning slowly on humanity as renewable energy starts to scale and we begin to recognize the ramifications of mis-aligned regulatory policy. Therefore, we choose to highlight this necessity of having a reason to trust the alignment by calling the creation of TaDeed by a TaValidator a process of creating a **link of trust** between a real-world device and its GridWorks avatar.

TaDeed technical details

A TaDeed can either be an Algorand Standards Asset (ASA), or an Algorand Smart Contract. In either case, the TaDeed makes publicly available both the GNodeAlias of the TerminalAsset, and the Algorand address of the TaValidator.

Why are there two variants of a TaDeed?

- On the one hand, ASAs are easier for people learning the ropes with Algorand development. Anyone who has made an NFT will be familiar with the mechanics, and it is easier to track transactions in an online tool like Algosearch.
- On the other hand, the GNodeAlias length cannot exceed 32 characters for an ASA TaDeed. Many TerminalAssets will start with GNodeAliases less than 32 characters, but some will not. In addition, as the online mapping of the electric grid captured by the set of GNodeAliases expands, a TerminalAsset (which represents a leaf node in that tree) will likely get longer.

There is a practical issue with this: the unique ids for ASAs are ints, and the unique ids for SmartSigs are strings (their public AlgoAddress).

ASA TaDeed specs

An ASA TaDeed is an Algorand Standard Asset where:

- Creator (aka Sender) is *either*
  - a 2-sig MultiAddress [GnfAdminAddr, Addr2] where Addr2 is the public address of a TaValidator *or*
  - GnfAdminAddr
- Total is 1
- UnitName is “TADEED”
- ManagerAddr is GnfAdminAddr
- AssetName has the LeftRightDot format, and is no more than 32 characters

SmartSignature TaDeed specs

Back to Lexicon

3.10.26 TaOwner

The **TaOwner** is the entity who owns a *Transactive Device* in the real world, and its associated *TerminalAsset* in GridWorks. Ownership of a TerminalAsset by the TaOwner is established by a *TaDeed*, which is an Algorand object.

The TaDeed for a TerminalAsset will change from time to time, for example as the *GNodeAlias* for the TerminalAsset changes. The GNodeAlias is a structured identifier that helps capture the topology of the electric grid, and as such it is mutable, perhaps changing a handful of times over the lifetime of the GNode.

The TaOwner will often be a human. It is not appropriate, therefore, to expect an action from the TaOwner when the TaDeed changes. This is done by putting the TaDeed into the care of a *TaDaemon* application.

Back to Lexicon

3.10.27 TaTradingRights

A TaTradingRights certificate (*Ta* for TerminalAsset) is an Algorand blockchain object that provides an AtomicTNode with the authority to:

- enter in to a DispatchContract with a SCADA; and
- participate in MarketMaker markets on behalf of the TerminalAsset.

TaTradingRights technical details

As with TaDeeds, TaTradingRights can either be Algorand Standards Assets (ASAs), or Algorand Smart Contracts. In both cases, the TaTradingRights makes publicly available the GNodeAlias of the TerminalAsset, and the Algorand address of the TaValidator.

ASA TaTradingRights specs

An ASA TaTradingRights certificate is an Algorand Standard Asset where:

- Creator (aka Sender) is GnfAdminAddr
- Total is 1
- UnitName is “TATRADE”
- ManagerAddr is GnfAdminAddr
- AssetName has the LeftRightDot format, and is no more than 32 characters

SmartSignature TaDeed specs

Back to `Lexicon <lexicon.html> `_

3.10.28 TaValidator

A **TaValidator** is an entity authorized to validate Transactive Devices. Anyone can become a TaValidator, once they have gone through the validator certification process with the GNodeFactory.

The TaValidator role is a key in establishing the link of trust between a Transactive Devices and its TerminalAsset. In the Millinocket demo, the first step is a fictitious entity called Molly Metermaid becoming a TaValidator.

Molly Metermaid

Role: Trusted Validator

Molly runs a small organization on the mainland that installs equipment and does metering and verification of various efficiency and utility projects. She knows most of the electricians, plumbers and heat pump installers Downeast.

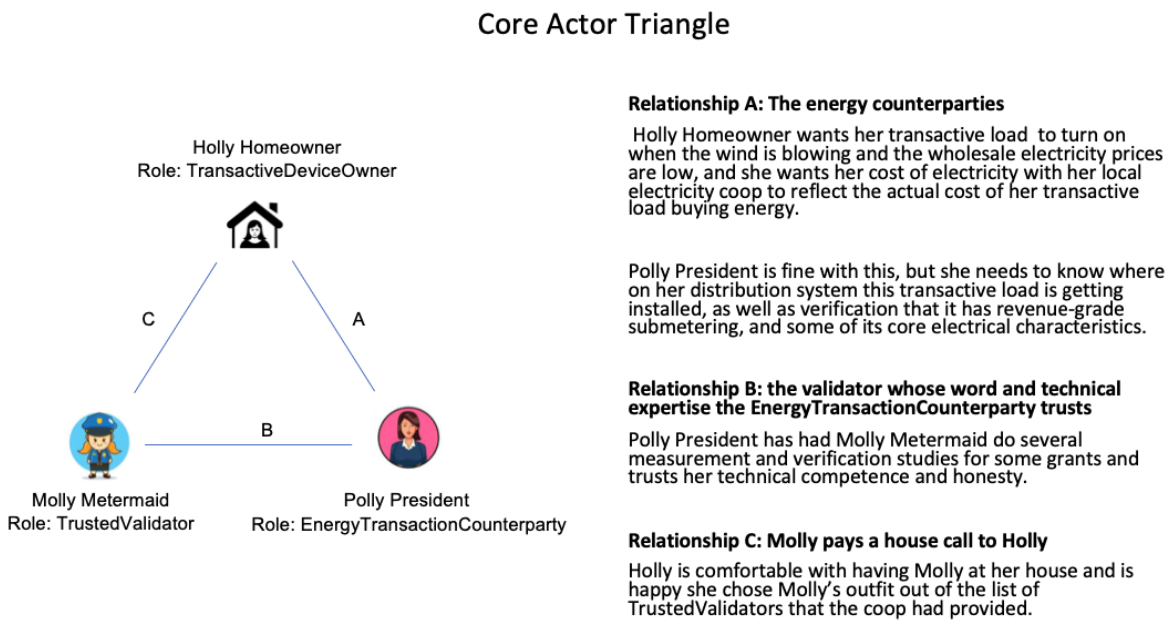
Role

Molly understands that Utilities and Grid Operators need to know where Transactive Devices are located on the electric system, that their meters are accurate and reliable, and enough about the underlying devices to get a sense of how they will impact the electric grid. She has a reputation with local utilities and governmental organizations as somebody whose work is accurate and done on time. She also prides herself on the high level of comfort people have when she is in their house installing measurement and verification equipment. She sees a role for herself as a trusted validator of these critical specifications of this new type of grid asset.

Core Needs

Molly is looking for a simple but totally reliable way to verify and certify that the metering for a Transactive Device is installed correctly and working properly, and at the specified location.

Who is likely to become a TaValidator? Organizations that provide Measurement and Verification services to utilities and grid operators are likely candidates. These are organizations involved in evaluating efficiency progams and Demand Response progams.



TaValidator Certificate

An entity is a TaValidator exactly if its Algorand address (TaValidatorAddr) owns a TaValidator Certificate.

This certificate is an Algorand Standard Asset meeting the following criteria:

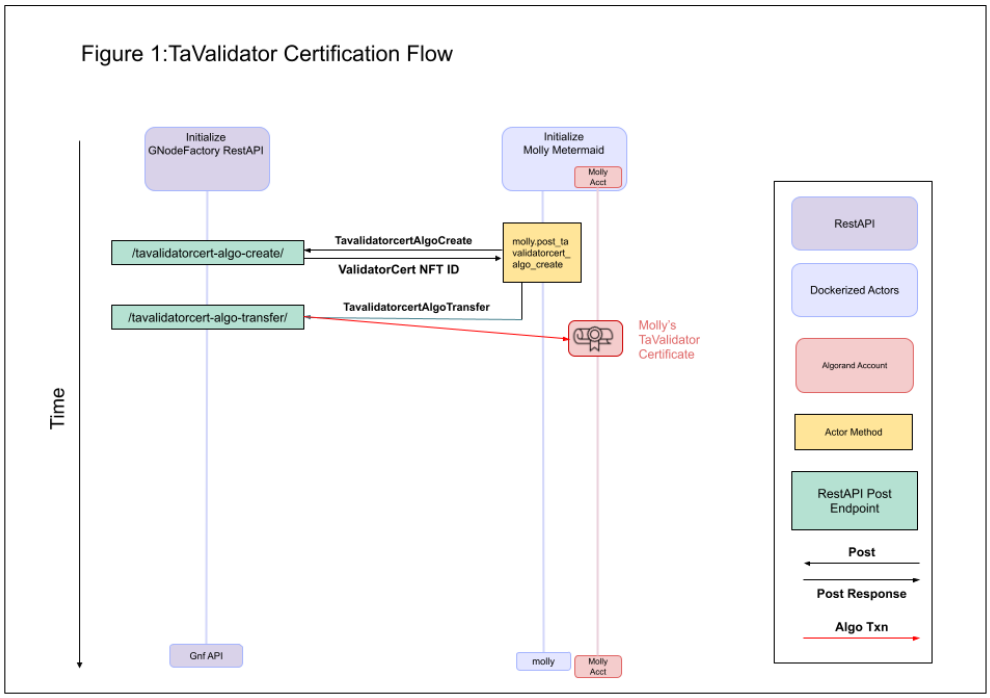
- Creator (aka Sender) is the 2-sig MultiAddress [GnfAdminAddr, ValidatorAddr]
- Total is 1
- Decimals is 0
- UnitName is "VLDTR"
- Manager is GnfAdminAddr
- AssetName its not blank

Unpacking this a bit.

Total An ASA with a total of 1 is an Algorand Non-fungible token, or NFT.

Creator Typically, Algorand Multisig addresses are used when 2 or more signatures are required to sign transactions. This is how we use them: *both* the candidate TaValidator *and* the GNodeFactory's Administrative account (with public address GnfAdminAddr) must sign. That is, they must work together to create the Validator Certificate and transfer it to the TaValidator's address.

TaValidator Certification Flow



Relevant messages:

- TavalidatorcertAlgoCreate (api and sdk)
- TavalidatorcertAlgoTransfer (api and sdk)

Back to Lexicon

3.10.29 Terminal Assets

A GNode whose *GNodeRole* is **TerminalAsset** is a GridWorks avatar for a *Transactive Device*, where this bijective link is established by a *TaDeed*. As such, it can be trusted to represent:

- **An electrical device** connected to the grid that can consume and/or produce electrical power;
- **An electrical meter** that meters exactly the TerminalAsset and has the accuracy characteristics required to meet existing and pending grid balancing challenges (that is, the challenge of keeping electric supply and electric demand in balance on various timescales as wind and solar electricity become more prevalent); and
- **A lat/lon pair** that can be used to capture where the electrical device is connected to the topology of the electric grid.

Here is the GridWorks icon used to represent TerminalAssets

Back to Lexicon

3.10.30 TimeCoordinator (GNodeRole)

A TimeCoordinator is a GNode responsible for managing time in simulation. Typically, a simulation will have a single TimeCoordinator which is a direct descendant of the WorldGNode. For example, `d1.time` is the alias of the TimeCoordinator in the *Millinocket demo*, using this gridworks repo.

It is also possible to have a hierarchy of TimeCoordinators working together. This could be useful in a simulation where a loosely tied sub-grid requires higher time resolution in order for the networks simulations to converge.

Back to Lexicon

3.10.31 Transactive Device

A **Transactive Device** is real-world electrical device that is consuming or producing energy *transactively*. Consider the first *Maine demo* installation at the house of one of the demo team members. The space heating distribution system in their house is an underfloor heating system on the first floor, and a combination of radiant heat and air on the second. The distribution system can keep the house warm with water at 120F, even on the coldest day. Plumbed into this is 3 water tanks with 9 kW of resistive boost elements and a Spacepak heat pump with a max draw of 4.5 kW, capable of producing 135 F water. The heat pump and resistive elements are metered by an eGauge meter.

Transactive Device

In the Physical World

- At a specific geographical location

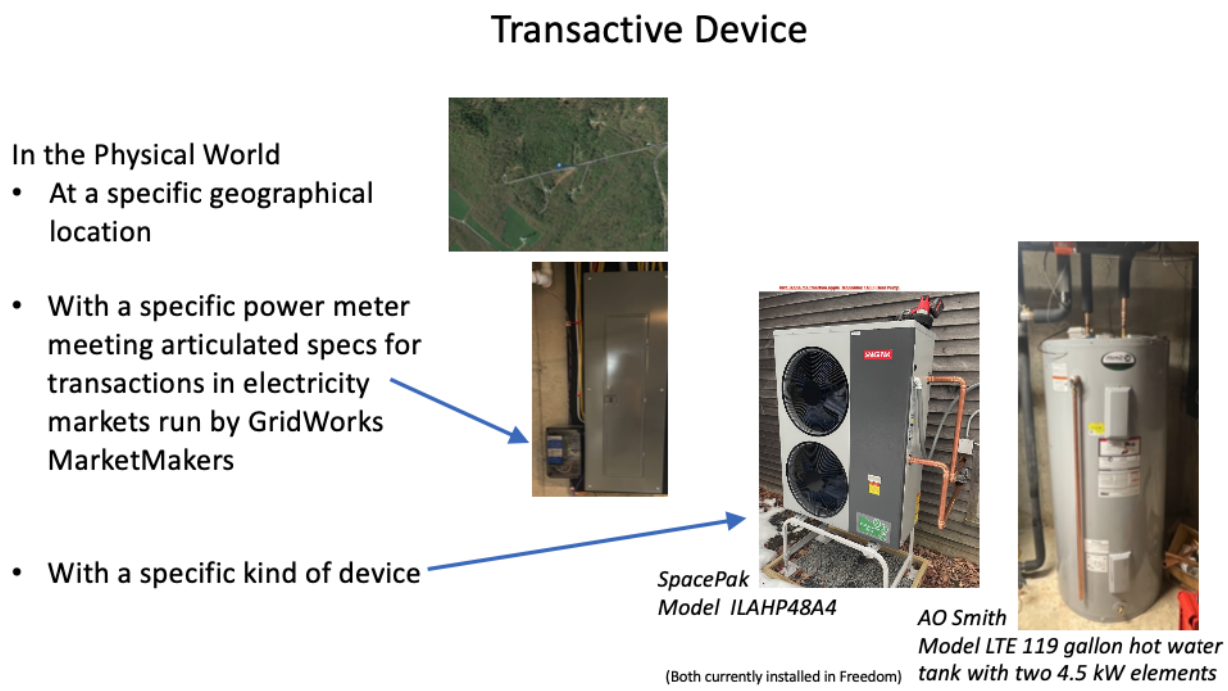
First Installation at 105 Deer Hill Rd. Freedom, Maine.

Transactive Device

In the Physical World

- At a specific geographical location
- With a specific power meter meeting articulated specs for transactions in electricity markets run by GridWorks MarketMakers

eGauge model 4030 multi-port electric meter installed in Freedom



In summary, a **Transactive Device** is a tuple of 3 physical things:

- **A geographical location** of the location where the electrical device connects to the electric grid.
- **An electrical meter** that meters exactly the electrical device and has the accuracy characteristics required to meet existing and pending grid balancing challenges (that is, the challenge of keeping electric supply and electric demand in balance on various timescales as wind and solar electricity become more prevalent); and
- **An electrical device** connected to the grid that can consume and/or produce electrical power;

Back to Lexicon

3.10.32 Transactive Energy Resource

A **Transactive Energy Resource** is a physical resource capable of 24-7, real-time, geographically localized response to grid conditions that can significantly shift and adapt its pattern of electric power use and/or generation with negligible negative consequences on its primary use.

Back to Lexicon

3.10.33 Universe

This concept allows for for development and shared simulations. The hybrid universe allows for coupling real and simulated devices, which can support high-fidelity analysis of outcomes before larger rollouts.

It is an enum in the gridworks SDK.

Dev

Simulation running on a single computer.

- Used for learning and code development.
- Can run without Internet.
- Designed to be run by a single individual on their computer. If you are familiar with Algorand, this is like running in *sandbox dev* mode (and requires the algo sandbox under the hood).
- Time is not unique. That is, you can run the simulation again, using different parameters, and get different data for the same timestamp.
- No security.

Hybrid

Anything goes.

- Designed for multiple people/organizations to interact in a non-production environment.
- Requires Internet.
- Financial transactions are simulated.
- Unique global Hybrid GNodeFactory.
- TerminalAssets can be avatars for either **real** or **simulated** Transactive Devices. Put another way, the validation process for TerminalAssets can be real or simulated.
- Multiple WorldInstances.
- Time is unique per WorldInstance. - In ex-poste analysis, data from different actors can be trusted to refer to the same events. - If all devices are simulated, then WorldInstance time can be decoupled from real time. - Evidently, if there are any real devices, WorldInstance time must track real time.
- Non-production code is allowed to run GNode Actors.
- There is some basic security.

Production

Money at stake.

- Unique global Production GNodeFactory
- Only one WorldInstance.
- Financial transactions are real.
- Time is real.
- Transactive Devices must be **real**. Put another way, the validation process for a TerminalAsset must be done by a real company that is staking its reputation on the validation.
- Only allows production code to run a GNode Actor.
- Production security.

Back to Lexicon

3.10.34 WeatherService (GNodeRole)

A **WeatherService** is an actor responsible for providing weather forecasts. These services are typically provided via RabbitMQ using a pubsub pattern. *AtomicTNodes* with [TradingRights] for *TransactiveDevices* whose electricity use is weather dependant (like heating systems) will subscribe to weather services.

Back to Lexicon

3.10.35 World (as GNodeRole)

The World is an adminstrative GNode responsible for managing and authorizing which actor instances (*GNodeInstances*) are currently representing GNodes.

It gets information about identity (alias, location, private Algorand keys) for its GNodes from the *GNodeFactory*. It is responsible for instantiating GNodeInstances and managing the disbursement of their secrets.

For more details on these mechanics go *here*.

Back to Lexicon

3.11 Type API Specs

3.11.1 BaseGNodeGt


```
{
  "gwapi": "001",
  "type_name": "base.g.node.gt",
  "version": "002",
  "owner": "gridworks@gridworks-consulting.com",
  "description": ". BaseGNode. Authority is GNodeFactory.",
  "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, most␣
↪significant word (on the left) starting with an alphabet character.",
      "example": "dw1.isone.me.freedom.apple"
    },
    "AlgoAddressStringFormat": {
      "type": "string",
      "description": "String of length 32, characters are all base32 digits.",
      "example": "RNMHG32VTIHTC7W3LZOEPTDGREL5IQGK46HKD3KBLZHYQUCAKLMT4G5ALI"
    }
  },
  "enums": {
    "CoreGNodeRole000": {
      "type": "string",
      "name": "core.g.node.role.000",
```


(continues on next page)

(continued from previous page)


```
    "description": "CoreGNodeRole assigned by GNodeFactory",
    "url": "https://gridworks.readthedocs.io/en/latest/core-g-node-role.html",
    "oneOf": [
      {
        "const": "00000000",
        "title": "Other",
        "description": ""
      },
      {
        "const": "0f8872f7",
        "title": "TerminalAsset",
        "description": ""
      },
      {
        "const": "d9823442",
        "title": "AtomicTNode",
        "description": ""
      },
      {
        "const": "86f21dd2",
        "title": "MarketMaker",
        "description": ""
      },
      {
        "const": "9521af06",
        "title": "AtomicMeteringNode",
        "description": ""
      },
      {
        "const": "4502e355",
        "title": "ConductorTopologyNode",
        "description": ""
      },
      {
        "const": "d67e564e",
        "title": "InterconnectionComponent",
        "description": ""
      },
      {
        "const": "7a8e4046",
        "title": "Scada",
        "description": ""
      }
    ]
  },
  "GNodeStatus100": {
    "type": "string",
    "name": "g.node.status.100",
    "description": "Enum for managing GNode lifecycle",
    "url": "https://gridworks.readthedocs.io/en/latest/g-node-status.html",
    "oneOf": [
      {
```


(continues on next page)

(continued from previous page)

```
          "const": "00000000",
          "title": "Unknown",
          "description": "Default value"
        },
        {
          "const": "153d3475",
          "title": "Pending",
          "description": "The GNode exists but cannot be used yet."
        },
        {
          "const": "a2cfc2f7",
          "title": "Active",
          "description": "The GNode can be used."
        },
        {
          "const": "839b38db",
          "title": "PermanentlyDeactivated",
          "description": "The GNode can no longer be used, now or in the future."
        },
        {
          "const": "f5831e1d",
          "title": "Suspended",
          "description": "The GNode cannot be used, but may become active in the future."
        }
      ]
    }
  },
  "properties": {
    "GNodeId": {
      "type": "string",
      "format": "UuidCanonicalTextual",
      "title": "",
      "required": true
    },
    "Alias": {
      "type": "string",
      "format": "LeftRightDot",
      "title": "",
      "required": true
    },
    "Status": {
      "type": "string",
      "format": "g.node.status.100",
      "title": "",
      "required": true
    },
    "Role": {
      "type": "string",
      "format": "core.g.node.role.000",
      "title": "",
      "required": true
    },
```

(continues on next page)

(continued from previous page)

```
    "GNodeRegistryAddr": {
      "type": "string",
      "format": "AlgoAddressStringFormat",
      "title": "",
      "required": true
    },
    "PrevAlias": {
      "type": "string",
      "format": "LeftRightDot",
      "title": "",
      "required": false
    },
    "GpsPointId": {
      "type": "string",
      "format": "UuidCanonicalTextual",
      "title": "",
      "required": false
    },
    "OwnershipDeedId": {
      "type": "integer",
      "minimum": 0,
      "title": "",
      "required": false
    },
    "OwnershipDeedValidatorAddr": {
      "type": "string",
      "format": "AlgoAddressStringFormat",
      "title": "",
      "required": false
    },
    "OwnerAddr": {
      "type": "string",
      "format": "AlgoAddressStringFormat",
      "title": "",
      "required": false
    },
    "DaemonAddr": {
      "type": "string",
      "format": "AlgoAddressStringFormat",
      "title": "",
      "required": false
    },
    "TradingRightsId": {
      "type": "integer",
      "minimum": 0,
      "title": "",
      "required": false
    },
    "ScadaAlgoAddr": {
      "type": "string",
      "format": "AlgoAddressStringFormat",
      "title": "",
```

(continues on next page)

(continued from previous page)

```
      "required": false
    },
    "ScadaCertId": {
      "type": "integer",
      "minimum": 0,
      "title": "",
      "required": false
    },
    "TypeName": {
      "type": "string",
      "value": "base.g.node.gt",
      "title": "The type name"
    },
    "Version": {
      "type": "string",
      "title": "The type version",
      "default": "002",
      "required": true
    }
  }
}
```

3.11.2 GNodeGt

```
{
  "gwapi": "001",
  "type_name": "g.node.gt",
  "version": "002",
  "owner": "gridworks@gridworks-consulting.com",
  "description": "Used to send and receive updates about GNodes. GNodes are the building␣
↪blocks of Gridworks. They have slowly-changing state that must be kept in sync across␣
↪a distributed system. Therefore, they require a global registry to act as Single␣
↪Source of Truth (SSoT). This class is used for that SSoT to share information with␣
↪actors about their GNodes, and the GNodes that they will observe and communicate with.
↪",
  "url": "https://gridworks.readthedocs.io/en/latest/g-node.html",
  "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, most␣
↪significant word (on the left) starting with an alphabet character.",
      "example": "dw1.isone.me.freedom.apple"
    },
    "AlgoAddressStringFormat": {
      "type": "string",
```

(continues on next page)

(continued from previous page)


```
      "description": "String of length 32, characters are all base32 digits.",
      "example": "RNMHG32VTIHTC7W3LZOEPTDGREL5IQGK46HKD3KBLZHYQUCAKLMT4G5ALI"
    }
  },
  "enums": {
    "GNodeRole000": {
      "type": "string",
      "name": "g.node.role.000",
      "description": "Categorizes GNodes by their function within GridWorks",
      "url": "https://gridworks.readthedocs.io/en/latest/g-node-role.html",
      "oneOf": [
        {
          "const": "00000000",
          "title": "GNode",
          "description": "Default value"
        },
        {
          "const": "bdeaa0b1",
          "title": "TerminalAsset",
          "url": "https://gridworks.readthedocs.io/en/latest/transactive-device.html",
          "description": "An avatar for a real-word Transactive Device"
        },
        {
          "const": "8021dcad",
          "title": "AtomicTNode",
          "url": "https://gridworks.readthedocs.io/en/latest/atomic-t-node.html",
          "description": "Transacts in markets on behalf of, and controlling the power␣
↪use of, a TerminalAsset"
        },
        {
          "const": "304890c5",
          "title": "MarketMaker",
          "url": "https://gridworks.readthedocs.io/en/latest/market-maker.html",
          "description": "Runs energy markets at its Node in the GNodeTree"
        },
        {
          "const": "8eb5b9e1",
          "title": "AtomicMeteringNode",
          "description": "Role of a GNode that will become an AtomicTNode, prior to it␣
↪owning TaTradingRights"
        },
        {
          "const": "234cfaa2",
          "title": "ConductorTopologyNode",
          "description": "An avatar for a real-world electric grid node - e.g. a␣
↪substation or transformer"
        },
        {
          "const": "fec0c127",
          "title": "InterconnectionComponent",
          "description": "An avatar for a cable or wire on the electric grid"
        },
```


(continues on next page)

(continued from previous page)

```
        {
          "const": "3901c7d2",
          "title": "World",
          "url": "https://gridworks.readthedocs.io/en/latest/world-role.html",
          "description": "Adminstrative GNode responsible for managing and authorizing␣
↪instances"
        },
        {
          "const": "c499943c",
          "title": "TimeCoordinator",
          "description": "Responsible for managing time in simulations"
        },
        {
          "const": "88112a93",
          "title": "Supervisor",
          "url": "https://gridworks.readthedocs.io/en/latest/supervisor.html",
          "description": "Responsible for GNode actors running in a container"
        },
        {
          "const": "674ad859",
          "title": "Scada",
          "description": "GNode associated to the device and code that directly monitors␣
↪and actuates a Transactive Device"
        },
        {
          "const": "2161739f",
          "title": "PriceService",
          "description": "Provides price forecasts for markets run by MarketMakers"
        },
        {
          "const": "1dce1efd",
          "title": "WeatherService",
          "description": "Provides weather forecasts"
        },
        {
          "const": "db57d184",
          "title": "AggregatedTNode",
          "description": "An aggregation of AtomicTNodes"
        },
        {
          "const": "07f28817",
          "title": "Persister",
          "description": "Responsible for acking events with delivery guarantees"
        }
      ]
    },
    "GNodeStatus100": {
      "type": "string",
      "name": "g.node.status.100",
      "description": "Enum for managing GNode lifecycle",
      "url": "https://gridworks.readthedocs.io/en/latest/g-node-status.html",
      "oneOf": [
```

(continues on next page)

(continued from previous page)

```
        {
          "const": "00000000",
          "title": "Unknown",
          "description": "Default value"
        },
        {
          "const": "153d3475",
          "title": "Pending",
          "description": "The GNode exists but cannot be used yet."
        },
        {
          "const": "a2cfc2f7",
          "title": "Active",
          "description": "The GNode can be used."
        },
        {
          "const": "839b38db",
          "title": "PermanentlyDeactivated",
          "description": "The GNode can no longer be used, now or in the future."
        },
        {
          "const": "f5831e1d",
          "title": "Suspended",
          "description": "The GNode cannot be used, but may become active in the future."
        }
      ]
    }
  },
  "properties": {
    "GNodeId": {
      "type": "string",
      "format": "UuidCanonicalTextual",
      "title": "Immutable identifier for GNode",
      "required": true
    },
    "Alias": {
      "type": "string",
      "format": "LeftRightDot",
      "title": "Structured mutable identifier for GNode",
      "description": "The GNode Aliases are used for organizing how actors in Gridworks␣
↪communicate. Together, they also encode the known topology of the electric grid.",
      "required": true
    },
    "Status": {
      "type": "string",
      "format": "g.node.status.100",
      "title": "Lifecycle indicator",
      "required": true
    },
    "Role": {
      "type": "string",
      "format": "g.node.role.000",
```

(continues on next page)

(continued from previous page)

```
      "title": "Role within Gridworks",
      "required": true
    },
    "GNodeRegistryAddr": {
      "type": "string",
      "format": "AlgoAddressStringFormat",
      "title": "Algorand address for GNodeRegistry",
      "description": "For actors in a Gridworks world, the GNodeRegistry is the Single␣
↪Source of Truth for existence and updates to GNodes.",
      "required": true
    },
    "PrevAlias": {
      "type": "string",
      "format": "LeftRightDot",
      "title": "Previous GNodeAlias",
      "description": "As the topology of the grid updates, GNodeAliases will change to␣
↪reflect that. This may happen a handful of times over the life of a GNode.",
      "required": false
    },
    "GpsPointId": {
      "type": "string",
      "format": "UuidCanonicalTextual",
      "title": "Lat/lon of GNode",
      "description": "Some GNodes, in particular those acting as avatars for physical␣
↪devices that are part of or are attached to the electric grid, have physical locations.
↪ These locations are used to help validate the grid topology.",
      "required": false
    },
    "OwnershipDeedId": {
      "type": "integer",
      "minimum": 0,
      "title": "Algorand Id of ASA Deed",
      "description": "The Id of the TaDeed Algorand Standard Asset if the GNode is a␣
↪TerminalAsset.",
      "required": false
    },
    "OwnershipDeedValidatorAddr": {
      "type": "string",
      "format": "AlgoAddressStringFormat",
      "title": "Algorand address of Validator",
      "description": "Deeds are issued by the GNodeFactory, in partnership with third␣
↪party Validators.",
      "required": false
    },
    "OwnerAddr": {
      "type": "string",
      "format": "AlgoAddressStringFormat",
      "title": "Algorand address of the deed owner",
      "required": false
    },
    "DaemonAddr": {
      "type": "string",
```

(continues on next page)

(continued from previous page)

```
      "format": "AlgoAddressStringFormat",
      "title": "Algorand address of the daemon app",
      "description": "Some GNodes have Daemon applications associated to them to handle␣
↪blockchain operations.",
      "required": false
    },
    "TradingRightsId": {
      "type": "integer",
      "minimum": 0,
      "title": "Algorand Id of ASA TradingRights",
      "description": "The Id of the TradingRights Algorand Standard Asset.",
      "required": false
    },
    "ScadaAlgoAddr": {
      "type": "string",
      "format": "AlgoAddressStringFormat",
      "title": "",
      "required": false
    },
    "ScadaCertId": {
      "type": "integer",
      "minimum": 0,
      "title": "",
      "required": false
    },
    "ComponentId": {
      "type": "string",
      "format": "UuidCanonicalTextual",
      "title": "Unique identifier for GNode's Component",
      "description": "Used if a GNode is an avatar for a physical device. The serial␣
↪number of a device is different from its make/model. The ComponentId captures the␣
↪specific instance of the device.",
      "required": false
    },
    "DisplayName": {
      "type": "string",
      "title": "Display Name",
      "required": false
    },
    "TypeName": {
      "type": "string",
      "value": "g.node.gt",
      "title": "The type name"
    },
    "Version": {
      "type": "string",
      "title": "The type version",
      "default": "002",
      "required": true
    }
  },
  "example": {
```

(continues on next page)

(continued from previous page)

```
    "GNodeId": "575f374f-8533-4733-baf7-91146c607445",
    "Alias": "d1.isone.ver.keene",
    "StatusGtEnumSymbol": "a2cfc2f7",
    "RoleGtEnumSymbol": "234cfaa2",
    "GNodeRegistryAddr": "MONSDN5MXG4VMIOHJNCJJBVASG7HEZQSCEIKJAPEPVI5ZJUMQGXQKSOAYU",
    "TypeName": "g.node.gt",
    "Version": "000"
  }
}
```

3.11.3 GNodeInstanceGt

```
{
  "gwapi": "001",
  "type_name": "g.node.instance.gt",
  "version": "000",
  "owner": "gridworks@gridworks-consulting.com",
  "description": "Used to send and receive updates about GNodeInstances. One of the␣
↪layers of abstraction connecting a GNode with a running app in a Docker container.",
  "url": "https://gridworks.readthedocs.io/en/latest/g-node-instance.html",
  "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"
    },
    "AlgoAddressStringFormat": {
      "type": "string",
      "description": "String of length 32, characters are all base32 digits.",
      "example": "RNMHG32VTIHTC7W3LZOEPTDGREL5IQGK46HKD3KBLZHYQUCAKLMT4G5ALI"
    }
  },
  "enums": {
    "GniStatus000": {
      "type": "string",
      "name": "gni.status.000",
      "description": "Enum for managing GNodeInstance lifecycle",
      "url": "https://gridworks.readthedocs.io/en/latest/g-node-instance.html",
      "oneOf": [
        {
          "const": "00000000",
          "title": "Unknown",
          "description": "Default Value"
        },
        {
```

(continues on next page)

(continued from previous page)

```
          "const": "7890ab0a",
          "title": "Pending",
          "description": "Has been created by the World, but has not yet finished␣
↪provisioning"
        },
        {
          "const": "69241259",
          "title": "Active",
          "description": "Active in its GridWorks world. If the GNodeInstance has an␣
↪actor, that actor is communicating"
        },
        {
          "const": "8222421f",
          "title": "Done",
          "description": "No longer represents the GNode."
        }
      ]
    },
    "StrategyName001": {
      "type": "string",
      "name": "strategy.name.001",
      "description": "Used to assign code to run a particular GNodeInstance",
      "oneOf": [
        {
          "const": "00000000",
          "title": "NoActor",
          "description": "Assigned to GNodes that do not have actors"
        },
        {
          "const": "642c83d3",
          "title": "WorldA",
          "description": "Authority on GNodeInstances, and their private keys. Maintains␣
↪a FastAPI used for relational information about backoffice information held locally␣
↪and/or in the GNodeRegistry/GNodeFactory. [More Info](https://gridworks.readthedocs.io/
↪en/latest/world-role.html)"
        },
        {
          "const": "4bb2cf7e",
          "title": "SupervisorA",
          "description": "A simple supervisor that monitors its supervised AtomicTNode␣
↪GNode strategies via a heartbeat health check. [More Info](https://gridworks.
↪readthedocs.io/en/latest/supervisor.html)"
        },
        {
          "const": "f5961401",
          "title": "AtnHeatPumpWithBoostStore",
          "description": "AtomicTNode for a heat pump thermal storage heating system␣
↪with a boost element and a thermal \n heated by the boost element. [More on␣
↪AtomicTNodes](https://gridworks.readthedocs.io/en/latest/atomic-t-node.html)"
        },
        {
          "const": "73fbe6ab",
```

(continues on next page)

(continued from previous page)

```
          "title": "TcGlobalA",
          "description": "Used to manage the global time of the Gridworks system when␣
↪run with\n in a fully simulated universe. \n [More on TimeCoordinators](https://
↪gridworks.readthedocs.io/en/latest/time-coordinator.html)"
        },
        {
          "const": "5e18a52e",
          "title": "MarketMakerA",
          "description": "Runs a two-sided market associated to its GNode as part of the␣
↪copper GNode sub-tree. [More on MarketMakers](https://gridworks.readthedocs.io/en/
↪latest/market-maker.html)"
        },
        {
          "const": "b2a125d6",
          "title": "AtnBrickStorageHeater",
          "description": "Publicly available Dijkstra-based AtomicTNode strategy for a␣
↪brick storage heater. These heaters are rooom units that store heat in a brick core,␣
↪are heated with resistive elements, and typically have a fan to blow air over the␣
↪brick core. They are sometimes called Electric Thermal Storage (ETS) heaters, and in␣
↪the UK are often called Economy 7 heaters or Night Storage Heaters. A strategy very␣
↪similar to this was used by VCharge to manage an ETS fleet of several thousand heaters␣
↪in Pennsylvania. This strategy is meant to serve as a template for other private␣
↪strategies, and also allows for an end-to-end simulation of a realistic aggregated␣
↪transactive load capable of generating a highly elastic bid curve [More Info](https://
↪gridworks-atn.readthedocs.io/en/latest/brick-storage-heater.html)"
        }
      ]
    }
  },
  "properties": {
    "GNodeInstanceId": {
      "type": "string",
      "format": "UuidCanonicalTextual",
      "title": "Immutable identifier for GNodeInstance (Gni)",
      "required": true
    },
    "GNode": {
      "type": "g.node.gt.002",
      "title": "The GNode represented by the Gni",
      "required": true
    },
    "Strategy": {
      "type": "string",
      "format": "strategy.name.001",
      "title": "Used to determine the code running in a GNode actor application",
      "required": true
    },
    "Status": {
      "type": "string",
      "format": "gni.status.000",
      "title": "Lifecycle Status for Gni",
      "required": true
```

(continues on next page)

(continued from previous page)

```
    },
    "SupervisorContainerId": {
      "type": "string",
      "format": "UuidCanonicalTextual",
      "title": "The Id of the docker container where the Gni runs",
      "required": true
    },
    "StartTimeUnixS": {
      "type": "integer",
      "format": "ReasonableUnixTimeS",
      "title": "When the gni starts representing the GNode",
      "description": "Specifically, when the Status changes from Pending to Active. Note␣
↪that this is time in the GNode's World, which may not be real time if it is a␣
↪simulation.",
      "required": true
    },
    "EndTimeUnixS": {
      "type": "integer",
      "title": "When the gni stops representing the GNode",
      "description": "Specifically, when the Status changes from Active to Done.",
      "required": true
    },
    "AlgoAddress": {
      "type": "string",
      "format": "AlgoAddressStringFormat",
      "title": "Algorand address for Gni",
      "required": false
    },
    "TypeName": {
      "type": "string",
      "value": "g.node.instance.gt",
      "title": "The type name"
    },
    "Version": {
      "type": "string",
      "title": "The type version",
      "default": "000",
      "required": true
    }
  }
}
```

3.11.4 GwCertId


```
{
  "gwapi": "001",
  "type_name": "gw.cert.id",
  "version": "000",
  "owner": "gridworks@gridworks-consulting.com",
  "description": "Clarifies whether cert id is an Algorand Standard Asset or SmartSig",
  "formats": {
    "AlgoAddressStringFormat": {
      "type": "string",
      "description": "String of length 32, characters are all base32 digits.",
      "example": "RNMHG32VTIHTC7W3LZOEPTDGREL5IQGK46HKD3KBLZHYQUCAKLMT4G5ALI"
    }
  },
  "enums": {
    "AlgoCertType000": {
      "type": "string",
      "name": "algo.cert.type.000",
      "description": "Used to distinguish ASA vs SmartSignature certificates",
      "oneOf": [
        {
          "const": "00000000",
          "title": "ASA",
          "description": "Certificate based on Algorand Standard Asset"
        },
        {
          "const": "086b5165",
          "title": "SmartSig",
          "description": "Certificate based on Algorand Smart Signature"
        }
      ]
    }
  },
  "properties": {
    "Type": {
      "type": "string",
      "format": "algo.cert.type.000",
      "title": "",
      "required": true
    },
    "Idx": {
      "type": "integer",
      "minimum": 0,
      "title": "ASA Index",
      "required": false
    },
    "Addr": {
      "type": "string",
      "format": "AlgoAddressStringFormat",
      "title": "Algorand Smart Signature Address",
      "required": false
    },
```


(continues on next page)

(continued from previous page)

```
    "TypeName": {
      "type": "string",
      "value": "gw.cert.id",
      "title": "The type name"
    },
    "Version": {
      "type": "string",
      "title": "The type version",
      "default": "000",
      "required": true
    }
  },
  "example": {
    "TypeGtEnumSymbol": "00000000",
    "Idx": 14,
    "TypeName": "gw.cert.id",
    "Version": "000"
  }
}
```

3.11.5 HeartbeatA

```
{
  "gwapi": "001",
  "type_name": "heartbeat.a",
  "version": "100",
  "owner": "gridworks@gridworks-consulting.com",
  "description": "Used to check that an actor can both send and receive messages.␣
↪Payload for direct messages sent back and forth between actors, for example a␣
↪Supervisor and one of its subordinates.",
  "url": "https://gridworks.readthedocs.io/en/latest/g-node-instance.html",
  "formats": {
    "HexChar": {
      "type": "string",
      "description": "single-char string in '0123456789abcdefABCDEF'",
      "example": "d"
    }
  },
  "properties": {
    "MyHex": {
      "type": "string",
      "format": "HexChar",
      "title": "Hex character getting sent",
      "required": true
    },
    "YourLastHex": {
      "type": "string",
      "format": "HexChar",
      "title": "Last hex character received from heartbeat partner",
      "required": true
```

(continues on next page)

(continued from previous page)

```
    },
    "TypeName": {
      "type": "string",
      "value": "heartbeat.a",
      "title": "The type name"
    },
    "Version": {
      "type": "string",
      "title": "The type version",
      "default": "100",
      "required": true
    }
  }
}
```

3.11.6 Ready

```
{
  "gwapi": "001",
  "type_name": "ready",
  "version": "001",
  "owner": "gridworks@gridworks-consulting.com",
  "description": "Used in simulations by TimeCoordinator GNodes. Only intended for␣
↪simulations that do not have sub-second TimeSteps. TimeCoordinators based on␣
↪```gridworks-timecoordinator``` have a notion of actors whose `Ready` must be received␣
↪before issuing the next TimeStep.",
  "url": "https://gridworks.readthedocs.io/en/latest/time-coordinator.html",
  "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, most␣
↪significant word (on the left) starting with an alphabet character.",
      "example": "dw1.isone.me.freedom.apple"
    }
  },
  "properties": {
    "FromGNodeAlias": {
      "type": "string",
      "format": "LeftRightDot",
      "title": "The GNodeAlias of the sender",
      "required": true
    },
    "FromGNodeInstanceId": {
      "type": "string",
      "format": "UuidCanonicalTextual",
```

(continues on next page)

(continued from previous page)

```
      "title": "The GNodeInstanceId of the sender",
      "required": true
    },
    "TimeUnixS": {
      "type": "integer",
      "title": "Latest simulated time for sender",
      "description": "The time in unix seconds of the latest TimeStep received from the␣
↪TimeCoordinator by the actor that sent the payload.",
      "required": true
    },
    "TypeName": {
      "type": "string",
      "value": "ready",
      "title": "The type name"
    },
    "Version": {
      "type": "string",
      "title": "The type version",
      "default": "001",
      "required": true
    }
  }
}
```

3.11.7 SimTimestep

```
{
  "gwapi": "001",
  "type_name": "sim.timestep",
  "version": "000",
  "owner": "gridworks@gridworks-consulting.com",
  "description": "Sent by TimeCoordinators to coordinate time. For simulated actors,␣
↪time progresses discretely on receipt of these time steps.",
  "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, 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 ms between midnight Jan 1 2000␣
↪and midnight Jan 1 3000 UTC",
      "example": ""
    }
  },
  "properties": {
    "FromGNodeAlias": {
      "type": "string",
      "format": "LeftRightDot",
      "title": "The GNodeAlias of the sender",
      "description": "The sender should always be a GNode Actor of role TimeCoordinator.
↪",
      "required": true
    },
    "FromGNodeInstanceId": {
      "type": "string",
      "format": "UuidCanonicalTextual",
      "title": "The GNodeInstanceId of the sender",
      "required": true
    },
    "TimeUnixS": {
      "type": "integer",
      "format": "ReasonableUnixTimeS",
      "title": "Current time in unix seconds",
      "required": true
    },
    "TimestepCreatedMs": {
      "type": "integer",
      "format": "ReasonableUnixTimeMs",
      "title": "The real time created, in unix milliseconds",
      "required": true
    },
    "MessageId": {
      "type": "string",
      "format": "UuidCanonicalTextual",
      "title": "MessageId",
      "required": true
    },
    "TypeName": {
      "type": "string",
      "value": "sim.timestep",
      "title": "The type name"
    },
    "Version": {
      "type": "string",
      "title": "The type version",
      "default": "000",
      "required": true
    }
  }
```

(continues on next page)

(continued from previous page)

```
}
```

3.11.8 SuperStarter

```
{
  "gwapi": "001",
  "type_name": "super.starter",
  "version": "000",
  "owner": "gridworks@gridworks-consulting.com",
  "description": "Used by world to seed a docker container with data needed to spawn and␣
↪superviser GNodeInstances",
  "formats": {
    "LeftRightDot": {
      "type": "string",
      "description": "Lowercase alphanumeric words separated by periods, most␣
↪significant word (on the left) starting with an alphabet character.",
      "example": "dw1.isone.me.freedom.apple"
    }
  },
  "properties": {
    "SupervisorContainer": {
      "type": "supervisor.container.gt.000",
      "title": "Key data about the docker container",
      "required": true
    },
    "GniList": {
      "type": "g.node.instance.gt.000",
      "title": "List of GNodeInstances (Gnis) run in the container",
      "required": true
    },
    "AliasWithKeyList": {
      "type": "string",
      "format": "LeftRightDot",
      "title": "Aliases of Gnis that own Algorand secret keys",
      "required": true
    },
    "KeyList": {
      "type": "string",
      "title": "Algorand secret keys owned by Gnis",
      "required": true
    },
    "TypeName": {
      "type": "string",
      "value": "super.starter",
      "title": "The type name"
    },
    "Version": {
      "type": "string",
      "title": "The type version",
      "default": "000",
```

(continues on next page)

(continued from previous page)

```
      "required": true
    }
  }
}
```

3.11.9 SupervisorContainerGt

```
{
  "gwapi": "001",
  "type_name": "supervisor.container.gt",
  "version": "000",
  "owner": "gridworks@gridworks-consulting.com",
  "description": "Used to send and receive updates about SupervisorContainers. Sent from␣
↪a GNodeRegistry to a World, and used also by the World as it spawns GNodeInstances in␣
↪docker instances (i.e., the SupervisorContainers).",
  "url": "https://gridworks.readthedocs.io/en/latest/supervisor.html",
  "formats": {
    "WorldInstanceNameFormat": {
      "type": "string",
      "description": "AlphanumericString + '__' + Integer",
      "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, most␣
↪significant word (on the left) starting with an alphabet character.",
      "example": "dw1.isone.me.freedom.apple"
    }
  },
  "enums": {
    "SupervisorContainerStatus000": {
      "type": "string",
      "name": "supervisor.container.status.000",
      "description": "Manages lifecycle of the docker containers where GridWorks actors␣
↪run",
      "oneOf": [
        {
          "const": "00000000",
          "title": "Unknown",
          "description": "Default value"
        },
        {
          "const": "f48cff43",
          "title": "Authorized",
          "description": "World has created the information for starting the container"
```

(continues on next page)

(continued from previous page)

```
        },
        {
          "const": "17c5cc54",
          "title": "Launching",
          "description": "World has launched the container"
        },
        {
          "const": "ec342324",
          "title": "Provisioning",
          "description": "Container has started, but is going through its provisioning␣
↪process"
        },
        {
          "const": "cfde1b40",
          "title": "Running",
          "description": "GNode actors in the container are active"
        },
        {
          "const": "4e28b6ae",
          "title": "Stopped",
          "description": "Stopped"
        },
        {
          "const": "da2dafe0",
          "title": "Deleted",
          "description": "Deleted"
        }
      ]
    }
  },
  "properties": {
    "SupervisorContainerId": {
      "type": "string",
      "format": "UuidCanonicalTextual",
      "title": "Id of the docker SupervisorContainer",
      "required": true
    },
    "Status": {
      "type": "string",
      "format": "supervisor.container.status.000",
      "title": "",
      "required": true
    },
    "WorldInstanceName": {
      "type": "string",
      "format": "WorldInstanceNameFormat",
      "title": "Name of the WorldInstance",
      "description": "For example, d1__1 is a potential name for a World whose World␣
↪GNode has alias d1.",
      "required": true
    },
    "SupervisorGNodeInstanceId": {
```

(continues on next page)

(continued from previous page)

```
      "type": "string",
      "format": "UuidCanonicalTextual",
      "title": "Id of the SupervisorContainer's prime actor (aka the Supervisor GNode)",
      "required": true
    },
    "SupervisorGNodeAlias": {
      "type": "string",
      "format": "LeftRightDot",
      "title": "Alias of the SupervisorContainer's prime actor (aka the Supervisor GNode)
↪",
      "required": true
    },
    "TypeName": {
      "type": "string",
      "value": "supervisor.container.gt",
      "title": "The type name"
    },
    "Version": {
      "type": "string",
      "title": "The type version",
      "default": "000",
      "required": true
    }
  }
}
```

3.12 ActorBase

This class should **not** be initialized directly.

class gridworks.actor_base.**ActorBase**(*settings*, *api_type_maker_by_name={'base.g.node.gt': <class 'gridworks.types.base_g_node_gt.BaseGNodeGt_Maker'>, 'g.node.gt': <class 'gridworks.types.g_node_gt.GNodeGt_Maker'>, 'g.node.instance.gt': <class 'gridworks.types.g_node_instance_gt.GNodeInstanceGt_Maker'>, 'gw.cert.id': <class 'gridworks.types.gw_cert_id.GwCertId_Maker'>, 'heartbeat.a': <class 'gridworks.types.heartbeat_a.HeartbeatA_Maker'>, 'ready': <class 'gridworks.types.ready.Ready_Maker'>, 'sim.timestep': <class 'gridworks.types.sim_timestep.SimTimestep_Maker'>, 'super.starter': <class 'gridworks.types.super_starter.SuperStarter_Maker'>, 'supervisor.container.gt': <class 'gridworks.types.supervisor_container_gt.SupervisorContainerGt_Maker'>}*)

This is the base class for GNodes, used to communicate via RabbitMQ

Parameters

- **settings** (GNodeSettings) –
- **api_type_maker_by_name** (*Dict[str, HeartbeatA_Maker]*) –

acknowledge_message(*delivery_tag*)

Acknowledge the message delivery from RabbitMQ by sending a Basic.Ack RPC method for the delivery tag. :param int delivery_tag: The delivery tag from the Basic.Deliver frame

Return type
None

add_on_cancel_consumer_callback()

Add a callback that will be invoked if RabbitMQ cancels the consumer for some reason. If RabbitMQ does cancel the consumer, on_consumer_cancelled will be invoked by pika.

Return type
None

add_on_consume_channel_close_callback()

This method tells pika to call the on_consumer_channel_closed method if RabbitMQ unexpectedly closes the channel.

Return type
None

add_on_publish_channel_close_callback()

This method tells pika to call the on_channel_closed method if RabbitMQ unexpectedly closes the channel.

Return type
None

close_consumer_channel()

Call to close the channel with RabbitMQ cleanly by issuing the Channel.Close RPC command.

Return type
None

close_publish_channel()

Invoke this command to close the channel with RabbitMQ by sending the Channel.Close RPC command.

Return type
None

close_publish_connection()

This method closes the production connection to RabbitMQ.

Return type
None

connect_consumer()

This method connects to RabbitMQ, returning the connection handle. When the connection is established, the on_consumer_connection_open method will be invoked by pika. :rtype: pika.SelectConnection

Return type
SelectConnection

connect_publisher()

This method connects to RabbitMQ, returning the connection handle. When the connection is established, the on_connection_open method will be invoked by pika. :rtype: pika.SelectConnection

Return type
SelectConnection

from_alias_from_routing_key(*routing_key*)

Returns the GNodeAlias in left-right-dot format. Raises a SchemaError if there is trouble getting this. :param routing_key: This is the basic_deliver.routing_key string :type routing_key: str :param in a rabbit message:

Parameters
routing_key (*str*) –

Return type
str

from_role_from_routing_key(*routing_key*)

Returns the GNodeRole that the message came from. Raises a SchemaError if there is trouble getting this. :param routing_key: basic_deliver.routing_key string :type routing_key: str :param in a rabbit message:

Raises
SchemaError – Error in message construction

Returns
GNodeRole of the actor that sent the message

Return type
GNodeRole

Parameters
routing_key (*str*) –

get_payload_type_name(*basic_deliver*)

The TypeName is a string that provides the strongly typed specification
(API/ABI) for the incoming message. This is similar to knowing the protobuf name/method or the ABI name/method.

The TypeName will articulate, in particular, how to decode the payload.

Parameters

- **basic_deliver** – the rabbit basic_deliver object
- **body** – the rabbit body object (i.e. the payload as incoming type)

Returns
raises SchemaError if the TypeName is not accessible. Otherwise returns the TypeName

Return type
str

heartbeat_from_super(*from_alias*, *ping*)

Subordinate GNode responds to its supervisor's heartbeat with a "pong" message.

Both the received heartbeat (ping) and the response (pong) have the type HeartbeatA (see: https://gridworks.readthedocs.io/en/latest/apis/types.html#heartbeata).

The subordinate GNode generates its own unique identifier (hex) and includes it in the pong message along with the heartbeat it received from the supervisor.

Note that the subordinate GNode does not have the responsibility of verifying the authenticity of the last heartbeat received from the supervisor - although typically, the supervisor does send the last heartbeat from this GNode (except during the initial heartbeat exchange).

Parameters

- **from_alias** (*str*) – the alias of the GNode that sent the ping.
- **ping** (HeartbeatA) – the heartbeat sent.

Return type
None

Raises: ValueError: If *from_alias* is not this GNode's Supervisor alias.

local_rabbit_startup()

This should be overwritten in derived class for any additional rabbit bindings. DO NOT start queues here. It is called at the end of self.start_consuming()

Return type
None

local_start()

This should be overwritten in derived class for additional threads. It cannot assume the rabbit channels are established and that messages can be received or sent.

Return type
None

local_stop()

Join any threads in the derived class.

Return type
None

message_category_from_routing_key(*routing_key*)

Returns the MessageCategory of the message given the routing key.

Raises a SchemaError exception if there is a problem decoding the MessageCategory

Parameters

- **routing_key** (*str*) – This is the basic_deliver.routing_key string
- **message** (*in a rabbit*) –

Returns
the MessageCategory, as enum

Return type
MessageCategory

on_basic_qos_ok(*_unused_frame*)

Invoked by pika when the Basic.QoS method has completed. At this point we will start consuming messages by calling start_consuming which will invoke the needed RPC commands to start the process. :param pika.frame.Method _unused_frame: The Basic.QosOk response frame

Return type
None

on_cancelconsumer_ok(*_unused_frame*, *userdata*)

This method is invoked by pika when RabbitMQ acknowledges the cancellation of a consumer. At this point we will close the channel. This will invoke the on_consumer_channel_closed method once the channel has been closed, which will in-turn close the connection. :param pika.frame.Method _unused_frame: The Basic.CancelOk frame :param str|unicode userdata: Extra user data (consumer tag)

Return type
None

on_consumer_cancelled(*method_frame*)

Invoked by pika when RabbitMQ sends a Basic.Cancel for a consumer receiving messages. :param pika.frame.Method method_frame: The Basic.Cancel frame

Return type
None

on_consumer_channel_closed(*channel*, *reason*)

Invoked by pika when the RabbitMQ channel is unexpectedly closed.

This callback is triggered when a channel is closed, usually due to violating the protocol by attempting to re-declare an exchange or queue with different parameters. In this case, the connection is closed to gracefully shutdown the object.

Parameters

- **channel** (*PikaChannel*) – The closed channel object.
- **reason** (*Exception*) – why the channel was closed

Return type
None

on_consumer_channel_open(*channel*)

Invoked by pika when the channel has been successfully opened.

This callback is triggered when the channel is opened, and it provides the channel object that can be used for further operations. In this case, we'll proceed to declare the exchange to be used.

Parameters
channel (*PikaChannel*) – The opened channel object.

Return type
None

on_consumer_connection_closed(*_unused_connection*, *reason*)

This method is invoked by pika when the connection to RabbitMQ is closed unexpectedly. Since it is unexpected, we will reconnect to RabbitMQ if it disconnects. :param pika.connection.Connection connection: The closed connection obj :param Exception reason: exception representing reason for loss of connection.

Parameters

- **_unused_connection** (*SelectConnection*) –
- **reason** (*Exception*) –

Return type
None

on_consumer_connection_open(*_unused_connection*)

This method is called by pika once the connection to RabbitMQ has been established. It passes the handle to the connection object in case we need it, but in this case, we'll just mark it unused. :param pika.SelectConnection _unused_connection: The connection

Parameters
_unused_connection (*SelectConnection*) –

Return type
None

on_consumer_connection_open_error(*_unused_connection*, *err*)

This method is called by pika if the connection to RabbitMQ can't be established. :param pika.SelectConnection _unused_connection: The connection :param Exception err: The error

Parameters

- **_unused_connection** (*SelectConnection*) –
- **err** (*Exception*) –

Return type
None

on_direct_message_bindok(*_unused_frame*, *binding*)

Invoked by pika when the Queue.Bind method has completed for direct messages. At this point we will set the prefetch count for the channel. :param pika.frame.Method _unused_frame: The Queue.BindOk response frame :param str|unicode userdata: Extra user data (queue name)

Return type
None

on_exchange_declareok(*_unused_frame*, *userdata*)

Invoked by pika when RabbitMQ has finished the Exchange.Declare RPC command. :param pika.Frame.Method unused_frame: Exchange.DeclareOk response frame :param str|unicode userdata: Extra user data (exchange name)

Return type
None

on_message(*_unused_channel*, *basic_deliver*, *properties*, *body*)

Invoked by pika when a message is delivered from RabbitMQ. If a message does not get here that you expect should get here, check the routing key of the outbound message and the rabbitmq bindings.

Parses the TypeName of the message payload and the GNodeAlias of the sender. If it recognizes the GNode and the TypeName, then it sends the message on to the check_routing function, which will be defined in a child class (e.g., the GNodeFactoryActorBase if the actor is a GNodeFactory).

From RabbitMQ: The channel is passed for your convenience. The basic_deliver object that is passed in carries the exchange, delivery tag, and a redelivered flag for the message. The properties passed in is an instance of BasicProperties with the message properties including the routing key. The body is the message that was sent.

Parameters

- **_unused_channel** (*pika.channel.Channel*) – The channel object
- **basic_deliver** (*pika.spec.Basic.Deliver*) – The basic.deliver method
- **properties** (*pika.spec.BasicProperties*) – The message properties including the routing key
- **body** (*bytes*) – The message body

Return type
None

on_publish_channel_closed(*channel*, *reason*)

Invoked by pika when the RabbitMQ channel is unexpectedly closed.

This callback is triggered when a channel is closed, usually due to violating the protocol by attempting to re-declare an exchange or queue with different parameters. In this case, the connection is closed to gracefully shutdown the object.

Parameters

- **channel** (*PikaChannel*) – The closed channel object.

- **reason** (*Exception*) – The reason why the channel was closed.

Return type
None

on_publish_channel_open(*channel*)

Invoked by pika when the channel has been successfully opened.

This callback is triggered when the channel is opened, and it provides the channel object that can be used for further operations. In this case, we'll proceed to declare the exchange to be used.

Parameters
channel (*PikaChannel*) – The opened channel object.

Return type
None

on_publish_connection_closed(*_unused_connection*, *reason*)

This method is invoked by pika when the connection to RabbitMQ is closed unexpectedly. Since it is unexpected, we will reconnect to RabbitMQ if it disconnects. :param pika.connection.Connection connection: The closed connection obj :param Exception reason: exception representing reason for loss of connection.

Return type
None

on_publish_connection_open(*_unused_connection*)

This method is called by pika once the publisher connection to RabbitMQ has been established. It passes the handle to the connection object in case we need it, but in this case, we'll just mark it unused. :param pika.SelectConnection _unused_connection: The connection

Return type
None

on_publish_connection_open_error(*_unused_connection*, *err*)

This method is called by pika if the connection to RabbitMQ can't be established. :param pika.SelectConnection _unused_connection: The connection :param Exception err: The error

Return type
None

on_queue_declareok(*_unused_frame*)

Method invoked by pika when the Queue.Declare RPC call made in setup_queue has completed. In this method we will bind the queue and exchange together with the routing key by issuing the Queue.Bind RPC command. When this command is complete, the on_bindok method will be invoked by pika. :param pika.frame.Method _unused_frame: The Queue.DeclareOk frame :param str|unicode userdata: Extra user data (queue name)

Return type
None

open_consume_channel()

Open a new channel with RabbitMQ by issuing the Channel.Open RPC command. When RabbitMQ responds that the channel is open, the on_channel_open callback will be invoked by pika.

Return type
None

open_publish_channel()

This method will open a new channel with RabbitMQ by issuing the Channel.Open RPC command. When RabbitMQ confirms the channel is open by sending the Channel.OpenOK RPC reply, the on_channel_open method will be invoked.

Return type
None

abstract prepare_for_death()

Use actor_main_stopped to exit out of any threads in the derived class. Then use local_stop to join those threads.

If there are no threads in the derived class, copy this method into the derived class, get rid of the abstract-method decorator, and delete the exception

Return type
None

reconnect_consumer()

Will be invoked if the connection can't be opened or is closed. Indicates that a reconnect is necessary then stops the ioloop.

Return type
None

route_message(*from_alias*, *from_role*, *payload*)

Base class for message routing in GNode Actors, handling interactions with the Supervisor and TimeCoordinator.

Derived classes are expected to implement their own *route_message* method. It is recommended to call *super().route_message(from_alias, from_role, payload)* at the end of the method if the message has not been routed yet.

Parameters

- **from_alias** (*str*) –
- **from_role** (GNodeRole) –
- **payload** (SimTimestep) –

Return type
None

route_mqtt_message(*from_alias*, *payload*)

Base class for message routing from SCADA actors, which use MessageCategory.MqttJsonBroadcast

This is only intended to be used for AtomicTNodes and Ears.

Parameters

- **from_alias** (*str*) –
- **payload** (HeartbeatA) –

Return type
None

run_consumer()

Run the example consumer by connecting to RabbitMQ and then starting the IOLoop to block and allow the SelectConnection to operate.

Return type
None

run_publisher()

Run the example code by connecting and then starting the IOLoop.

Return type
None

send_message(*payload*, *message_category=MessageCategory.RabbitJsonDirect*, *to_role=None*, *to_g_node_alias=None*, *radio_channel=None*)

Publish a direct message to another GNode in the registry world. The only type of direct messages in the registry use json (i.e. no more streamlined serial encoding), unlike in non-registry worlds.

Parameters

- **payload** (HeartbeatA) – Any GridWorks types with a json content-type
- **key** (*that includes TypeName as a json*) –
- **as_type()** (*and has*) –
- **method.** (*as an encoding*) –
- **routing_key_type** – for creating routing key
- **to_role** (*Optional[GNodeRole]*) – used if a direct message
- **to_g_node_alias** (*str*) – used if a direct message
- **message_category** (MessageCategory) –
- **radio_channel** (*str | None*) –

Returns
MESSAGE_SENT with success, otherwise some description of why the message was not sent.

Return type
OnSendMessageDiagnostic

set_qos()

This method sets up the consumer prefetch to only be delivered one message at a time. The consumer must acknowledge this message before RabbitMQ will deliver another one. You should experiment with different prefetch values to achieve desired performance.

Return type
None

setup_exchange()

Setup the exchange on RabbitMQ by invoking the Exchange.Declare RPC command. When it is complete, the on_exchange_declareok method will be invoked by pika. :param str|unicode exchange_name: The name of the exchange to declare

Return type
None

setup_queue()

Setup the queue on RabbitMQ by invoking the Queue.Declare RPC command. When it is complete, the on_queue_declareok method will be invoked by pika. :param str|unicode queue_name: The name of the queue to declare.

Return type
None

start_consuming()

This method sets up the consumer by first calling add_on_cancel_consumer_callback so that the object is notified if RabbitMQ cancels the consumer. It then issues the Basic.Consume RPC command which returns the consumer tag that is used to uniquely identify the consumer with RabbitMQ. We keep the value to use

it when we want to cancel consuming. The on_message method is passed in as a callback pika will invoke when a message is fully received.

Return type
None

`stop_consumer()`

Cleanly shutdown the connection to RabbitMQ by stopping the consumer with RabbitMQ. When RabbitMQ confirms the cancellation, on_cancelconsumer_ok will be invoked by pika, which will then closing the channel and connection. If you want to use this with CTRL-C, figure out how to add back the commented out ioloop.start() below without error.

Return type
None

`stop_consuming()`

Tell RabbitMQ that you would like to stop consuming by sending the Basic.Cancel RPC command.

Return type
None

`stop_publisher()`

Stop the example by closing the channel and connection. We set a flag here so that we stop scheduling new messages to be published. The IOLoop is started because this method is invoked by the Try/Catch below when KeyboardInterrupt is caught. Starting the IOLoop again will allow the publisher to cleanly disconnect from RabbitMQ.

Return type
None

`type_name_from_routing_key(routing_key)`

Returns the TypeName of the message given the routing key. Raises a SchemaError exception if there is a problem decoding the TypeName, or if it does not have the appropriate left-right-dot format.

Parameters

- **routing_key** (*str*) – This is the basic_deliver.routing_key string
- **message** (*in a rabbit*) –

Returns
the TypeName of the payload, in Lrd format

Return type
str

3.13 AlgoUtils

`class gridworks.algo_utils.BasicAccount(private_key=None)`

Representation of the public and private information of an Algorand account, with a few shorthands on top of algosdk.account - in particular x = BasicAccount() will generate a new BasicAccount.

Parameters
private_key (*str | None*) –

`property addr: str`

Shorthand property for the account's public key

property addr_short_hand: str

Returns the last 6 characters of the account address. Used for logging messages and human eyes.

address()

A method returning the account's public key. Equivalent to self.addr

Return type
str

property address_as_bytes: bytes

Useful if digging into encoding and decoding between strings and bytes.

classmethod from_mnemonic(*m*)

Takes the string of 25 words (the mnemonic) and returns the BasicAccount

Parameters
m (*str*) –

Return type
BasicAccount

property mnemonic: str

m stands for *mnemonic*, the string of 25 words that has equivalent information content to the private key

private_key()

A method returning the account's private key. Equivalent to self.sk

Return type
str

property sk: str

Shorthand for the account's private key

class gridworks.algo_utils.**MultisigAccount**(*version*, *threshold*, *addresses*)

Represents exactly the information to create the public address of a multisig account: the version, threshold and the ordered list of public addresses in the account. The MultiSigAccount is not meant to change. Unlike a BasicAccount, it does not store private information and can be entirely public.

This is different from a algosdk.futures.transaction.Multisig, which stores the latest signatures it knows about in its ordered list of MultisigSubsigs. Using the same Multisig for multiple transactions will result in errors.

Parameters

- **version** (*int*) – currently, the version is 1
- **threshold** (*int*) – how many signatures are necessary
- **addresses** (*str[]*) – addresses in the multisig account

version

Type
int

threshold

Type
int

addresses

Type
PublicAddress

property addr: str

Return the MultisigAccount address, again piggybacking on multiSig

property addr_short_hand: str

Returns the last 6 characters of the account address.

This is just used for logging messages and human eyes!

address()

Return the account address, again piggybacking on multiSig

Return type
str

create_mtx(*txn*)

Returns the MultisigTransaction for this MultisigAccount

Parameters
txn (*<module 'algosdk.future.transaction' from '/home/docs/checkouts/readthedocs.org/user_builds/gridworks/envs/latest/lib/python3.10/site-packages/algosdk/future/transaction.py'>*) –

Return type
MultisigTransaction

validate()

Check if the account is valid by piggybacking on multiSig.validate

Return type
None

class gridworks.algo_utils.**PendingTxnResponse**(*tx_id*, *response*)

Parses the dict object of a transaction response into a Python class

Parameters

- **tx_id** (*str*) –
- **response** (*Dict[str, Any]*) –

gridworks.algo_utils.**algos**(*addr*)

Parameters

- **acct** (*AlgoAccount*) – can also be a multiSig
- **addr** (*str*) –

Returns
Return the number of microAlgos in an account, or None if there is an issue getting this number

Return type
Optional[int]

gridworks.algo_utils.**get_app_global_state**(*client*, *app_id*)

Returns the global state of an Algorand application

Parameters

- **client** (*AlgodClient*) – Any AlgodClient
- **app_id** (*int*) – the application id of the smart contract

Returns
Returns the decoded key/value pairs of an app's global state (uints and bytes)

Return type
Dict[bytes, Union[int, bytes]]

gridworks.algo_utils.**get_balances**(*client*, *addr*)

Returns a dictionary of Algorand Standard Asset holdings

Parameters

- **addr** (*str*) – public address holding the ASAs
- **client** (*AlgodClient*) –

Returns
A dictionary taking the ids of the held ASAs to the amounts of those ASAs held by the address.

Return type
Dict[int, int]

gridworks.algo_utils.**micro_algos**(*addr*)

Args: acct (AlgoAccount): can also be a multiSig

Returns
Return the number of microAlgos in an account, or None if there is an issue getting this number

Return type
Optional[int]

Parameters
addr (*str*) –

gridworks.algo_utils.**pay_account**(*client*, *sender*, *to_addr*, *amt_in_micros*)

Sends micro algos from one account to another, and waits to

Parameters

- **client** (*AlgodClient*) – AlgodClient
- **sender** (BasicAccount) – algo_utils.BasicAccount (not algo_utils.MultisigAccount)
- **to_addr** (*str*) – public address receiving the algos
- **amtInMicros** (*int*) – Algos * 10**6 getting sent
- **amt_in_micros** (*int*) –

Raises
errors.AlgoError – raises error if fails to send, if there is a txid discrepancy

Return type
PendingTxnResponse

Returns: PendingTxnResponse

gridworks.algo_utils.**send_signed_mtx**(*client*, *mtx*)

Combines sending a Multisig transaction (sing send_raw_transaction) with waiting for the transaction to complete, returning a python PendingTxnResponse class object with the response.

Parameters

- **client** (*AlgodClient*) –
- **mtx** (*MultisigTransaction*) –

Return type
PendingTxnResponse

gridworks.algo_utils.**string_to_algo_addr**(*input_str*)

Encodes a string as an algorand address. Could be used to overload mutable optional address fields in an asset (freeze, reserve, clawback)

Parameters
input_str (*str*) – Aribtrary string.

Returns
AlgoAddressStringFormat

Return type
str

gridworks.algo_utils.**verify_account_exists_and_funded**(*addr*)

Raises an exception if an address is not a valid format, or if the account does not have at least 1 Algo. Note that every account on Algorand must have a minimum balance of 100,000 microAlgos and this minimum balance increases with more holdings.

Parameters
addr (*str*) – the Algorand public address in question

Raises
errors.AlgoError – if addr does not have AlgoAddressStringFormat, or does not have at least 1 Algo.

Return type
None

gridworks.algo_utils.**wait_for_transaction**(*client*, *tx_id*, *timeout=10*)

Translates algosdk client.pending_transaction_info into a Python class object PendingTxnResponse after waiting for a confirmed transaction

Parameters

- **tx_id** (*str*) – id of transaction
- **timeout** (*int*) – rounds to repeat before raising an error and giving up
- **client** (*AlgodClient*) –

Raises
Exception if transaction is not confirmed in timeout rounds –

Return type
PendingTxnResponse

3.14 ApiUtils

Helpers for working with GridWorks specific Algorand certificates and Smart Contracts. Heavily used in API Type validation.

gridworks.api_utils.**alias_from_deed_idx**(*asset_idx*)

Returns the TerminalAsset's GNodeAlias from the the TaDeed unique identifer (when the AlgoCertType is ASA, and the identifier is an integer). Returns None if the asset_idx is not in fact the identifier for a TaDeed [more info](https://gridworks.readthedocs.io/en/latest/ta-deed.html#tadeed)

Parameters
asset_idx (*int*) – the ASA id in question

Return type
str | None

gridworks.api_utils.**check_mtx_subsig**(*mtx*, *signer_addr*)

Throws a SchemaError if the signer_addr is not a signer for mtx or did not sign. TODO: add error if the signature does not match the txn.

Parameters
mtx (*MultisigTransaction*) –

Return type
None

gridworks.api_utils.**check_validator_multi_has_enough_algos**(*ta_validator_addr*)

Raises exception if the 2-sig multi [GNfAdminAddr, ta_validator_addr] insufficiently funded. Set publicly by the GNodeFactory. [More info](https://gridworks.readthedocs.io/en/latest/g-node-factory.html#tavalidatorfundingthresholdalgos)

Parameters

- **validator_addr** – the public address of the pending validator
- **ta_validator_addr** (*str*) –

Raises
SchemaError if joint account does not have Public(). gnf_validator_funding_threshold_algos. –

Return type
None

gridworks.api_utils.**get_discoverer_account_with_admin**(*discoverer_addr*)

Returns the 2-sig multi [discoverer, gnf_admin_addr] address for a GridWorks Discoverer.

Parameters
discoverer_addr (*str*) – The Algorand address of the Discoverer

Raises
Exception SchemaError – Returned if discoverer_addr has wrong format.

Return type
MultisigAccount

gridworks.api_utils.**get_tadeed_id**(*terminal_asset_alias*, *validator_addr*)

Looks for an asset created in the 2-sig [Gnf Admin, validator_addr] account
that is a tadeed for terminal_asset_alias.

[more info](https://gridworks.readthedocs.io/en/latest/ta-deed.html#tadeed-technical-details) [GwCertId.Type](https://gridworks.readthedocs.io/en/latest/enums.html#gridworks.enums.AlgoCertType)

Parameters

- **terminal_asset_alias** (*str*) – the alias of the Terminal Asset
- **validator_addr** (*str*) – the AlgoAddress of the Validator

Returns

returns None if no TaDeed for this alias was created by the two-sig Multi [GnfAdminAddr, validator_addr]. Otherwise returns the unique identifier of the TaDeed, which is an AlgoAddress string if the GwCertId.Type=SmartSig, and is an integer if the GwCertId.Type=ASA

Return type

Optional[*GwCertId*]

gridworks.api_utils.**get_tatrading_rights_id**(*terminal_asset_alias*)

Returns unique identifier for TaTradingRights [more info](https://gridworks.readthedocs.io/en/latest/ta-trading-rights.html#tatradingrights-technical-details) [GwCertId.Type](https://gridworks.readthedocs.io/en/latest/enums.html#gridworks.enums.AlgoCertType)

Parameters

- **terminal_asset_alias** (*str*) – The GNodeAlias of the TerminalAsset that
- **for** (*the TaTradingRights are*) –

Returns

returns None if no TaTradingRights are found. Otherwise returns the unique identifier of the TaTradingRights, which is an AlgoAddress string if the GwCertId.Type==SmartSig, and is an integer if the GwCertId.Type==ASA

Return type

Optional[*GwCertId*]

gridworks.api_utils.**get_validator_account_with_admin**(*validator_addr*)

Returns the 2-sig multi [gnf_admin_addr, validator_addr] address for a GridWorks Validator.

Parameters

validator_addr (*str*) – The Algorand address of the Discoverer

Raises

Exception SchemaError – Returned if discoverer_addr has wrong format.

Return type

MultisigAccount

gridworks.api_utils.**get_validator_cert_idx**(*validator_addr*)

Looks for an asset in the validatorMsig account that is a validator certificate (based on unit name). [More info](https://gridworks.readthedocs.io/en/latest/ta-validator.html#tavalidator-certificate)

Parameters

validator_addr (*str*) – the public address of the validator (NOT the multi)

Returns

returns None if no Validator Certificate is found, otherwise the unique identifier of the certificate. Since Validator certificates are always ASA, this identifier is an int.

Return type

Optional[int]

gridworks.api_utils.**is_ta_deed**(*asset_idx*)

Returns True if the asset_idx is the unique identifier for a TaDeed certificate. Note that the AlgoCertType must be ASA [more info](https://gridworks.readthedocs.io/en/latest/ta-deed.html#tadeed-technical-details)

Parameters
asset_idx (*int*) – the ASA id in question

Return type
bool

gridworks.api_utils.**is_ta_validator**(*acct_addr*)

Returns True if the account_addr is the TaValidatorAddr for a TaValidator False otherwise [more info](https://gridworks.readthedocs.io/en/latest/ta-validator.html)

Parameters
acct_addr (*str*) –

Return type
bool

gridworks.api_utils.**is_validator_addr**(*validator_addr*)

Checks if Validator Multi has a single valid TaValidator certificate

Parameters
validator_addr (*str*) –

Return type
bool

3.15 AlgoSetup

For use in *sandbox dev* mode.

gridworks.dev_utils.algo_setup.**dev_fund_to_min**(*addr*, *min_algos*)

Tops up the AlgoAddr so that it has at least min_algos Algos, using a randomly chosen genesis acct from the sandbox. Note that the native algosdk pay_account uses micro Algos (in ints).

Returns: PendingTxnResponse

Parameters

- **addr** (*str*) –
- **min_algos** (*int*) –

Return type
PendingTxnResponse

3.16 GwConfig

Settings for the GNodeFactory, readable from environment and/or from env files.

class gridworks.gw_config.**GNodeSettings**(*_env_file='<object object>'*, *_env_file_encoding=None*, *_env_nested_delimiter=None*, *_secrets_dir=None*, ***, *public=Public(gnf_api_root='http://localhost:8000', dev_market_maker_api_root='http://localhost:7997', dev_ta_validator_api_root='http://localhost:8001', gnf_admin_addr='RNMHG32VTIHTC7W3LZOEPTDGREL5IQGK46HKD3KBLZ gnr_addr='X2ASUAUPK5ICMGDXQZQKBPSXWEJLBA4KKQ2TXW2KWO2JQT dev_market_maker_addr='JMEUH2AXM6UGRJO2DBZXDOA2OMIWQFNQZ54 dev_ta_validator_addr='7QQT4GN3ZPAQEFCNWF5BMF7NULVK3CWICZVT4C dev_ta_validator_multi_addr="Y5TRQXIJHWJ4OHCZSWP4PZTCES5VWOF2KD ta_validator_funding_threshold_algos=100, ta_deed_consideration_algos=50, universe='dev', gnf_graveyard_addr='COA6SYUOBE33F5JDYEGC5XAD43QRG3VGHNNQXLYW algod_address='http://localhost:4001', kmd_address='http://localhost:4002', gen_kmd_wallet_name='unencrypted-default-wallet'), algo_api_secrets=AlgoApiSecrets(algod_token=SecretStr('**********'), kmd_token=SecretStr('**********'), gen_kmd_wallet_password=SecretStr('')), rabbit=RabbitBrokerClient(url=SecretStr('**********')), redis_endpoint='localhost', g_node_alias='d1.isone.unknown.gnode', g_node_id='e23eb2ec-4064-4921-89d4-b006edc81216', g_node_instance_id='00000000-0000-0000-0000-000000000000', g_node_role_value='GNode', sk=SecretStr('**********'), universe_type_value='Dev', my_super_alias='d1.super1', my_time_coordinator_alias='d1.time', initial_time_unix_s=1577852400, log_level='INFO', minute_cron_file='cron_last_minute.txt', hour_cron_file='cron_last_hour.txt', day_cron_file='cron_last_day.txt'*)

Template settings for a GNode.

Parameters

- **_env_file** (*str | PathLike | List[str | PathLike] | Tuple[str | PathLike, ...] | None*) –
- **_env_file_encoding** (*str | None*) –
- **_env_nested_delimiter** (*str | None*) –
- **_secrets_dir** (*str | PathLike | None*) –
- **public** (Public) –
- **algo_api_secrets** (*AlgoApiSecrets*) –
- **rabbit** (RabbitBrokerClient) –
- **redis_endpoint** (*str*) –
- **g_node_alias** (*str*) –

- **g_node_id** (*str*) –
- **g_node_instance_id** (*str*) –
- **g_node_role_value** (*str*) –
- **sk** (*SecretStr*) –
- **universe_type_value** (*str*) –
- **my_super_alias** (*str*) –
- **my_time_coordinator_alias** (*str*) –
- **initial_time_unix_s** (*int*) –
- **log_level** (*str*) –
- **minute_cron_file** (*str*) –
- **hour_cron_file** (*str*) –
- **day_cron_file** (*str*) –

classmethod validate_setting_axioms(*values*)

Validates the following:

- universe_type_value belongs to the GridWorks UniverseType enum
- g_node_role_value belongs to the GridWOrks GNodeRole enum
- All GNodeAliases (self, my time coordinator, my super) have the correct

LeftRightDot format and match the alias pattern for the universe type - sk.get_secret_value() has the format of an Algorand secret key - initial_time_unix_s is a reasonable unix time in ms

class gridworks.gw_config.**Public**(*, *gnf_api_root='http://localhost:8000'*,
dev_market_maker_api_root='http://localhost:7997',
dev_ta_validator_api_root='http://localhost:8001',
gnf_admin_addr='RNMHG32VTIHTC7W3LZOEPTDGREL5IQGK46HKD3KBLZHYQUCA
gnr_addr='X2ASUAUPK5ICMGDXQZQKBPSXWEJLBA4KKQ2TXW2KWO2JQTLY3J2Q4
dev_market_maker_addr='JMEUH2AXM6UGRJO2DBZXDOA2OMIWQFNQZ54LCVC4G
dev_ta_validator_addr='7QQT4GN3ZPAQEFCNWF5BMF7NULVK3CWICZVT4GM3BQR
dev_ta_validator_multi_addr="Y5TRQXIJHWJ4OHCZSWP4PZTCES5VWOF2KDTNYSMU
ta_validator_funding_threshold_algos=100,
ta_deed_consideration_algos=50, *universe='dev'*,
gnf_graveyard_addr='COA6SYUOBE33F5JDYEGC5XAD43QRG3VGHNNQXLYWFSSQEH
algod_address='http://localhost:4001',
kmd_address='http://localhost:4002',
gen_kmd_wallet_name='unencrypted-default-wallet')

Publicly available information about the GNodeFactory, including:

- GnfAdminAddr
- GnfApiRoot
- TaValidatorFundingThresholdAlgos
- TaDeedConsiderationAlgos

Also includes useful information shartcuts for running the simulated Millinocket demo. In this demo there is only one MarketMaker, and one TaValidator. Public Algorand addresses and ApiRoots are included for both.

Parameters

- **gnf_api_root** (*str*) –
- **dev_market_maker_api_root** (*str*) –
- **dev_ta_validator_api_root** (*str*) –
- **gnf_admin_addr** (*str*) –
- **gnr_addr** (*str*) –
- **dev_market_maker_addr** (*str*) –
- **dev_ta_validator_addr** (*str*) –
- **dev_ta_validator_multi_addr** (*str*) –
- **ta_validator_funding_threshold_algos** (*int*) –
- **ta_deed_consideration_algos** (*int*) –
- **universe** (*str*) –
- **gnf_graveyard_addr** (*str*) –
- **algod_address** (*str*) –
- **kmd_address** (*str*) –
- **gen_kmd_wallet_name** (*str*) –

class gridworks.gw_config.**RabbitBrokerClient**(*, *url=SecretStr('**********')*)

Settings for connecting to a Rabbit Broker

Parameters

url (*SecretStr*) –

class gridworks.gw_config.**SupervisorSettings**(*_env_file='<object object>'*, *_env_file_encoding=None*, *_env_nested_delimiter=None*, *_secrets_dir=None*, ***, *g_node_alias='d1.isone.ver.keene.super1'*, *g_node_id='664a3250-ce51-4fe3-9ce9-a4b6416451fb'*, *g_node_instance_id='20e7edec-05e5-4152-bfec-ec21ddd2e3dd'*, *supervisor_container_id='995b0334-9940-424f-8fb1-4745e52ba295'*, *g_node_role_value='Supervisor'*, *log_level='INFO'*, *universe_type_value='Dev'*, *world_instance_name='d1__1'*, *rabbit=RabbitBrokerClient(url=SecretStr('**********'))*, *my_time_coordinator_alias='d1.time'*)

Parameters

- **_env_file** (*str | PathLike | List[str | PathLike] | Tuple[str | PathLike, ...] | None*) –
- **_env_file_encoding** (*str | None*) –
- **_env_nested_delimiter** (*str | None*) –
- **_secrets_dir** (*str | PathLike | None*) –
- **g_node_alias** (*str*) –
- **g_node_id** (*str*) –
- **g_node_instance_id** (*str*) –

- **supervisor_container_id** (*str*) –
- **g_node_role_value** (*str*) –
- **log_level** (*str*) –
- **universe_type_value** (*str*) –
- **world_instance_name** (*str*) –
- **rabbit** (RabbitBrokerClient) –
- **my_time_coordinator_alias** (*str*) –

class gridworks.gw_config.**VanillaSettings**(*_env_file='<object object>'*, *_env_file_encoding=None*, *_env_nested_delimiter=None*, *_secrets_dir=None*, ***, *algo_api_secrets=AlgoApiSecrets(algod_token=SecretStr('**********'), kmd_token=SecretStr('**********'), gen_kmd_wallet_password=SecretStr(''))*, *public=Public(gnf_api_root='http://localhost:8000', dev_market_maker_api_root='http://localhost:7997', dev_ta_validator_api_root='http://localhost:8001', gnf_admin_addr='RNMHG32VTIHTC7W3LZOEPTDGREL5IQGK46HKD3KB gnr_addr='X2ASUAUPK5ICMGDXQZQKBPSXWEJLBA4KKQ2TXW2KWO2J dev_market_maker_addr='JMEUH2AXM6UGRJO2DBZXDOA2OMIWQFNQZ dev_ta_validator_addr='7QQT4GN3ZPAQEFCNWF5BMF7NULVK3CWICZVT dev_ta_validator_multi_addr="Y5TRQXIJHWJ4OHCZSWP4PZTCES5VWOF2 ta_validator_funding_threshold_algos=100, ta_deed_consideration_algos=50, universe='dev', gnf_graveyard_addr='COA6SYUOBE33F5JDYEGC5XAD43QRG3VGHNNQXL algod_address='http://localhost:4001', kmd_address='http://localhost:4002', gen_kmd_wallet_name='unencrypted-default-wallet')*)

Parameters

- **_env_file** (*str | PathLike | List[str | PathLike] | Tuple[str | PathLike, ...] | None*) –
- **_env_file_encoding** (*str | None*) –
- **_env_nested_delimiter** (*str | None*) –
- **_secrets_dir** (*str | PathLike | None*) –
- **algo_api_secrets** (*AlgoApiSecrets*) –
- **public** (Public) –

gridworks.gw_config.**check_is_left_right_dot**(*v*)

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

Raises

ValueError – if not LeftRightDot format

Parameters

v (*str*) –

Return type

None

gridworks.gw_config.**check_is_reasonable_unix_time_s**(*v*)

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

Raises
ValueError – if not ReasonableUnixTimeS format

Parameters
v (*int*) –

Return type
None

3.17 GridWorks DataClasses

class gridworks.data_classes.base_g_node.**BaseGNode**(*g_node_id*, **args*, ***kwargs*)

DataClass corresponding to base.g.node.gt type.

The global authority for state of BaseGNodes is managed by the GNodeFactory, and shared with AtomicTNodes from its World Registry. Do not update or create these objects except when starting up and using local data from a file (provided by the World to its docker instance) or in a direct message (Rest POST or rabbit broadcast) from the World.

[more info](https://gridworks.readthedocs.io/en/latest/g-node.html#basegnodes)

children()

Returns the list of BaseGnodes identifying this node as parent

Return type
List[BaseGNode]

is_copper()

Returns true if role is not other

Return type
bool

parent()

Raises: DcError if "natural" parent (as suggested by alias) is not Active, and either

- prev_alias is None, OR
- the parent as suggested by prev_alias is not Active and/or

does not exist.

Returns

BaseGNode. Parent BaseGNode

- If the parent as suggested by the alias exists as an

Active BaseGNode, returns that ("natural" parent)

- Else, if the parent as suggested by the prev_alais exists

as an active BaseGNode, returns that.

None.

- If alias is one word long (i.e. root of world)

Return type
BaseGNode | None

classmethod parent_from_alias(*alias*)

Returns

- BaseGNode. If the parent as suggested by the alias exists as an

Active BaseGNode, returns that. - None. If alias is one word long (i.e. root of world)

Parameters
alias (*str*) –

Return type
BaseGNode | None

class gridworks.data_classes.base_g_node.**BaseGNode**(*g_node_id*, **args*, ***kwargs*)

DataClass corresponding to base.g.node.gt type.

The global authority for state of BaseGNodes is managed by the GNodeFactory, and shared with AtomicTNodes from its World Registry. Do not update or create these objects except when starting up and using local data from a file (provided by the World to its docker instance) or in a direct message (Rest POST or rabbit broadcast) from the World.

[more info](https://gridworks.readthedocs.io/en/latest/g-node.html#basegnodes)

children()

Returns the list of BaseGnodes identifying this node as parent

Return type
List[BaseGNode]

is_copper()

Returns true if role is not other

Return type
bool

parent()

Raises: DcError if "natural" parent (as suggested by alias) is not Active, and either

- prev_alias is None, OR
- the parent as suggested by prev_alias is not Active and/or

does not exist.

Returns

BaseGNode. Parent BaseGNode

- If the parent as suggested by the alias exists as an

Active BaseGNode, returns that ("natural" parent)

- Else, if the parent as suggested by the prev_alais exists

as an active BaseGNode, returns that.

None.

- If alias is one word long (i.e. root of world)

Return type
BaseGNode | None

classmethod parent_from_alias(*alias*)

Returns

- BaseGNode. If the parent as suggested by the alias exists as an

Active BaseGNode, returns that. - None. If alias is one word long (i.e. root of world)

Parameters
alias (*str*) –

Return type
BaseGNode | None

class gridworks.data_classes.g_node.**GNode**(*g_node_id*, **args*, ***kwargs*)

Parameters
g_node_id (*str*) –

Return type
GNode

children()

Returns the list of BaseGnodes identifying this node as parent.

Return type
List[GNode]

parent()

Raises: DcError if "natural" parent (as suggested by alias) is not Active, and either

- prev_alias is None, OR
- the parent as suggested by prev_alias is not Active and/or

does not exist.

Returns

BaseGNode. Parent BaseGNode

- If the parent as suggested by the alias exists as an

Active GNode, returns that ("natural" parent)

- Else, if the parent as suggested by the prev_alais exists

as an active GNode, returns that.

None.

- If alias is one word long (i.e. root of world)

Return type
GNode | None

classmethod parent_from_alias(*alias*)

Returns

- GNode. If the parent as suggested by the alias exists as an

Active BaseGNode, returns that. - None. If alias is one word long (i.e. root of world)

Parameters
alias (*str*) –

Return type
GNode | None

class gridworks.data_classes.g_node_instance.**GNodeInstance**(*g_node_instance_id*, **args*, ***kwargs*)

Parameters
g_node_instance_id (*str*) –

Return type
GNodeInstance

children()

Returns the list of BaseGnodes identifying this node as parent

Return type
List[GNodeInstance]

parent()

Raises: DcError if "natural" parent (as suggested by alias) is not Active, and either

- prev_alias is None, OR
- the parent as suggested by prev_alias is not Active and/or

does not exist.

Returns

BaseGNode. Parent BaseGNode

- If the parent as suggested by the alias exists as an

Active GNode, returns that ("natural" parent)

- Else, if the parent as suggested by the prev_alais exists

as an active GNode, returns that.

None.

- If alias is one word long (i.e. root of world)

Return type
GNodeInstance | None

classmethod **parent_from_alias**(*alias*)

Returns

- GNode. If the parent as suggested by the alias exists as an

Active BaseGNode, returns that. - None. If alias is one word long (i.e. root of world)

Parameters
alias (*str*) –

Return type
GNodeInstance | None

class gridworks.data_classes.g_node_strategy.**GNodeStrategy**(*name*, **args*, ***kwargs*)

Parameters
name (StrategyName) –

Return type
GNodeStrategy

class gridworks.data_classes.market_type.**MarketType**(*name*, **args*, ***kwargs*)

Parameters
name (MarketTypeName) –

Return type
MarketType

class gridworks.data_classes.supervisor_container.**SupervisorContainer**(*supervisor_container_id*, **args*, ***kwargs*)

Parameters
supervisor_container_id (*str*) –

Return type
SupervisorContainer

3.18 GridWorks Enums

GwSchema Enums used in gridworks

class gridworks.enums.**AlgoCertType**(*value*)

Used to distinguish ASA vs SmartSignature certificates

Choices and descriptions:

- ASA: Certificate based on Algorand Standard Asset
- SmartSig: Certificate based on Algorand Smart Signature

classmethod default()

Returns default value ASA

Return type
AlgoCertType

classmethod values()

Returns enum choices

Return type
List[str]

class gridworks.enums.**CoreGNodeRole**(*value*)

CoreGNodeRole assigned by GNodeFactory. [More Info](https://gridworks.readthedocs.io/en/latest/core-g-node-role.html).

Choices and descriptions:

- Other:
- TerminalAsset:
- AtomicTNode:

- MarketMaker:
- AtomicMeteringNode:
- ConductorTopologyNode:
- InterconnectionComponent:
- Scada:

classmethod default()

Returns default value Other

Return type
CoreGNodeRole

classmethod values()

Returns enum choices

Return type
List[str]

class gridworks.enums.**GNodeRole**(*value*)

Categorizes GNodes by their function within GridWorks. [More Info](https://gridworks.readthedocs.io/en/latest/g-node-role.html).

Choices and descriptions:

- GNode: Default value
- TerminalAsset: An avatar for a real-word Transactive Device. [More Info](https://gridworks.readthedocs.io/en/latest/transactive-device.html).
- AtomicTNode: Transacts in markets on behalf of, and controlling the power use of, a TerminalAsset. [More Info](https://gridworks.readthedocs.io/en/latest/atomic-t-node.html).
- MarketMaker: Runs energy markets at its Node in the GNodeTree. [More Info](https://gridworks.readthedocs.io/en/latest/market-maker.html).
- AtomicMeteringNode: Role of a GNode that will become an AtomicTNode, prior to it owning TaTradingRights
- ConductorTopologyNode: An avatar for a real-world electric grid node - e.g. a substation or transformer
- InterconnectionComponent: An avatar for a cable or wire on the electric grid
- World: Adminstrative GNode responsible for managing and authorizing instances. [More Info](https://gridworks.readthedocs.io/en/latest/world-role.html).
- TimeCoordinator: Responsible for managing time in simulations
- Supervisor: Responsible for GNode actors running in a container. [More Info](https://gridworks.readthedocs.io/en/latest/supervisor.html).
- Scada: GNode associated to the device and code that directly monitors and actuates a Transactive Device
- PriceService: Provides price forecasts for markets run by MarketMakers
- WeatherService: Provides weather forecasts
- AggregatedTNode: An aggregation of AtomicTNodes
- Persister: Responsible for acking events with delivery guarantees

classmethod default()

Returns default value GNode

Return type
GNodeRole

classmethod values()

Returns enum choices

Return type
List[str]

class gridworks.enums.**GNodeStatus**(*value*)

Enum for managing GNode lifecycle. [More Info](https://gridworks.readthedocs.io/en/latest/g-node-status.html).

Choices and descriptions:

- Unknown: Default value
- Pending: The GNode exists but cannot be used yet.
- Active: The GNode can be used.
- PermanentlyDeactivated: The GNode can no longer be used, now or in the future.
- Suspended: The GNode cannot be used, but may become active in the future.

classmethod default()

Returns default value Unknown

Return type
GNodeStatus

classmethod values()

Returns enum choices

Return type
List[str]

class gridworks.enums.**GniStatus**(*value*)

Enum for managing GNodeInstance lifecycle. [More Info](https://gridworks.readthedocs.io/en/latest/g-node-instance.html).

Choices and descriptions:

- Unknown: Default Value
- Pending: Has been created by the World, but has not yet finished provisioning
- Active: Active in its GridWorks world. If the GNodeInstance has an actor, that actor is communicating
- Done: No longer represents the GNode.

classmethod default()

Returns default value Unknown

Return type
GniStatus

classmethod values()

Returns enum choices

Return type
List[str]

class gridworks.enums.**MarketPriceUnit**(*value*)

Price unit assigned to MarketMaker MarketType

Choices and descriptions:

- USDPerMWh:

classmethod default()

Returns default value USDPerMWh

Return type
MarketPriceUnit

classmethod values()

Returns enum choices

Return type
List[str]

class gridworks.enums.**MarketQuantityUnit**(*value*)

Quantity unit assigned to MarketMaker MarketType

Choices and descriptions:

- AvgMW:
- AvgkW:

classmethod default()

Returns default value AvgMW

Return type
MarketQuantityUnit

classmethod values()

Returns enum choices

Return type
List[str]

class gridworks.enums.**MarketTypeName**(*value*)

Categorizes different markets run by MarketMaker

Choices and descriptions:

- unknown: Default unknown
- rt5gate5: Real-time energy, 5 minute MarketSlots, gate closing 5 minutes prior to start
- rt60gate5: Real-time energy, 60 minute MarketSlots, gate closing 5 minutes prior to start
- da60: Day-ahead energy, 60 minute MarketSlots
- rt60gate30: Real-time energy, 60 minute MarketSlots, gate closing 30 minutes prior to start
- rt15gate5: Real-time energy, 15 minute MarketSlots, gate closing 5 minutes prior to start
- rt30gate5: Real-time energy, 30 minute MarketSlots, gate closing 5 minutes prior to start
- rt60gate30b: Real-time energy, 30 minute MarketSlots, gate closing 5 minutes prior to start, QuantityUnit AvgkW

classmethod default()

Returns default value unknown

Return type
MarketTypeName

classmethod values()

Returns enum choices

Return type
List[str]

class gridworks.enums.**MessageCategory**(*value*)

Categorizes how GridWorks messages are sent and decoded/encoded

Choices and descriptions:

- Unknown: Default value
- RabbitJsonDirect: Serialized Json message sent on the world rabbit broker from one GNode actor to another
- RabbitJsonBroadcast: Serailized Json message broadcast on the world rabbit broker by a GNode actor
- RabbitGwSerial: GwSerial protocol message sent on the world rabbit broker
- MqttJsonBroadcast: Serialized Json message following MQTT topic format, sent on the world rabbit broker
- RestApiPost: REST API post
- RestApiPostResponse: REST API post response
- RestApiGet: REST API GET

classmethod default()

Returns default value Unknown

Return type
MessageCategory

classmethod values()

Returns enum choices

Return type
List[str]

class gridworks.enums.**MessageCategorySymbol**(*value*)

Shorthand symbols for MessageCategory000 Enum, used in meta-data like routing keys

Choices and descriptions:

- unknown: Default value
- rj: Serialized Json message sent on the world rabbit broker from one GNode actor to another
- rjb: Serailized Json message broadcast on the world rabbit broker by a GNode actor
- s: GwSerial protocol message sent on the world rabbit broker
- gw: Serialized Json message following MQTT topic format, sent on the world rabbit broker
- post: REST API post
- postack: REST API post response
- get: REST API GET

classmethod default()

Returns default value unknown

Return type
MessageCategorySymbol

classmethod values()

Returns enum choices

Return type
List[str]

class gridworks.enums.**RecognizedCurrencyUnit**(*value*)

Unit of currency

Choices and descriptions:

- Unknown:
- USD: US Dollar
- GBP: Pounds sterling

classmethod default()

Returns default value UNKNOWN

Return type
RecognizedCurrencyUnit

classmethod values()

Returns enum choices

Return type
List[str]

class gridworks.enums.**StrategyName**(*value*)

Used to assign code to run a particular GNodeInstance

Choices and descriptions:

- NoActor: Assigned to GNodes that do not have actors
- WorldA: Authority on GNodeInstances, and their private keys. Maintains a FastAPI used for relational information about backoffice information held locally and/or in the GNodeRegistry/GNodeFactory. [More Info](https://gridworks.readthedocs.io/en/latest/world-role.html)
- SupervisorA: A simple supervisor that monitors its supervised AtomicTNode GNode strategies via a heartbeat health check. [More Info](https://gridworks.readthedocs.io/en/latest/supervisor.html)
- AtnHeatPumpWithBoostStore: AtomicTNode for a heat pump thermal storage heating system with a boost element and a thermal

heated by the boost element. [More on AtomicTNodes](https://gridworks.readthedocs.io/en/latest/atomic-t-node.html)

- TcGlobalA: Used to manage the global time of the Gridworks system when run with

in a fully simulated universe. [More on TimeCoordinators](https://gridworks.readthedocs.io/en/latest/time-coordinator.html)

- MarketMakerA: Runs a two-sided market associated to its GNode as part of the copper GNode sub-tree. [More on MarketMakers](https://gridworks.readthedocs.io/en/latest/market-maker.html)
- AtnBrickStorageHeater: Publicly available Dijkstra-based AtomicTNode strategy for a brick storage heater. These heaters are rooom units that store heat in a brick core, are heated with resistive elements, and typically have a fan to blow air over the brick core. They are sometimes called Electric Thermal Storage (ETS) heaters, and in the UK are often called Economy 7 heaters or Night Storage Heaters. A strategy very similar to this was used by VCharge to manage an ETS fleet of several thousand heaters in Pennsylvania. This strategy is meant to serve as a template for other private strategies, and also allows for an end-to-end simulation of a realistic aggregated transactive load capable of generating a highly elastic bid curve [More Info](https://gridworks-atn.readthedocs.io/en/latest/brick-storage-heater.html)

classmethod default()

Returns default value NoActor

Return type
StrategyName

classmethod values()

Returns enum choices

Return type
List[str]

class gridworks.enums.**SupervisorContainerStatus**(*value*)

Manages lifecycle of the docker containers where GridWorks actors run

Choices and descriptions:

- Unknown: Default value
- Authorized: World has created the information for starting the container
- Launching: World has launched the container
- Provisioning: Container has started, but is going through its provisioning process
- Running: GNode actors in the container are active
- Stopped: Stopped
- Deleted: Deleted

classmethod default()

Returns default value Unknown

Return type
SupervisorContainerStatus

classmethod values()

Returns enum choices

Return type
List[str]

class gridworks.enums.**UniverseType**(*value*)

Allows for multiple GridWorks, in particular for development and shared simulations. [More Info](https://gridworks.readthedocs.io/en/latest/universe.html).

Choices and descriptions:

- Dev: Simulation running on a single computer.
- Hybrid: Anything goes.

- Production: Money at stake.

classmethod default()

Returns default value Dev

Return type
UniverseType

classmethod values()

Returns enum choices

Return type
List[str]

3.19 SDK for gridworks Types

The Python classes enumerated below provide an interpretation of GridWorks type instances (serialized JSON) as Python objects. Types are the building blocks for 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 schema types

3.19.1 BaseGNodeGt

Python class corresponding to the GridWorks type base.g.node.gt.002 (VersionedTypeName).

class gridworks.types.**BaseGNodeGt**(*, *GNodeId*, *Alias*, *Status*, *Role*, *GNodeRegistryAddr*, *PrevAlias=None*, *GpsPointId=None*, *OwnershipDeedId=None*, *OwnershipDeedValidatorAddr=None*, *OwnerAddr=None*, *DaemonAddr=None*, *TradingRightsId=None*, *ScadaAlgoAddr=None*, *ScadaCertId=None*, *TypeName='base.g.node.gt'*, *Version='002'*)

.

BaseGNode. Authority is GNodeFactory.

Parameters

- **GNodeId** (*str*) –
- **Alias** (*str*) –
- **Status** (GNodeStatus) –
- **Role** (CoreGNodeRole) –
- **GNodeRegistryAddr** (*str*) –
- **PrevAlias** (*str | None*) –
- **GpsPointId** (*str | None*) –
- **OwnershipDeedId** (*int | None*) –
- **OwnershipDeedValidatorAddr** (*str | None*) –
- **OwnerAddr** (*str | None*) –
- **DaemonAddr** (*str | None*) –
- **TradingRightsId** (*int | None*) –
- **ScadaAlgoAddr** (*str | None*) –

- **ScadaCertId** (*int | None*) –
- **TypeName** (*Literal['base.g.node.gt']*) –
- **Version** (*str*) –

GNodeId:

- Description:
- Format: UuidCanonicalTextual

Alias:

- Description:
- Format: LeftRightDot

Status:

- Description:

Role:

- Description:

GNodeRegistryAddr:

- Description:
- Format: AlgoAddressStringFormat

PrevAlias:

- Description:
- Format: LeftRightDot

GpsPointId:

- Description:
- Format: UuidCanonicalTextual

OwnershipDeedId:

- Description:

OwnershipDeedValidatorAddr:

- Description:
- Format: AlgoAddressStringFormat

OwnerAddr:

- Description:
- Format: AlgoAddressStringFormat

DaemonAddr:

- Description:
- Format: AlgoAddressStringFormat

TradingRightsId:

- Description:

ScadaAlgoAddr:

- Description:
- Format: AlgoAddressStringFormat

ScadaCertId:

- Description:

class gridworks.types.base_g_node_gt.**check_is_uuid_canonical_textual**(*v*)

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

Raises
ValueError – if not UuidCanonicalTextual format

Parameters
v (*str*) –

class gridworks.types.base_g_node_gt.**check_is_left_right_dot**(*v*)

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

Raises
ValueError – if not LeftRightDot format

Parameters
v (*str*) –

class gridworks.types.base_g_node_gt.**check_is_algo_address_string_format**(*v*)

AlgoAddressStringFormat format: The public key of a private/public Ed25519 key pair, transformed into an Algorand address, by adding a 4-byte checksum to the end of the public key and then encoding in base32.

Raises
ValueError – if not AlgoAddressStringFormat format

Parameters
v (*str*) –

class gridworks.types.**BaseGNodeGt_Maker**(*g_node_id*, *alias*, *status*, *role*, *g_node_registry_addr*, *prev_alias*, *gps_point_id*, *ownership_deed_id*, *ownership_deed_validator_addr*, *owner_addr*, *daemon_addr*, *trading_rights_id*, *scada_algo_addr*, *scada_cert_id*)

Parameters

- **g_node_id** (*str*) –
- **alias** (*str*) –
- **status** (GNodeStatus) –
- **role** (CoreGNodeRole) –
- **g_node_registry_addr** (*str*) –
- **prev_alias** (*str | None*) –
- **gps_point_id** (*str | None*) –
- **ownership_deed_id** (*int | None*) –
- **ownership_deed_validator_addr** (*str | None*) –
- **owner_addr** (*str | None*) –
- **daemon_addr** (*str | None*) –

- **trading_rights_id** (*int | None*) –
- **scada_algo_addr** (*str | None*) –
- **scada_cert_id** (*int | None*) –

classmethod tuple_to_type(*tuple*)

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

Parameters
tuple (BaseGNodeGt) –

Return type
str

classmethod type_to_tuple(*t*)

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

Parameters
t (*str*) –

Return type
BaseGNodeGt

3.19.2 GNodeGt

Python class corresponding to the GridWorks type g.node.gt.002 (VersionedTypeName).

class gridworks.types.**GNodeGt**(*, *GNodeId*, *Alias*, *Status*, *Role*, *GNodeRegistryAddr*, *PrevAlias=None*, *GpsPointId=None*, *OwnershipDeedId=None*, *OwnershipDeedValidatorAddr=None*, *OwnerAddr=None*, *DaemonAddr=None*, *TradingRightsId=None*, *ScadaAlgoAddr=None*, *ScadaCertId=None*, *ComponentId=None*, *DisplayName=None*, *TypeName='g.node.gt'*, *Version='002'*)

Used to send and receive updates about GNodes.

GNodes are the building blocks of Gridworks. They have slowly-changing state that must be kept in sync across a distributed system. Therefore, they require a global registry to act as Single Source of Truth (SSoT). This class is used for that SSoT to share information with actors about their GNodes, and the GNodes that they will observe and communicate with. [More info](https://gridworks.readthedocs.io/en/latest/g-node.html).

Parameters

- **GNodeId** (*str*) –
- **Alias** (*str*) –
- **Status** (GNodeStatus) –
- **Role** (GNodeRole) –
- **GNodeRegistryAddr** (*str*) –
- **PrevAlias** (*str | None*) –
- **GpsPointId** (*str | None*) –
- **OwnershipDeedId** (*int | None*) –
- **OwnershipDeedValidatorAddr** (*str | None*) –
- **OwnerAddr** (*str | None*) –

- **DaemonAddr** (*str | None*) –
- **TradingRightsId** (*int | None*) –
- **ScadaAlgoAddr** (*str | None*) –
- **ScadaCertId** (*int | None*) –
- **ComponentId** (*str | None*) –
- **DisplayName** (*str | None*) –
- **TypeName** (*Literal['g.node.gt']*) –
- **Version** (*str*) –

GNodeId:

- Description: Immutable identifier for GNode
- Format: UuidCanonicalTextual

Alias:

- Description: Structured mutable identifier for GNode. The GNode Aliases are used for organizing how actors in Gridworks communicate. Together, they also encode the known topology of the electric grid.
- Format: LeftRightDot

Status:

- Description: Lifecycle indicator

Role:

- Description: Role within Gridworks

GNodeRegistryAddr:

- Description: Algorand address for GNodeRegistry. For actors in a Gridworks world, the GNodeRegistry is the Single Source of Truth for existence and updates to GNodes.
- Format: AlgoAddressStringFormat

PrevAlias:

- Description: Previous GNodeAlias. As the topology of the grid updates, GNodeAliases will change to reflect that. This may happen a handful of times over the life of a GNode.
- Format: LeftRightDot

GpsPointId:

- Description: Lat/lon of GNode. Some GNodes, in particular those acting as avatars for physical devices that are part of or are attached to the electric grid, have physical locations. These locations are used to help validate the grid topology.
- Format: UuidCanonicalTextual

OwnershipDeedId:

- Description: Algorand Id of ASA Deed. The Id of the TaDeed Algorand Standard Asset if the GNode is a TerminalAsset.

OwnershipDeedValidatorAddr:

- Description: Algorand address of Validator. Deeds are issued by the GNodeFactory, in partnership with third party Validators.

- Format: AlgoAddressStringFormat

OwnerAddr:

- Description: Algorand address of the deed owner
- Format: AlgoAddressStringFormat

DaemonAddr:

- Description: Algorand address of the daemon app. Some GNodes have Daemon applications associated to them to handle blockchain operations.
- Format: AlgoAddressStringFormat

TradingRightsId:

- Description: Algorand Id of ASA TradingRights. The Id of the TradingRights Algorand Standard Asset.

ScadaAlgoAddr:

- Description:
- Format: AlgoAddressStringFormat

ScadaCertId:

- Description:

ComponentId:

- Description: Unique identifier for GNode's Component. Used if a GNode is an avatar for a physical device. The serial number of a device is different from its make/model. The ComponentId captures the specific instance of the device.
- Format: UuidCanonicalTextual

DisplayName:

- Description: Display Name

class gridworks.types.g_node_gt.**check_is_uuid_canonical_textual**(*v*)

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

Raises
ValueError – if not UuidCanonicalTextual format

Parameters
v (*str*) –

class gridworks.types.g_node_gt.**check_is_left_right_dot**(*v*)

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

Raises
ValueError – if not LeftRightDot format

Parameters
v (*str*) –

class gridworks.types.g_node_gt.**check_is_algo_address_string_format**(*v*)

AlgoAddressStringFormat format: The public key of a private/public Ed25519 key pair, transformed into an Algorand address, by adding a 4-byte checksum to the end of the public key and then encoding in base32.

Raises
ValueError – if not AlgoAddressStringFormat format

Parameters
v (*str*) –

class gridworks.types.**GNodeGt_Maker**(*g_node_id*, *alias*, *status*, *role*, *g_node_registry_addr*, *prev_alias*, *gps_point_id*, *ownership_deed_id*, *ownership_deed_validator_addr*, *owner_addr*, *daemon_addr*, *trading_rights_id*, *scada_algo_addr*, *scada_cert_id*, *component_id*, *display_name*)

Parameters

- **g_node_id** (*str*) –
- **alias** (*str*) –
- **status** (GNodeStatus) –
- **role** (GNodeRole) –
- **g_node_registry_addr** (*str*) –
- **prev_alias** (*str | None*) –
- **gps_point_id** (*str | None*) –
- **ownership_deed_id** (*int | None*) –
- **ownership_deed_validator_addr** (*str | None*) –
- **owner_addr** (*str | None*) –
- **daemon_addr** (*str | None*) –
- **trading_rights_id** (*int | None*) –
- **scada_algo_addr** (*str | None*) –
- **scada_cert_id** (*int | None*) –
- **component_id** (*str | None*) –
- **display_name** (*str | None*) –

classmethod **tuple_to_type**(*tuple*)

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

Parameters
tuple (GNodeGt) –

Return type
str

classmethod **type_to_tuple**(*t*)

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

Parameters
t (*str*) –

Return type
GNodeGt

3.19.3 GNodeInstanceGt

Python class corresponding to the GridWorks type g.node.instance.gt.000 (VersionedTypeName).

class gridworks.types.**GNodeInstanceGt**(*, *GNodeInstanceId*, *GNode*, *Strategy*, *Status*, *SupervisorContainerId*, *StartTimeUnixS*, *EndTimeUnixS*, *AlgoAddress=None*, *TypeName='g.node.instance.gt'*, *Version='000'*)

Used to send and receive updates about GNodeInstances.

One of the layers of abstraction connecting a GNode with a running app in a Docker container.

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

Parameters

- **GNodeInstanceId** (*str*) –
- **GNode** (GNodeGt) –
- **Strategy** (StrategyName) –
- **Status** (GniStatus) –
- **SupervisorContainerId** (*str*) –
- **StartTimeUnixS** (*int*) –
- **EndTimeUnixS** (*int*) –
- **AlgoAddress** (*str | None*) –
- **TypeName** (*Literal['g.node.instance.gt']*) –
- **Version** (*str*) –

GNodeInstanceId:

- Description: Immutable identifier for GNodeInstance (Gni)
- Format: UuidCanonicalTextual

GNode:

- Description: The GNode represented by the Gni

Strategy:

- Description: Used to determine the code running in a GNode actor application

Status:

- Description: Lifecycle Status for Gni

SupervisorContainerId:

- Description: The Id of the docker container where the Gni runs
- Format: UuidCanonicalTextual

StartTimeUnixS:

- Description: When the gni starts representing the GNode. Specifically, when the Status changes from Pending to Active. Note that this is time in the GNode's World, which may not be real time if it is a simulation.
- Format: ReasonableUnixTimeS

EndTimeUnixS:

- Description: When the gni stops representing the GNode. Specifically, when the Status changes from Active to Done.

AlgoAddress:

- Description: Algorand address for Gni
- Format: AlgoAddressStringFormat

class gridworks.types.g_node_instance_gt.**check_is_reasonable_unix_time_s**(*v*)

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

Raises
ValueError – if not ReasonableUnixTimeS format

Parameters
v (*int*) –

class gridworks.types.g_node_instance_gt.**check_is_uuid_canonical_textual**(*v*)

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

Raises
ValueError – if not UuidCanonicalTextual format

Parameters
v (*str*) –

class gridworks.types.g_node_instance_gt.**check_is_algo_address_string_format**(*v*)

AlgoAddressStringFormat format: The public key of a private/public Ed25519 key pair, transformed into an Algorand address, by adding a 4-byte checksum to the end of the public key and then encoding in base32.

Raises
ValueError – if not AlgoAddressStringFormat format

Parameters
v (*str*) –

class gridworks.types.**GNodeInstanceGt_Maker**(*g_node_instance_id*, *g_node*, *strategy*, *status*, *supervisor_container_id*, *start_time_unix_s*, *end_time_unix_s*, *algo_address*)

Parameters

- **g_node_instance_id** (*str*) –
- **g_node** (GNodeGt) –
- **strategy** (StrategyName) –
- **status** (GniStatus) –
- **supervisor_container_id** (*str*) –
- **start_time_unix_s** (*int*) –
- **end_time_unix_s** (*int*) –
- **algo_address** (*str | None*) –

classmethod tuple_to_type(*tuple*)

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

Parameters
tuple (GNodeInstanceGt) –

Return type
str

classmethod type_to_tuple(*t*)

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

Parameters
t (*str*) –

Return type
GNodeInstanceGt

3.19.4 GwCertId

Python class corresponding to the GridWorks type gw.cert.id.000 (VersionedTypeName).

class gridworks.types.**GwCertId**(*, *Type*, *Idx=None*, *Addr=None*, *TypeName='gw.cert.id'*, *Version='000'*)

Clarifies whether cert id is an Algorand Standard Asset or SmartSig

Parameters

- **Type** (AlgoCertType) –
- **Idx** (*int | None*) –
- **Addr** (*str | None*) –
- **TypeName** (*Literal['gw.cert.id']*) –
- **Version** (*str*) –

classmethod check_axiom_1(*v*)

Axiom 1: Cert type consistency. If Type is ASA, then Id exists and Addr does not. Otherwise, Addr exists and Id does not.

Parameters
v (*dict*) –

Return type
dict

Type:

- Description:

Idx:

- Description: ASA Index

Addr:

- Description: Algorand Smart Signature Address
- Format: AlgoAddressStringFormat

class gridworks.types.gw_cert_id.**check_is_algo_address_string_format**(*v*)

AlgoAddressStringFormat format: The public key of a private/public Ed25519 key pair, transformed into an Algorand address, by adding a 4-byte checksum to the end of the public key and then encoding in base32.

Raises
ValueError – if not AlgoAddressStringFormat format

Parameters
v (*str*) –

class gridworks.types.**GwCertId_Maker**(*type*, *idx*, *addr*)

Parameters

- **type** (AlgoCertType) –
- **idx** (*int | None*) –
- **addr** (*str | None*) –

classmethod tuple_to_type(*tuple*)

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

Parameters
tuple (GwCertId) –

Return type
str

classmethod type_to_tuple(*t*)

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

Parameters
t (*str*) –

Return type
GwCertId

3.19.5 HeartbeatA

Python class corresponding to the GridWorks type heartbeat.a.100 (VersionedTypeName).

class gridworks.types.**HeartbeatA**(*, *MyHex='0'*, *YourLastHex='0'*, *TypeName='heartbeat.a'*, *Version='100'*)

Used to check that an actor can both send and receive messages.

Payload for direct messages sent back and forth between actors, for example a Supervisor and one of its subordinates.

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

Parameters

- **MyHex** (*str*) –
- **YourLastHex** (*str*) –
- **TypeName** (*Literal['heartbeat.a']*) –
- **Version** (*str*) –

MyHex:

- Description: Hex character getting sent
- Format: HexChar

YourLastHex:

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

class gridworks.types.heartbeat_a.**check_is_hex_char**(*v*)

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

Raises
ValueError – if not HexChar format

Parameters
v (*str*) –

class gridworks.types.**HeartbeatA_Maker**(*my_hex*, *your_last_hex*)

Parameters

- **my_hex** (*str*) –
- **your_last_hex** (*str*) –

classmethod **tuple_to_type**(*tuple*)

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

Parameters
tuple (HeartbeatA) –

Return type
str

classmethod **type_to_tuple**(*t*)

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

Parameters
t (*str*) –

Return type
HeartbeatA

3.19.6 Ready

Python class corresponding to the GridWorks type ready.001 (VersionedTypeName).

class gridworks.types.**Ready**(***, *FromGNodeAlias*, *FromGNodeInstanceId*, *TimeUnixS*, *TypeName='ready'*, *Version='001'*)

Used in simulations by TimeCoordinator GNodes.

Only intended for simulations that do not have sub-second TimeSteps. TimeCoordinators based on `gridworks-timecoordinator` have a notion of actors whose *Ready* must be received before issuing the next TimeStep. [More info](https://gridworks.readthedocs.io/en/latest/time-coordinator.html).

Parameters

- **FromGNodeAlias** (*str*) –
- **FromGNodeInstanceId** (*str*) –
- **TimeUnixS** (*int*) –
- **TypeName** (*Literal['ready']*) –
- **Version** (*str*) –

FromGNodeAlias:

- Description: The GNodeAlias of the sender

- Format: LeftRightDot

FromGNodeInstanceId:

- Description: The GNodeInstanceId of the sender
- Format: UuidCanonicalTextual

TimeUnixS:

- Description: Latest simulated time for sender. The time in unix seconds of the latest TimeStep received from the TimeCoordinator by the actor that sent the payload.

class gridworks.types.ready.**check_is_uuid_canonical_textual**(*v*)

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

Raises
ValueError – if not UuidCanonicalTextual format

Parameters
v (*str*) –

class gridworks.types.ready.**check_is_left_right_dot**(*v*)

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

Raises
ValueError – if not LeftRightDot format

Parameters
v (*str*) –

class gridworks.types.**Ready_Maker**(*from_g_node_alias*, *from_g_node_instance_id*, *time_unix_s*)

Parameters

- **from_g_node_alias** (*str*) –
- **from_g_node_instance_id** (*str*) –
- **time_unix_s** (*int*) –

classmethod **tuple_to_type**(*tuple*)

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

Parameters
tuple (Ready) –

Return type
str

classmethod **type_to_tuple**(*t*)

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

Parameters
t (*str*) –

Return type
Ready

3.19.7 SimTimestep

Python class corresponding to the GridWorks type sim.timestep.000 (VersionedTypeName).

class gridworks.types.**SimTimestep**(*, *FromGNodeAlias*, *FromGNodeInstanceId*, *TimeUnixS*, *TimestepCreatedMs*, *MessageId*, *TypeName='sim.timestep'*, *Version='000'*)

Sent by TimeCoordinators to coordinate time.

For simulated actors, time progresses discretely on receipt of these time steps.

Parameters

- **FromGNodeAlias** (*str*) –
- **FromGNodeInstanceId** (*str*) –
- **TimeUnixS** (*int*) –
- **TimestepCreatedMs** (*int*) –
- **MessageId** (*str*) –
- **TypeName** (*Literal['sim.timestep']*) –
- **Version** (*str*) –

FromGNodeAlias:

- Description: The GNodeAlias of the sender. The sender should always be a GNode Actor of role TimeCoordinator.
- Format: LeftRightDot

FromGNodeInstanceId:

- Description: The GNodeInstanceId of the sender
- Format: UuidCanonicalTextual

TimeUnixS:

- Description: Current time in unix seconds
- Format: ReasonableUnixTimeS

TimestepCreatedMs:

- Description: The real time created, in unix milliseconds
- Format: ReasonableUnixTimeMs

MessageId:

- Description: MessageId
- Format: UuidCanonicalTextual

class gridworks.types.sim_timestep.**check_is_reasonable_unix_time_s**(*v*)

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

Raises

ValueError – if not ReasonableUnixTimeS format

Parameters

v (*int*) –

class gridworks.types.sim_timestep.**check_is_uuid_canonical_textual**(*v*)

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

Raises
ValueError – if not UuidCanonicalTextual format

Parameters
v (*str*) –

class gridworks.types.sim_timestep.**check_is_left_right_dot**(*v*)

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

Raises
ValueError – if not LeftRightDot format

Parameters
v (*str*) –

class gridworks.types.sim_timestep.**check_is_reasonable_unix_time_ms**(*v*)

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

Raises
ValueError – if not ReasonableUnixTimeMs format

Parameters
v (*str*) –

class gridworks.types.**SimTimestep_Maker**(*from_g_node_alias*, *from_g_node_instance_id*, *time_unix_s*, *timestep_created_ms*, *message_id*)

Parameters

- **from_g_node_alias** (*str*) –
- **from_g_node_instance_id** (*str*) –
- **time_unix_s** (*int*) –
- **timestep_created_ms** (*int*) –
- **message_id** (*str*) –

classmethod **tuple_to_type**(*tuple*)

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

Parameters
tuple (SimTimestep) –

Return type
str

classmethod **type_to_tuple**(*t*)

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

Parameters
t (*str*) –

Return type
SimTimestep

3.19.8 SuperStarter

Python class corresponding to the GridWorks type super.starter.000 (VersionedTypeName).

class gridworks.types.**SuperStarter**(*, *SupervisorContainer*, *GniList*, *AliasWithKeyList*, *KeyList*, *TypeName='super.starter'*, *Version='000'*)

Used by world to seed a docker container with data needed to spawn and superviser GNodeInstances

Parameters

- **SupervisorContainer** (SupervisorContainerGt) –
- **GniList** (*List[GNodeInstanceGt]*) –
- **AliasWithKeyList** (*List[str]*) –
- **KeyList** (*List[str]*) –
- **TypeName** (*Literal['super.starter']*) –
- **Version** (*str*) –

SupervisorContainer:

- Description: Key data about the docker container

GniList:

- Description: List of GNodeInstances (Gnis) run in the container

AliasWithKeyList:

- Description: Aliases of Gnis that own Algorand secret keys
- Format: LeftRightDot

KeyList:

- Description: Algorand secret keys owned by Gnis

class gridworks.types.super_starter.**check_is_left_right_dot**(*v*)

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

Raises

ValueError – if not LeftRightDot format

Parameters

v (*str*) –

class gridworks.types.**SuperStarter_Maker**(*supervisor_container*, *gni_list*, *alias_with_key_list*, *key_list*)

Parameters

- **supervisor_container** (SupervisorContainerGt) –
- **gni_list** (*List[GNodeInstanceGt]*) –
- **alias_with_key_list** (*List[str]*) –
- **key_list** (*List[str]*) –

classmethod **tuple_to_type**(*tuple*)

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

Parameters

tuple (SuperStarter) –

Return type
str

classmethod type_to_tuple(*t*)

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

Parameters
t (*str*) –

Return type
SuperStarter

3.19.9 SupervisorContainerGt

Python class corresponding to the GridWorks type supervisor.container.gt.000 (VersionedTypeName).

class gridworks.types.**SupervisorContainerGt**(*, *SupervisorContainerId*, *Status*, *WorldInstanceName*, *SupervisorGNodeInstanceId*, *SupervisorGNodeAlias*, *TypeName='supervisor.container.gt'*, *Version='000'*)

Used to send and receive updates about SupervisorContainers.

Sent from a GNodeRegistry to a World, and used also by the World as it spawns GNodeInstances in docker instances (i.e., the SupervisorContainers). [More info](https://gridworks.readthedocs.io/en/latest/supervisor.html).

Parameters

- **SupervisorContainerId** (*str*) –
- **Status** (SupervisorContainerStatus) –
- **WorldInstanceName** (*str*) –
- **SupervisorGNodeInstanceId** (*str*) –
- **SupervisorGNodeAlias** (*str*) –
- **TypeName** (*Literal['supervisor.container.gt']*) –
- **Version** (*str*) –

SupervisorContainerId:

- Description: Id of the docker SupervisorContainer
- Format: UuidCanonicalTextual

Status:

- Description:

WorldInstanceName:

- Description: Name of the WorldInstance. For example, d1__1 is a potential name for a World whose World GNode has alias d1.
- Format: WorldInstanceNameFormat

SupervisorGNodeInstanceId:

- Description: Id of the SupervisorContainer's prime actor (aka the Supervisor GNode)
- Format: UuidCanonicalTextual

SupervisorGNodeAlias:

- Description: Alias of the SupervisorContainer's prime actor (aka the Supervisor GNode)
- Format: LeftRightDot

class gridworks.types.supervisor_container_gt.**check_is_world_instance_name_format**(*v*)

WorldInstanceName format: A single alphanumerical word starting with an alphabet char (the root GNodeAlias) and an integer, seperated by '__'. For example 'd1__1'

Raises
ValueError – if not WorldInstanceNameFormat format

Parameters
v (*str*) –

class gridworks.types.supervisor_container_gt.**check_is_uuid_canonical_textual**(*v*)

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

Raises
ValueError – if not UuidCanonicalTextual format

Parameters
v (*str*) –

class gridworks.types.supervisor_container_gt.**check_is_left_right_dot**(*v*)

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

Raises
ValueError – if not LeftRightDot format

Parameters
v (*str*) –

class gridworks.types.**SupervisorContainerGt_Maker**(*supervisor_container_id*, *status*, *world_instance_name*, *supervisor_g_node_instance_id*, *supervisor_g_node_alias*)

Parameters

- **supervisor_container_id** (*str*) –
- **status** (SupervisorContainerStatus) –
- **world_instance_name** (*str*) –
- **supervisor_g_node_instance_id** (*str*) –
- **supervisor_g_node_alias** (*str*) –

classmethod tuple_to_type(*tuple*)

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

Parameters
tuple (SupervisorContainerGt) –

Return type
str

classmethod type_to_tuple(*t*)

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

Parameters
t (*str*) –

Return type
SupervisorContainerGt

3.20 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*

3.20.1 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.

3.20.2 How to request a feature

Request features on the Issue Tracker.

3.20.3 How to set up your development environment

You need Python 3.10+ 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
```

3.20.4 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.

3.20.5 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.
- 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.

3.21 GridWorks Energy Consulting Code of Conduct

3.21.1 Basic Truth

All humans are worthy.

3.21.2 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.

3.21.3 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.

3.21.4 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

3.21.5 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

3.21.6 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.

3.21.7 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.

3.22 License

PYTHON MODULE INDEX

g

INDEX

D

F

G

H

I

L

M

O

P

R

S

T

U

V

W