VOLTTRON Documentation

Release 7.0 Release Candidate

The VOLTTRON Community

Nov 04, 2020

Introduction

branch- mysql_fix

VOLTTRON™ is an open-source platform for distributed sensing and control. The platform is an open source tool for performing simulations, improving building system performance, and creating a more flexible and reliable power grid.

CHAPTER 1

Features

- a secure *message bus* allowing connectivity between modules on individual platforms and between platform instances in large scale deployments
- a flexible *agent framework* allowing users to adapt the platform to their unique use-cases
- a configurable *driver framework* for collecting data from and sending control signals to buildings and devices
- automatic data capture and retrieval through our *historian framework*
- an extensible *web framework* allowing users and services to securely connect to the platform from anywhere

VOLTTRON™ is open source and publicly available from GitHub. The project is supported by the U.S. Department of Energy and receives ongoing updates from a team of core developers at PNNL. The VOLTTRON team encourages and appreciates community involvement including issues and pull requests on Github, meetings at our bi-weekly office-hours and on Slack. To be invited to office-hours or slack, please send the team an email.

1.1 What is VOLTTRON?

VOLTTRON™ is a software platform on which software modules called "agents" and device driver modules to connect to a message bus to interact. Users may configure included drivers for industry standard device communication protocols such as BACnet or Modbus, or develop and configure their own. Additionally, agents can be installed or developed to perform a vast variety of tasks.

1.1.1 Design Philosophy

VOLTTRON was designed by Pacific Northwest National Laboratory to service building efficiency, building-grid integration and transactive controls systems. These systems are working to improve energy efficiency and resiliency in critical infrastructure. To this end, VOLTTRON was built with the following pillars in mind:

- Cost-Effectiveness - Open source software (free to users) and can be hosted on inexpensive computing resources
- Scalability - Can be used in one building or a fleet of buildings

- Interoperability - Enables interaction/connection with various systems and subsystems, in and out of the energy sector
- Security - Underpinned with a robust security foundation to combat today's cyber vulnerabilities and attacks

1.1.2 Basic Components

- *Message bus* - The VOLTTRON message bus uses message queueing software to exchange messages between agents and drivers installed on the platform. VOLTTRON messages are exchanged using a *publish/suscriber paradigm*, or messages can be routed to specific agents through the bus using *remote procedure calls*.
- *Agents* - Agents are software modules which autonomously perform a set of desired functions on behalf of a user. VOLTTRON agents are often use to collect data, send control signals to devices, implement control algorithms or perform simulations.
- *Drivers* - Drivers can be installed on the platform and configured to communicate with industrial or Internet of Things devices. Drivers provide a set of pre-defined functions which can be mapped to device communication methods to read or set values on the device.
- *Historians* - Historians are special purpose agents which are used to subscribe to sources broadcasting on the message bus and store their messages for later use.
- *Web Framework* - The VOLTTRON web framework

1.2 How Does it Work?

The VOLTTRON platform is built around the concept of software agents. Software agents perform autonomous functions on behalf of a user. The VOLTTRON platform was created to allow a suite of agents installed by a user to work together to achieve the user's goals.

1.2.1 Major Components

The platform comprises several components that allow agents to operate and connect to the platform.

- The *Message Bus* is central to the platform. All other VOLTTRON components communicate through it using *VOLTTRON Interconnect Protocol* (VIP). VIP implements the publish/subscribe paradigm over a variety of topics or directed communication using *Remote Procedure Calls*.
- *Agents* on the platform extend the base agent which provides a VIP connection to the message bus and an agent lifecycle. Agents subscribe to topics which allow it to read. The agent lifecycle is controlled by the Agent Instantiation and Packaging (AIP) component which launches agents in an agent execution environment.
- The *Master Driver Agent* can be configured with a number of driver configurations and will spawn corresponding driver instances. Each driver instance provides functions for collecting device data and setting values on the device. These functions implement device protocol or remote communication endpoint interfaces. Driver data is published to the message bus or if requested by an agent will be delivered in an RPC response.
- Agents can control devices by interacting with the *Actuator Agent* to schedule and send commands.
- The *Historian* framework subscribes to data published on the messages bus and stores it to a database or file, or sends it to another location.

1.2.2 Usability Components

Usability components exist to enhance the base capabilities of the platform for deployments.

- *VOLTTRON Control* is the command line interface to controlling a platform instance. VOLTTRON Control can be used to operate agents, configure drivers, get status and health details, etc.
- Data collection, command and control can be achieved in large deployments by *connecting multiple platform instances*.
- *VOLTTRON Central* is an agent which can be installed on a platform to provide a single management interface to multiple VOLTTRON platform instances.
- JSON, static and websocket endpoints can be registered to agents via the *Web Framework* and platform web server. This allows remote agent communication as well as for agents to serve web pages.

1.3 Installing the Platform

VOLTTRON is written in Python 3.6+ and runs on Linux Operating Systems. For users unfamiliar with those technologies, the following resources are recommended:

- Python 3.6 Tutorial
- Linux Tutorial

This guide will specify commands to use to successfully install the platform on supported Linux distributions, but a working knowledge of Linux will be helpful for troubleshooting and may improve your ability to get more out of your deployment.

Note: Volttron version 7.0rc1 is currently tested for Ubuntu versions 18.04 and 18.10 as well as Linux Mint version 19.3. Version 6.x is tested for Ubuntu versions 16.04 and 18.04 as well as Linux Mint version 19.1.

1.3.1 Step 1 - Install prerequisites

The following packages will need to be installed on the system:

- git
- build-essential
- python3.6-dev
- python3.6-venv
- openssl
- libssl-dev
- libevent-dev

On **Debian-based systems**, these can all be installed with the following command:

```
sudo apt-get update
sudo apt-get install build-essential python3-dev python3-venv openssl libssl-dev␣
↪libevent-dev git
```

On Ubuntu-based systems, available packages allow you to specify the Python3 version, 3.6 or greater is required (Debian itself does not provide those packages).

```
sudo apt-get install build-essential python3.6-dev python3.6-venv openssl libssl-dev␣
↪libevent-dev git
```

On arm-based systems (including, but not limited to, Raspbian), you must also install libffi-dev, you can do this with:

```
sudo apt-get install libffi-dev
```

Note: On arm-based systems, the available apt package repositories for Raspbian versions older than buster (10) do not seem to be able to be fully satisfied. While it may be possible to resolve these dependencies by building from source, the only recommended usage pattern for VOLTTRON 7 and beyond is on raspberry pi OS 10 or newer.

On **Redhat or CENTOS systems**, these can all be installed with the following command:

```
sudo yum update
sudo yum install make automake gcc gcc-c++ kernel-devel python3-devel openssl openssl-
↪devel libevent-devel git
```

Warning: Python 3.6 or greater is required, please ensure you have installed a supported version with `python3 --version`

If you have an agent which requires the pyodbc package, install the following additional requirements:

- freetds-bin
- unixodbc-dev

On **Debian-based systems** these can be installed with the following command:

```
sudo apt-get install freetds-bin  unixodbc-dev
```

On **Redhat or CentOS systems**, these can be installed from the Extra Packages for Enterprise Linux (EPEL) repository:

```
sudo yum install https://dl.fedoraproject.org/pub/epel/epel-release-latest-8.noarch.
↪rpm
sudo yum install freetds unixODBC-devel
```

Note: The above command to install the EPEL repository is for Centos/Redhat 8. Change the number to match your OS version. EPEL packages are included in Fedora repositories, so installing EPEL is not required on Fedora.

It may be possible to deploy VOLTTRON on a system not listed above but may involve some troubleshooting and dependency management on the part of the user.

1.3.2 Step 2 - Clone VOLTTRON code

Repository Structure

There are several options for using the VOLTTRON code depending on whether you require the most stable version of the code or want the latest updates as they happen. In order of decreasing stability and increasing currency:

- *Master* - Most stable release branch, current major release is 7.0. This branch is default.

- *develop* - contains the latest *finished* features as they are developed. When all features are stable, this branch will be merged into *Master*.

  Note: This branch can be cloned by those wanting to work from the latest version of the platform but should not be used in deployments.

- Features are developed on "feature" branches or developers' forks of the main repository. It is not recommended to clone these branches except for exploring a new feature.

Note: VOLTTRON versions 6.0 and newer support two message buses - ZMQ and RabbitMQ.

```
git clone https://github.com/VOLTTRON/volttron --branch <branch name>
```

1.3.3 Step 3 - Setup virtual environment

The *bootstrap.py* script in the VOLTTRON root directory will create a virtual environment and install the package's Python dependencies. Options exist for upgrading or rebuilding existing environments, and for adding additional dependencies for optional drivers and agents included in the repository.

Note: The `--help` option for *bootstrap.py* can specified to display all available optional parameters.

Steps for ZeroMQ

Run the following command to install all required packages:

```
cd <volttron clone directory>
python3 bootstrap.py
```

Then activate the Python virtual environment:

```
source env/bin/activate
```

Proceed to step 4.

Note: You can deactivate the environment at any time by running *deactivate*.

Steps for RabbitMQ

Step 1 - Install Erlang packages

For RabbitMQ based VOLTTRON, some of the RabbitMQ specific software packages have to be installed.

On Debian based systems and CentOS 6/7

If you are running a Debian or CentOS system, you can install the RabbitMQ dependencies by running the "rabbit_dependencies.sh" script, passing in the OS name and appropriate distribution as parameters. The following are supported:

- *debian bionic* (for Ubuntu 18.04)
- *debian xenial* (for Ubuntu 16.04 or Linux Mint 18.04)
- *debian stretch* (for Debian Stretch)
- *debian buster* (for Debian Buster)
- *raspbian buster* (for Raspbian/Raspberry Pi OS Buster)

Example command:

```
./scripts/rabbit_dependencies.sh debian xenial
```

Alternatively

You can download and install Erlang from [Erlang Solutions](https://www.erlang-solutions.com/resources/download.html). Please include OTP/components - ssl, public_key, asn1, and crypto. Also lock your version of Erlang using the [yum-plugin-versionlock](https://access.redhat.com/solutions/98873)

Note:

Currently VOLTTRON only officially supports specific versions of Erlang for each operating system:

- 1:22.1.8.1-1 for Debian
- 1:21.2.6+dfsg-1 for Raspbian
- Specific Erlang 21.x versions correspond to CentOS versions 6, 7, and 8, these can be found here

Step 2 - Configure hostname

Make sure that your hostname is correctly configured in /etc/hosts. See (<https://stackoverflow.com/questions/24797947/os-x-and-rabbitmq-error-epmd-error-for-host-xxx-address-cannot-connect-to-ho>). If you are testing with VMs make please make sure to provide unique host names for each of the VMs you are using.

The hostname should be resolvable to a valid IP when running on bridged mode. RabbitMQ checks for this during initial boot. Without this (for example, when running on a VM in NAT mode) RabbitMQ start-up would fail with the error "unable to connect to empd (port 4369) on <hostname>."

Note: RabbitMQ startup error would show up in the VM's syslog (/var/log/messages) file and not in RabbitMQ logs (/var/log/rabbitmq/rabbitmq@hostname.log)

Step 3 - Bootstrap the environment

```
cd volttron
python3 bootstrap.py --rabbitmq [optional install directory. defaults to <user_home>/
↪rabbitmq_server]
```

This will build the platform and create a virtual Python environment and dependencies for RabbitMQ. It also installs RabbitMQ server as the current user. If an install path is provided, that path should exist and the user should have write permissions. RabbitMQ will be installed under *<install dir>/rabbitmq_server-3.7.7*. The rest of the documentation refers to the directory *<install dir>/rabbitmq_server-3.7.7* as *$RABBITMQ_HOME*.

Note: There are many additional *options for bootstrap.py* for including dependencies, altering the environment, etc.

You can check if the RabbitMQ server is installed by checking its status:

```
service rabbitmq status
```

Note: The *RABBITMQ_HOME* environment variable can be set in ~/.bashrc. If doing so, it needs to be set to the RabbitMQ installation directory (default path is *<user_home>/rabbitmq_server/rabbitmq_server-3.7.7*)

```
echo 'export RABBITMQ_HOME=$HOME/rabbitmq_server/rabbitmq_server-3.7.7'|sudo tee --
↪append ~/.bashrc
source ~/.bashrc
$RABBITMQ_HOME/sbin/rabbitmqctl status
```

Step 4 - Activate the environment

```
source env/bin/activate
```

Note: You can deactivate the environment at any time by running `deactivate`.

Step 5 - Configure RabbitMQ setup for VOLTTRON

```
vcfg --rabbitmq single [optional path to rabbitmq_config.yml]
```

Refer to examples/configurations/rabbitmq/rabbitmq_config.yml for a sample configuration file. At a minimum you will need to provide the host name and a unique common-name (under certificate-data) in the configuration file.

Note: common-name must be unique and the general convention is to use *<volttron instance name>-root-ca*.

Running the above command without the optional configuration file parameter will cause the user user to be prompted for all the required data in the command prompt. "vcfg" will use that data to generate a rabbitmq_config.yml file in the *VOLTTRON_HOME* directory.

Note: If the above configuration file is being used as a basis for creating your own configuration file, be sure to update it with the hostname of the deployment (this should be the fully qualified domain name of the system).

This script creates a new virtual host and creates SSL certificates needed for this VOLTTRON instance. These certificates get created under the subdirectory "certificates" in your VOLTTRON home (typically in ~/.volttron). It then creates the main VIP exchange named "volttron" to route message between the platform and agents and alternate exchange to capture unrouteable messages.

Note: We configure the RabbitMQ instance for a single volttron_home and volttron_instance. This script will confirm with the user the volttron_home to be configured. The VOLTTRON instance name will be read from volttron_home/config if available, if not the user will be prompted for VOLTTRON instance name. To run the scripts without any prompts, save the the VOLTTRON instance name in volttron_home/config file and pass the VOLTTRON home directory as a command line argument. For example: *vcfg –vhome /home/vdev/.new_vhome –rabbitmq single*

The Following are the example inputs for *vcfg –rabbitmq single* command. Since no config file is passed the script prompts for necessary details.

```
Your VOLTTRON_HOME currently set to: /home/vdev/new_vhome2

Is this the volttron you are attempting to setup?  [Y]:
Creating rmq config yml
RabbitMQ server home: [/home/vdev/rabbitmq_server/rabbitmq_server-3.7.7]:
Fully qualified domain name of the system: [cs_cbox.pnl.gov]:

Enable SSL Authentication: [Y]:

Please enter the following details for root CA certificates
Country: [US]:
State: Washington
Location: Richland
Organization: PNNL
Organization Unit: Volttron-Team
Common Name: [volttron1-root-ca]:
Do you want to use default values for RabbitMQ home, ports, and virtual host: [Y]: N
Name of the virtual host under which RabbitMQ VOLTTRON will be running: [volttron]:
AMQP port for RabbitMQ: [5672]:
http port for the RabbitMQ management plugin: [15672]:
AMQPS (SSL) port RabbitMQ address: [5671]:
https port for the RabbitMQ management plugin: [15671]:
INFO:rmq_setup.pyc:Starting rabbitmq server
Warning: PID file not written; -detached was passed.
INFO:rmq_setup.pyc:**Started rmq server at /home/vdev/rabbitmq_server/rabbitmq_server-
↪3.7.7
INFO:requests.packages.urllib3.connectionpool:Starting new HTTP connection (1):␣
↪localhost
INFO:requests.packages.urllib3.connectionpool:Starting new HTTP connection (1):␣
↪localhost
INFO:requests.packages.urllib3.connectionpool:Starting new HTTP connection (1):␣
↪localhost
INFO:rmq_setup.pyc:
Checking for CA certificate

INFO:rmq_setup.pyc:
Root CA (/home/vdev/new_vhome2/certificates/certs/volttron1-root-ca.crt) NOT Found.␣
↪Creating root ca for volttron instance
```

(continues on next page)

(continued from previous page)

```
Created CA cert
INFO:requests.packages.urllib3.connectionpool:Starting new HTTP connection (1):␣
↪localhost
INFO:requests.packages.urllib3.connectionpool:Starting new HTTP connection (1):␣
↪localhost
INFO:rmq_setup.pyc:**Stopped rmq server
Warning: PID file not written; -detached was passed.
INFO:rmq_setup.pyc:**Started rmq server at /home/vdev/rabbitmq_server/rabbitmq_server-
↪3.7.7
INFO:rmq_setup.pyc:

######################

Setup complete for volttron home /home/vdev/new_vhome2 with instance name=volttron1
Notes:

-   Please set environment variable `VOLTTRON_HOME` to `/home/vdev/new_vhome2` before␣
↪starting volttron

-   On production environments, restrict write access to
    /home/vdev/new_vhome2/certificates/certs/volttron1-root-ca.crt to only admin user.
↪ For example: sudo chown root /home/vdev/new_vhome2/certificates/certs/volttron1-
↪root-ca.crt

-   A new admin user was created with user name: volttron1-admin and password=default_
↪passwd.
    You could change this user's password by logging into <https://cs_cbox.pnl.
↪gov:15671/> Please update /home/vdev/new_vhome2/rabbitmq_config.yml if you change␣
↪password

######################
```

1.3.4 Test the VOLTTRON Deployment

We are now ready to start VOLTTRON instance. If configured with RabbitMQ message bus a config file would have been generated in *$VOLTTRON_HOME/config* with the entry `message-bus=rmq`. If you need to revert back to ZeroMQ based VOLTTRON, you will have to either remove the `message-bus` parameter or set it to the default "zmq" in *$VOLTTRON_HOME/config*.

The following command starts volttron process in the background:

```
volttron -vv -l volttron.log&
```

This enters the virtual Python environment and then starts the platform in debug (vv) mode with a log file named volttron.log. Alternatively you can use the utility script start-volttron script that does the same.

```
./start-volttron
```

To stop the platform, use the *vct* command:

```
volttron-ctl shutdown --platform
```

or use the included *stop-volttron* script:

```
./stop-volttron
```

Warning: If you plan on running VOLTTRON in the background and detaching it from the terminal with the `disown` command be sure to redirect stderr and stdout to `/dev/null`. Some libraries which VOLTTRON relies on output directly to stdout and stderr. This will cause problems if those file descriptors are not redirected to `/dev/null`

```
#To start the platform in the background and redirect stderr and stdout
#to /dev/null
volttron -vv -l volttron.log > /dev/null 2>&1&
```

Installing and Running Agents

VOLTTRON platform comes with several built in services and example agents out of the box. To install a agent use the script *install-agent.py*

```
python scripts/install-agent.py -s <top most folder of the agent> [-c <config file.␣
↪Might be optional for some agents>]
```

For example, we can use the command to install and start the Listener Agent - a simple agent that periodically publishes heartbeat message and listens to everything on the message bus. Install and start the Listener agent using the following command:

```
python scripts/install-agent.py -s examples/ListenerAgent --start
```

Check volttron.log to ensure that the listener agent is publishing heartbeat messages.

```
tail volttron.log
```

```
2016-10-17 18:17:52,245 (listeneragent-3.2 11367) listener.agent INFO: Peer: 'pubsub',
↪ Sender: 'listeneragent-3.2_1':, Bus: u'', Topic: 'heartbeat/listeneragent-3.2_1',␣
↪Headers: {'Date': '2016-10-18T01:17:52.239724+00:00', 'max_compatible_version': u'',
↪ 'min_compatible_version': '3.0'}, Message: {'status': 'GOOD', 'last_updated':
↪'2016-10-18T01:17:47.232972+00:00', 'context': 'hello'}
```

You can also use the *volttron-ctl* (or *vctl*) command to start, stop or check the status of an agent

```
(volttron)volttron@volttron1:~/git/rmq_volttron$ vctl status
  AGENT                  IDENTITY             TAG           STATUS          HEALTH
6 listeneragent-3.2      listeneragent-3.2_1                running [13125] GOOD
f master_driveragent-3.2 platform.driver      master_driver
```

```
vctl stop <agent id>
```

Note: The default working directory is ~/.volttron. The default directory for creation of agent packages is *~/.volttron/packaged*

1.3.5 Next Steps

There are several walk-throughs and detailed explanations of platform features to explore additional aspects of the platform:

- *Agent Framework*
- *Driver Framework*
- Demonstration of the *management UI*
- *RabbitMQ setup* with Federation and Shovel plugins

1.4 Definition of Terms

This page lays out a common terminology for discussing the components and underlying technologies used by the platform. The first section discusses capabilities and industry standards that VOLTTRON conforms to while the latter is specific to the VOLTTRON domain.

1.4.1 Industry Terms

- **BACNet**: Building Automation and Control network, that leverages ASHRAE, ANSI, and IOS 16484-5 standard protocols
- **JSON-RPC**: JSON-encoded Remote Procedure Call
- **JSON**: JavaScript object notation is a text-based, human-readable, open data interchange format, similar to XML, but less verbose
- **Modbus**: Communications protocol for talking with industrial electronic devices
- **Publish/subscribe**: A message delivery pattern where senders (publishers) and receivers (subscribers) do not communicate directly nor necessarily have knowledge of each other, but instead exchange messages through an intermediary based on a mutual class or topic
- **RabbitMQ**:
- **SSH**: Secure shell is a network protocol providing encryption and authentication of data using public-key cryptography
- **SSL**: Secure sockets layer is a technology for encryption and authentication of network traffic based on a chain of trust
- **TLS**: Transport layer security is the successor to SSL
- **ZeroMQ or ØMQ**: A library used for inter-process and inter-computer communication

1.4.2 VOLTTRON Terms

Activated Environment

An activated environment is the environment a VOLTTRON instance is run in. The bootstrap process creates the environment from the shell and to activate it the following command is executed.

```
user@computer> source env/bin/activate

# Note once the above command has been run the prompt will have changed
(volttron)user@computer>
```

AIP

Agent Instantiation and Packaging - this is the module responsible for creating agent wheels, the agent execution environment and running agents. Found in the VOLTTRON repository in the *volttron/platform* directory.

Bootstrap Environment

The process by which an operating environment (activated environment) is produced. From the *VOLTTRON_ROOT* directory executing *python bootstrap.py* will start the bootstrap process.

VOLTTRON_HOME

The location for a specific *VOLTTRON_INSTANCE* to store its specific information. There can be many VOLTTRON_HOMEs on a single computing resource(VM, machine, etc.), and each *VOLTTRON_HOME* will correspond to a single instance of VOLTTRON.

VOLTTRON_INSTANCE

A single volttron process executing instructions on a computing resource. For each VOLTTRON_INSTANCE there WILL BE only one *VOLTTRON_HOME* associated with it. In order for a VOLTTRON_INSTANCE to be able to participate outside its computing resource it must be bound to an external ip address.

VOLTTRON_ROOT

The cloned directory from Github. When executing the command

```
git clone http://github.com/VOLTTRON/volttron
```

the top level volttron folder is the VOLTTRON_ROOT

VIP

VOLTTRON Interconnect Protocol is a secure routing protocol that facilitates communications between agents, controllers, services and the supervisory *VOLTTRON_INSTANCE*.

1.5 License

1.5.1 Terms

1.6 Join the Community

The VOLTTRON project is transitioning into the Eclipse Foundation as Eclipse VOLTTRON. Current resources will still be used during this time. Please watch this space!

The Eclipse VOLTTRON team aims to work with users and contributors to continuously improve the platform with features requested by the community as well as architectural features that improve robustness, security, and scalability. Contributing back to the project, which is encouraged but not required, enhances its capabilities for the whole community. To learn more, check out *Contributing* and *Documentation*.

1.6.1 Slack Channel

volttron-community.slack.com is where the VOLTTRON™ community at large can ask questions and meet with others using VOLTTRON™. To be added to Slack please email the VOLTTRON team at volttron@pnnl.gov.

1.6.2 Mailing List

Join the mailing list by emailing volttron@pnnl.gov.

1.6.3 Stack Overflow

The VOLTTRON community supports questions being asked and answered through Stack Overflow. The questions tagged with the *volttron* tag can be found at http://stackoverflow.com/questions/tagged/volttron.

1.6.4 Office Hours

PNNL hosts office hours every other week on Fridays at 11 AM (PST). These meetings are designed to be very informal where VOLTTRON developers can answer specific questions about the inner workings of VOLTTRON. These meetings are also available for topical discussions of different aspects of the VOLTTRON platform. Currently the office hours are available through a Zoom meeting. To be invited to the link meeting, contact the volttron team via email: mailto:volttron@pnnl.gov

Meetings are recorded and can be reviewed here.

1.7 Setting Up a Development Environment

An example development environment used by the VOLTTRON team would consist of a Linux VM running on the host development machine on which an IDE would be running. The guides can be used to set up a development environment.

1.7.1 Forking the Repository

The first step to editing the repository is to fork it into your own user space. Creating a fork makes a copy of the repository in your GitHub for you to make any changes you may require for your use-case. This allows you to make changes without impacting the core VOLTTRON repository.

Forking is done by pointing your favorite web browser to http://github.com/VOLTTRON/volttron and then clicking "Fork" on the upper right of the screen. (Note: You must have a GitHub account to fork the repository. If you don't have one, we encourage you to sign up.)

Note: After making changes to your repository, you may wish to contribute your changes back to the Core VOLTTRON repository. Instructions for contributing code may be found *here*.

Cloning 'YOUR' VOLTTRON forked repository

The next step in the process is to copy your forked repository onto your computer to work on. This will create an identical copy of the GitHub repository on your local machine. To do this you need to know the address of your repository. The URL to your repository address will be `https://github.com/<YOUR USERNAME>/volttron.git`. From a terminal execute the following commands:

```
# Here, we are assuming you are doing develop work in a folder called `git`. If you'd␣
↪rather use something else, that's OK.
mkdir -p ~/git
cd ~/git
git clone -b develop https://github.com/<YOUR USERNAME>/volttron.git
cd volttron
```

Note: VOLTTRON uses develop as its main development branch rather than the standard master branch (the default).

Adding and Committing files

Now that you have your repository cloned, it's time to start doing some modifications. Using a simple text editor you can create or modify any file in the volttron directory. After making a modification or creating a file it is time to move it to the stage for review before committing to the local repository. For this example let's assume we have made a change to *README.md* in the root of the volttron directory and added a new file called *foo.py*. To get those files in the staging area (preparing for committing to the local repository) we would execute the following commands:

```
git add foo.py
git add README.md

# Alternatively in one command
git add foo.py README.md
```

After adding the files to the stage you can review the staged files by executing:

```
git status
```

Finally, in order to commit to the local repository we need to think of what change we actually did and be able to document it. We do that with a commit message (the -m parameter) such as the following.

```
git commit -m "Added new foo.py and updated copyright of README.md"
```

Pushing to the remote repository

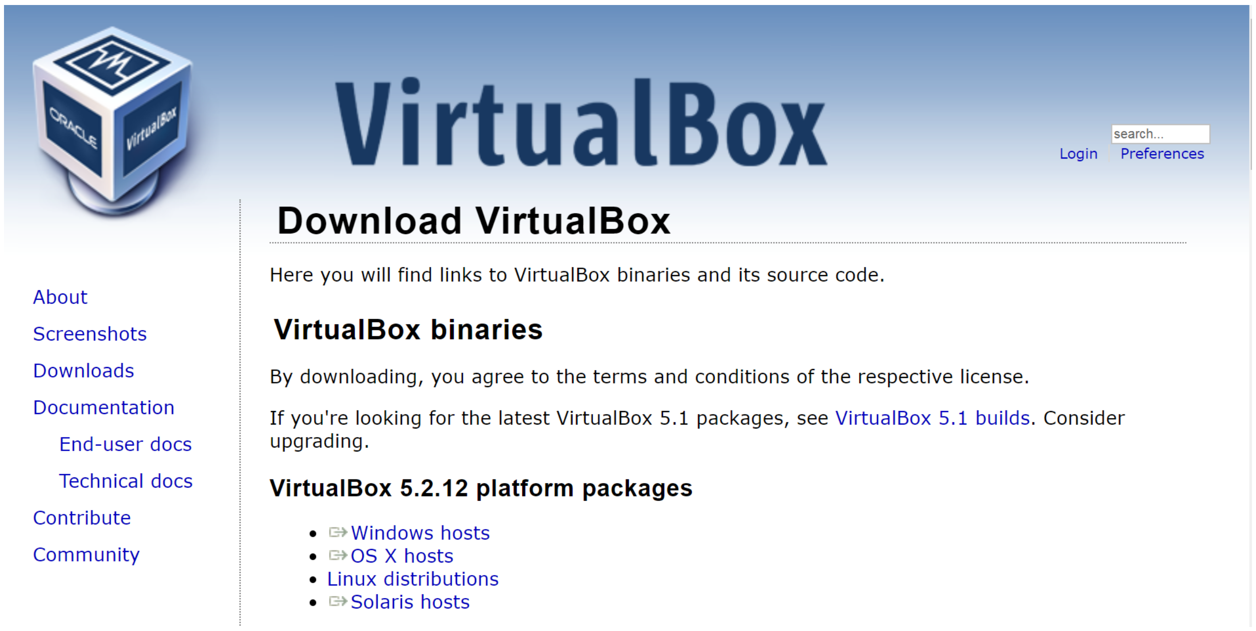
The next step is to share our changes with the world through GitHub. We can do this by pushing the commits from your local repository out to your GitHub repository. This is done by the following command:

```
git push
```

1.7.2 Installing a Linux Virtual Machine

VOLTTRON requires a Linux system to run. For Windows users this will require a virtual machine (VM).

This section describes the steps necessary to install VOLTTRON using Oracle VirtualBox software. Virtual Box is free and can be downloaded from https://www.virtualbox.org/wiki/Downloads.



Download VirtualBox

Here you will find links to VirtualBox binaries and its source code.

VirtualBox binaries

By downloading, you agree to the terms and conditions of the respective license.

If you're looking for the latest VirtualBox 5.1 packages, see VirtualBox 5.1 builds. Consider upgrading.

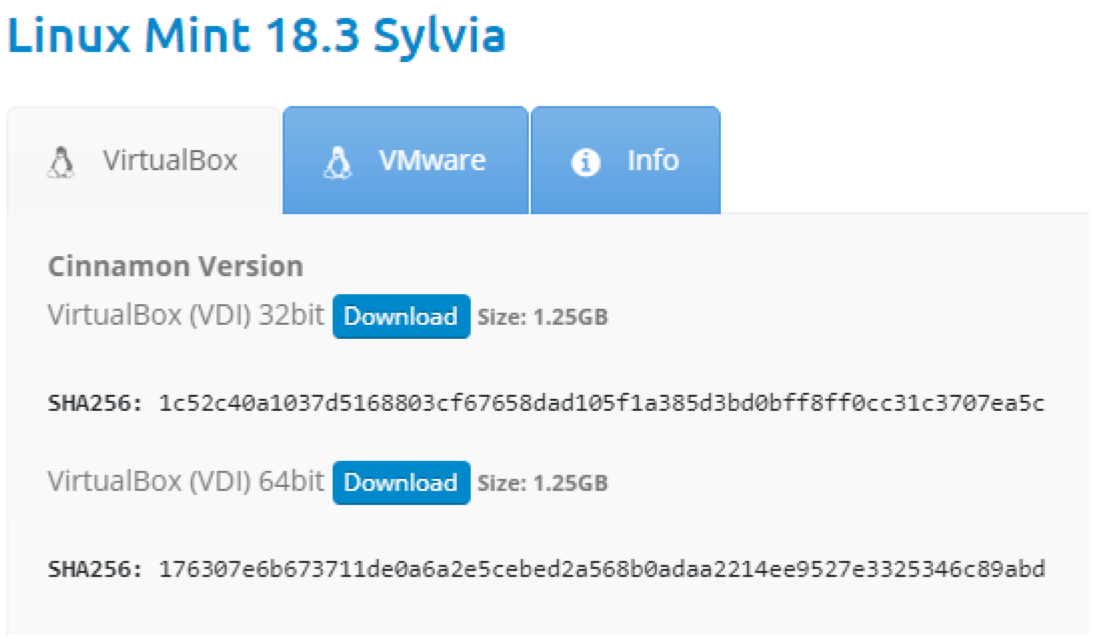
VirtualBox 5.2.12 platform packages

- Windows hosts
- OS X hosts
- Linux distributions
- Solaris hosts

After installing VirtualBox download a virtual box appliance from https://www.osboxes.org/linux-mint/ extract the VDI from the downlaoded archive, **or** download a system installation disk. VOLTTRON version 7.0.x has been tested using Ubuntu 18.04, 18.10; Linux Mint 19; VOLTTRON version 6.0.x has been tested with Ubuntu 16.04, 18.04. However, any modern apt based Linux distribution should work out of the box. Linux Mint 19.3 with the Xfce desktop is used as an example, however platform setup in Ubuntu should be identical.

Note: A 32-bit version of Linux should be used when running VOLTTRON on a system with limited hardware (less than 2 GB of RAM).

Adding a VDI Image to VirtualBox Environment



Cinnamon Version

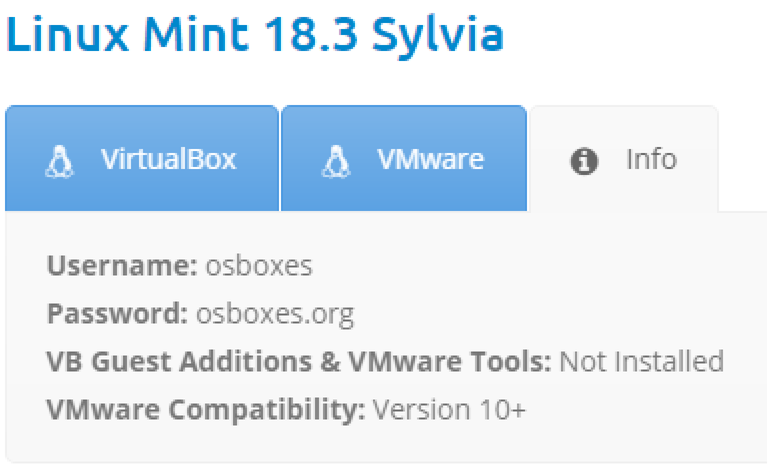
VirtualBox (VDI) 32bit Size: 1.25GB

SHA256: 1c52c40a1037d5168803cf67658dad105f1a385d3bd0bff8ff0cc31c3707ea5c

VirtualBox (VDI) 64bit Size: 1.25GB

SHA256: 176307e6b673711de0a6a2e5cebed2a568b0adaa2214ee9527e3325346c89abd

The below info holds the VM's preset username and password.


Linux Mint 18.3 Sylvia

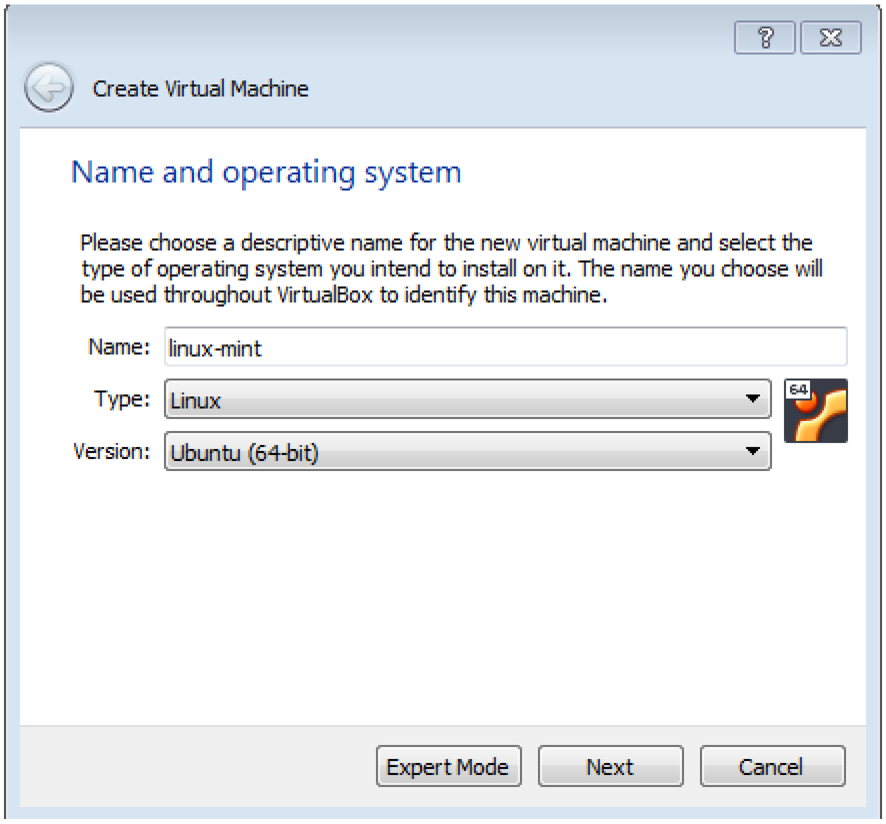
Username: osboxes

Password: osboxes.org

VB Guest Additions & VMware Tools: Not Installed

VMware Compatibility: Version 10+

Create a new VirtualBox Image.



Name and operating system

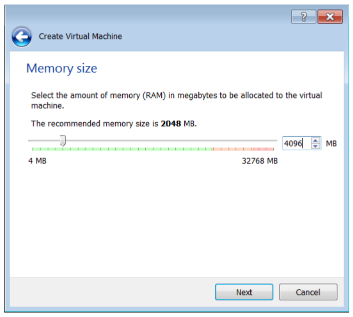
Please choose a descriptive name for the new virtual machine and select the type of operating system you intend to install on it. The name you choose will be used throughout VirtualBox to identify this machine.

Name: linux-mint

Type: Linux

Version: Ubuntu (64-bit)

Select the amount of RAM for the VM. The recommended minimum is shown in the image below:

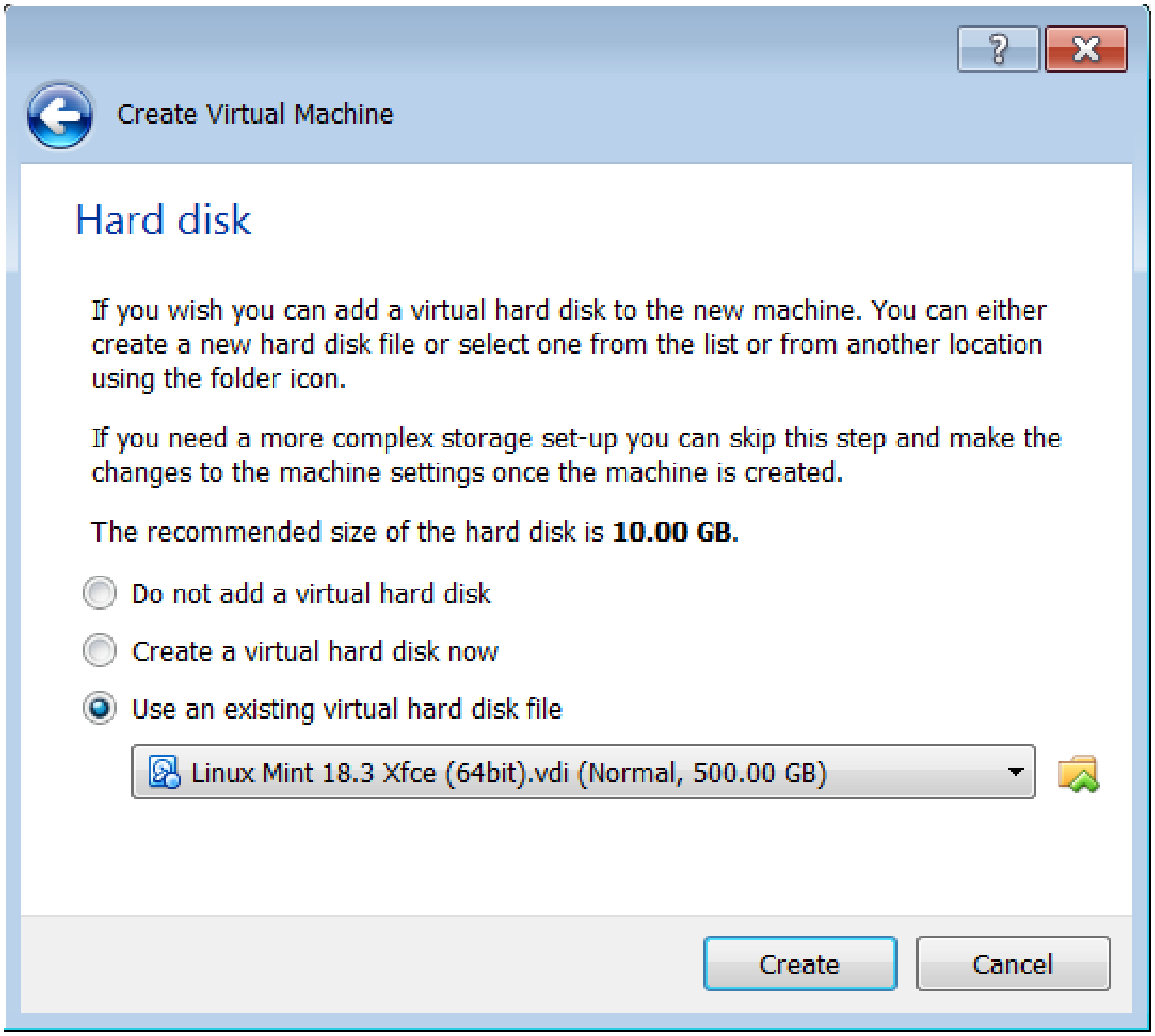


Select the amount of memory (RAM) in megabytes to be allocated to the virtual machine.

The recommended memory size is **2048** MB.

4096 MB

Specify the hard drive image using the extracted VDI file.



If you wish you can add a virtual hard disk to the new machine. You can either create a new hard disk file or select one from the list or from another location using the folder icon.

If you need a more complex storage set-up you can skip this step and make the changes to the machine settings once the machine is created.

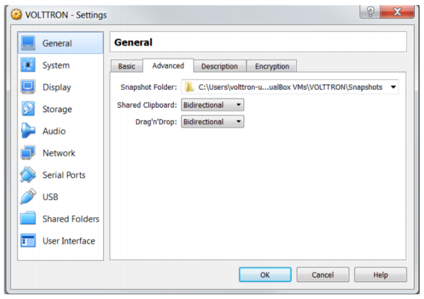
The recommended size of the hard disk is **10.00 GB.**

- Do not add a virtual hard disk
- Create a virtual hard disk now
- [✓ Use an existing virtual hard disk file]

Linux Mint 18.3 Xfce (64bit).vdi (Normal, 500.00 GB)

With the newly created VM selected, choose Machine from the VirtualBox menu in the top left corner of the VirtualBox window; from the drop down menu, choose Settings.

To enable bidirectional copy and paste, select the General tab in the VirtualBox Settings. Enable Shared Clipboard and Drag'n'Drop as Bidirectional.



General

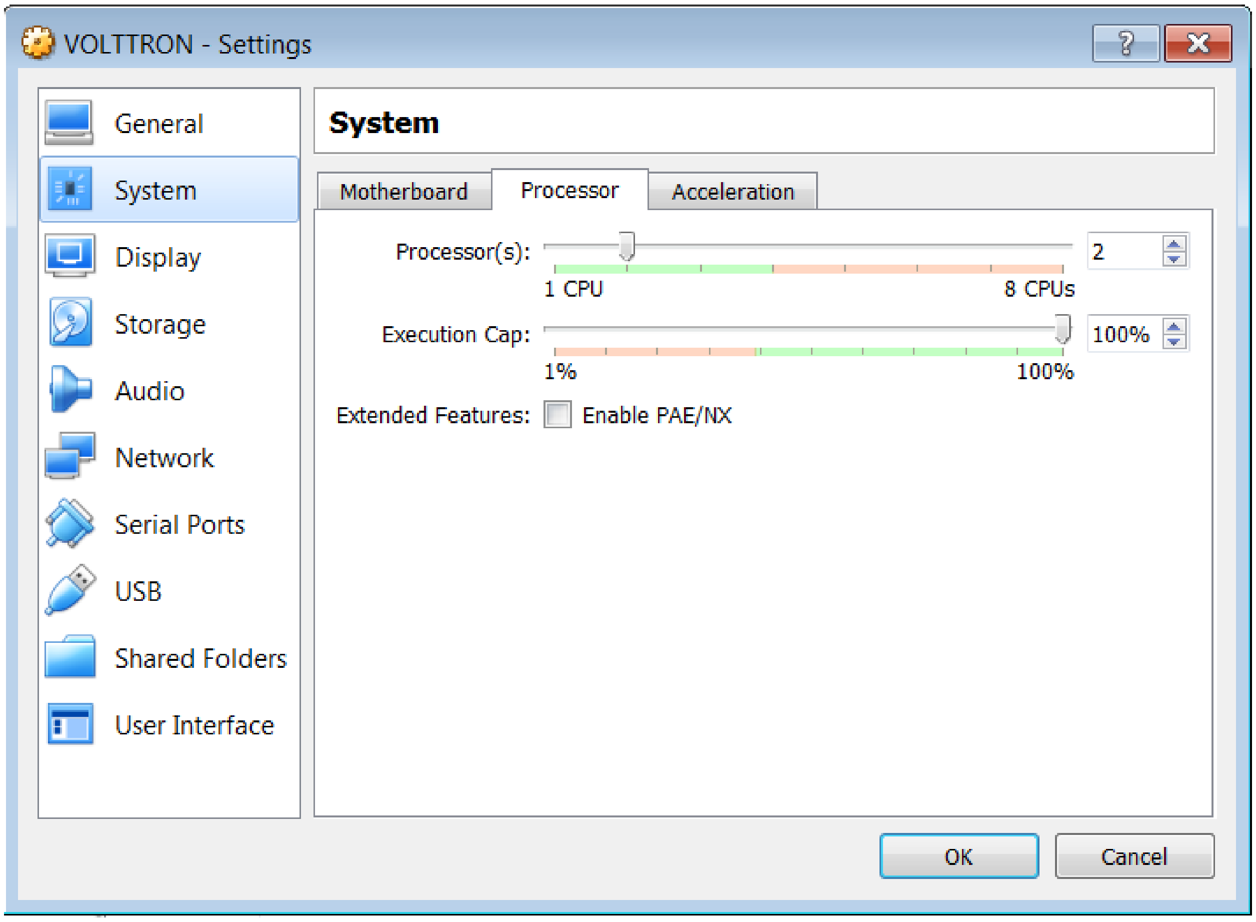
Snapshot Folder: C:\Users\volttron-u...ualBox VMs\VOLTTRON\Snapshots

Shared Clipboard: Bidirectional

Drag'n'Drop: Bidirectional

Note: Currently, this feature only works under certain circumstances (e.g. copying / pasting text).

Go to System Settings. In the processor tab, set the number of processors to two.



System

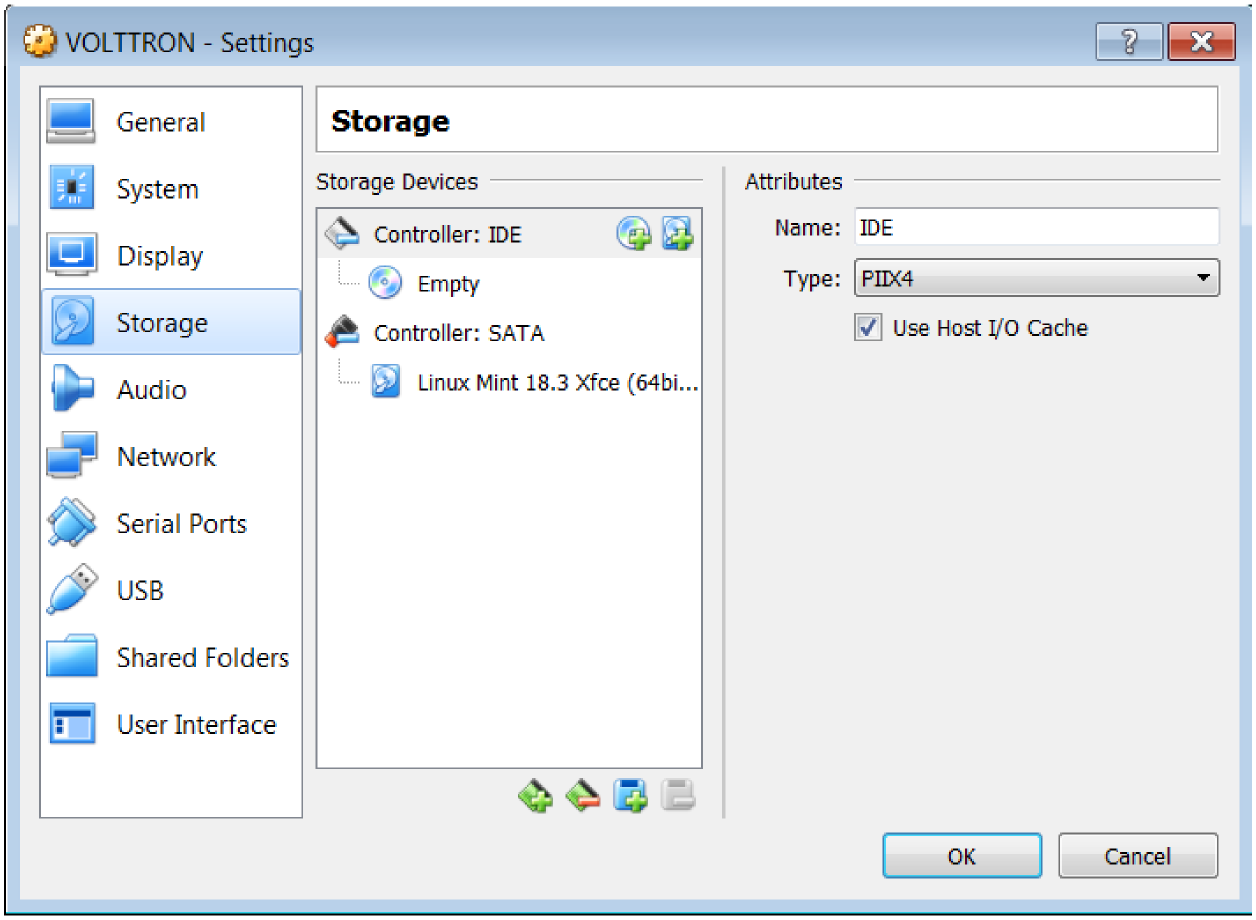
Processor(s): 2

Execution Cap: 100%

Extended Features: Enable PAE/NX

Go to Storage Settings. Confirm that the Linux Mint VDI is attached to Controller: SATA.

Danger: Do **NOT** mount the Linux Mint iso for Controller: IDE. **Will result in errors.**



Storage

Storage Devices

Controller: IDE
Empty
Controller: SATA
Linux Mint 18.3 Xfce (64bi...

Attributes

Name: IDE

Type: PIIX4

☑ Use Host I/O Cache

Start the machine by saving these changes and clicking the "Start" arrow located on the upper left hand corner of the main VirtualBox window.

1.7.3 Pycharm Development Environment

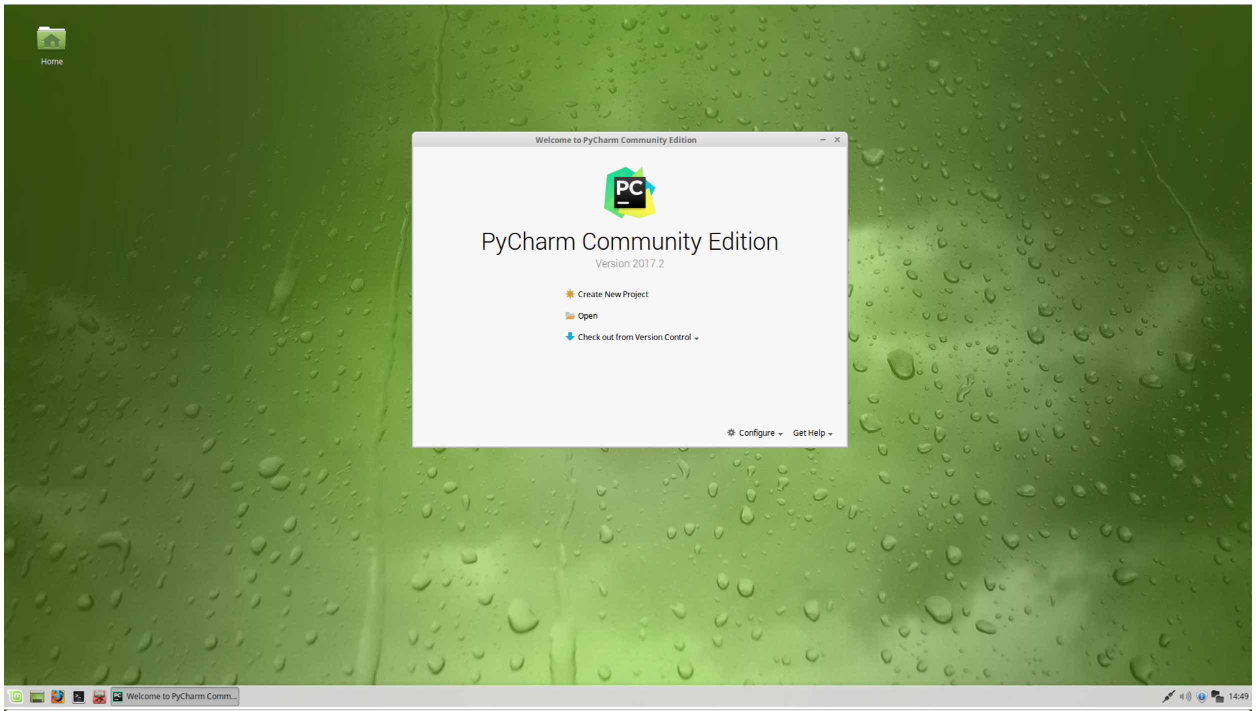
Pycharm is an IDE dedicated to developing python projects. It provides coding assistance and easy access to debugging tools as well as integration with py.test. It is a popular tool for working with VOLTTRON. Jetbrains provides a free community version that can be downloaded from https://www.jetbrains.com/pycharm/

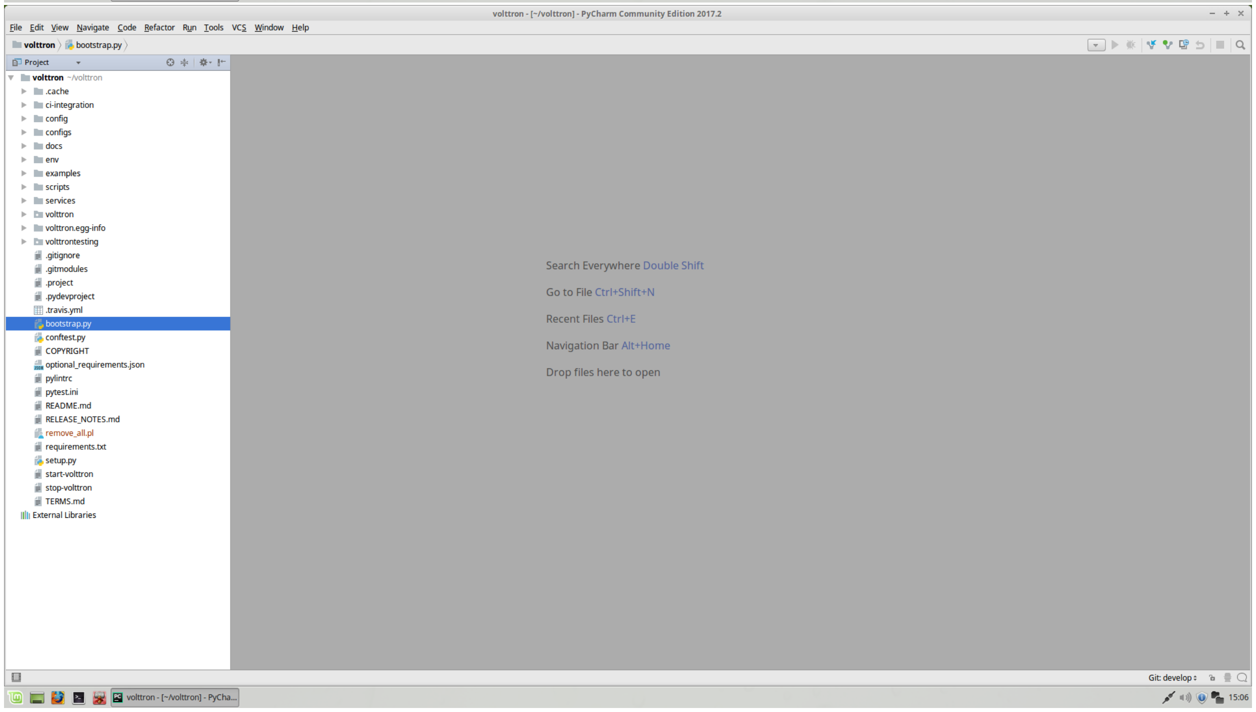
Open Pycharm and Load VOLTTRON

When launching Pycharm for the first time we have to tell it where to find the VOLTTRON source code. If you have already cloned the repo then point Pycharm to the cloned project. Pycharm also has options to access remote repositories.

Subsequent instances of Pycharm will automatically load the VOLTTRON project.

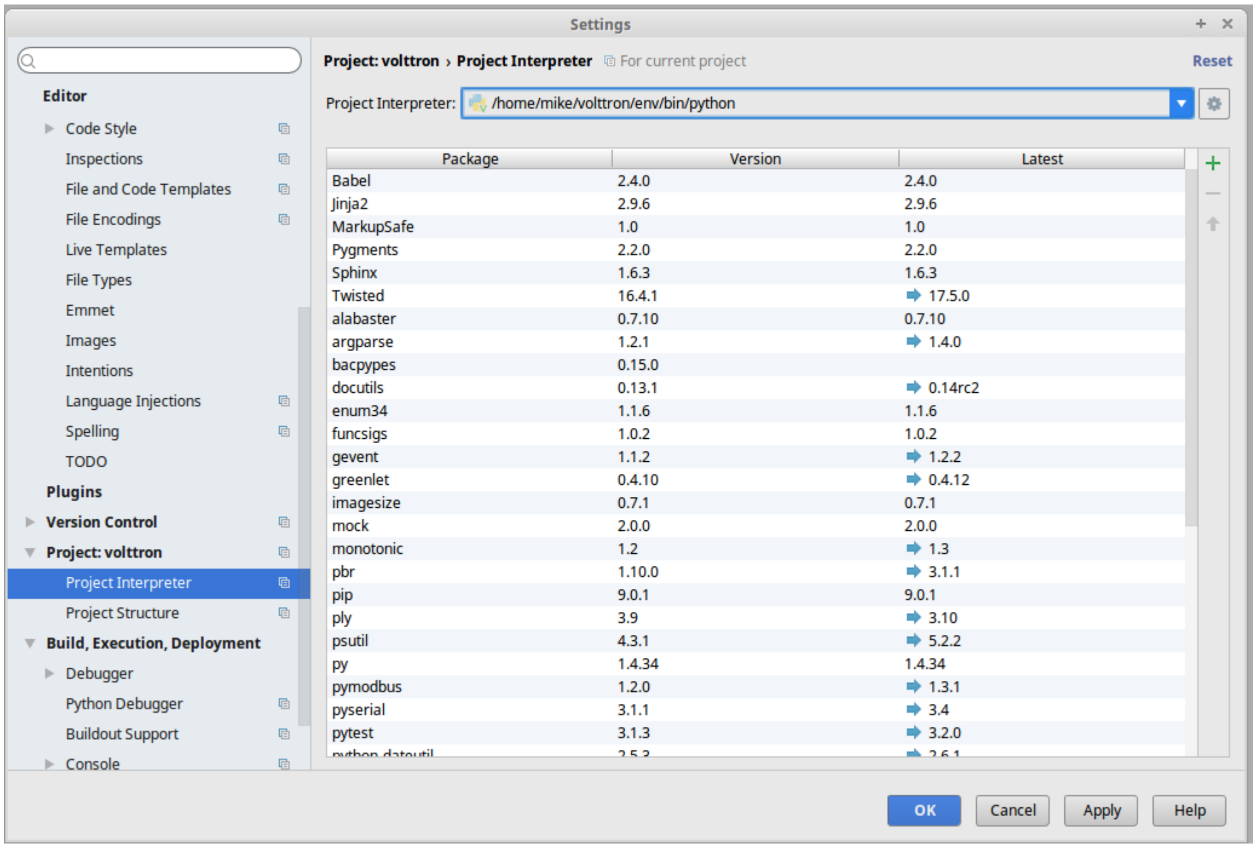
Note: When getting started make sure to search for *gevent* in the settings and ensure that support for it is enabled.


Welcome to PyCharm Community Edition
PyCharm Community Edition
Version 2017.2
Create New Project
Open
Check out from Version Control
Configure
Get Help



Set the Project Interpreter

This step should be completed after running the bootstrap script in the VOLTTRON source directory. Pycharm needs to know which python environment it should use when running and debugging code. This also tells Pycharm where to find python dependencies. Settings menu can be found under the File option in Pycharm.



Project Interpreter: /home/mike/volttron/env/bin/python

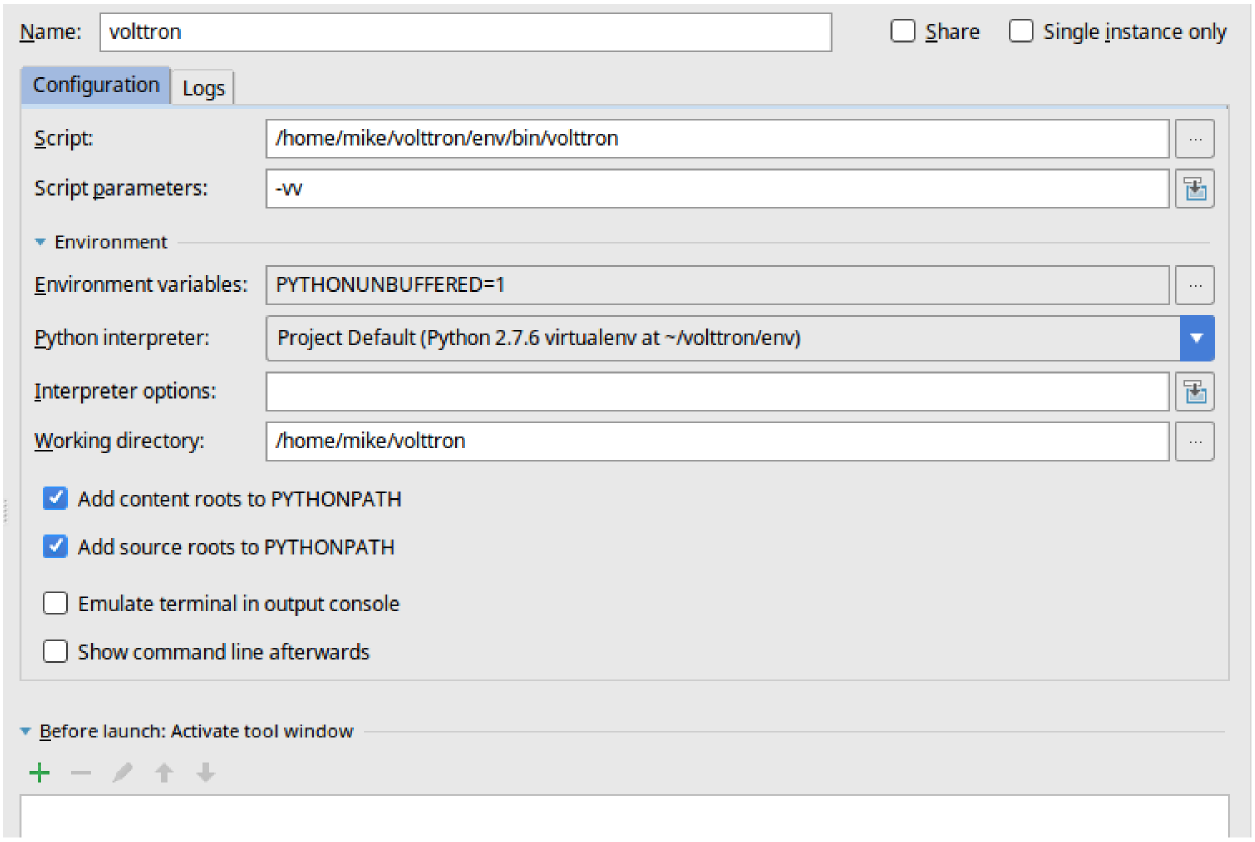
Package	Version	Latest
Babel	2.4.0	2.4.0
Jinja2	2.9.6	2.9.6
MarkupSafe	1.0	1.0
Pygments	2.2.0	2.2.0
Sphinx	1.6.3	1.6.3
Twisted	16.4.1	➡ 17.5.0
alabaster	0.7.10	0.7.10
argparse	1.2.1	➡ 1.4.0
bacpypes	0.15.0	
docutils	0.13.1	➡ 0.14rc2
enum34	1.1.6	1.1.6
funcsigs	1.0.2	1.0.2
gevent	1.1.2	➡ 1.2.2
greenlet	0.4.10	➡ 0.4.12
imagesize	0.7.1	0.7.1
mock	2.0.0	2.0.0
monotonic	1.2	➡ 1.3
pbr	1.10.0	➡ 3.1.1
pip	9.0.1	9.0.1
ply	3.9	➡ 3.10
psutil	4.3.1	➡ 5.2.2
py	1.4.34	1.4.34
pymodbus	1.2.0	➡ 1.3.1
pyserial	3.1.1	➡ 3.4
pytest	3.1.3	➡ 3.2.0
python-dateutil	2.5.3	➡ 2.6.1

Running the VOLTTRON Process

If you are not interested in running the VOLTTRON process itself in Pycharm then this step can be skipped.

In **Run > Edit Configurations** create a configuration that has *<your source dir>/env/bin/volttron* in the script field, *-vv* in the script parameters field (to turn on verbose logging), and set the working directory to the top level source directory.

VOLTTRON can then be run from the Run menu.



Name: volttron

☐ Share ☐ Single instance only

Script: /home/mike/volttron/env/bin/volttron

Script parameters: -vv

Environment variables: PYTHONUNBUFFERED=1

Python interpreter: Project Default (Python 2.7.6 virtualenv at ~/volttron/env)

Interpreter options:

Working directory: /home/mike/volttron

☑ Add content roots to PYTHONPATH

☑ Add source roots to PYTHONPATH

☐ Emulate terminal in output console

☐ Show command line afterwards

Before launch: Activate tool window

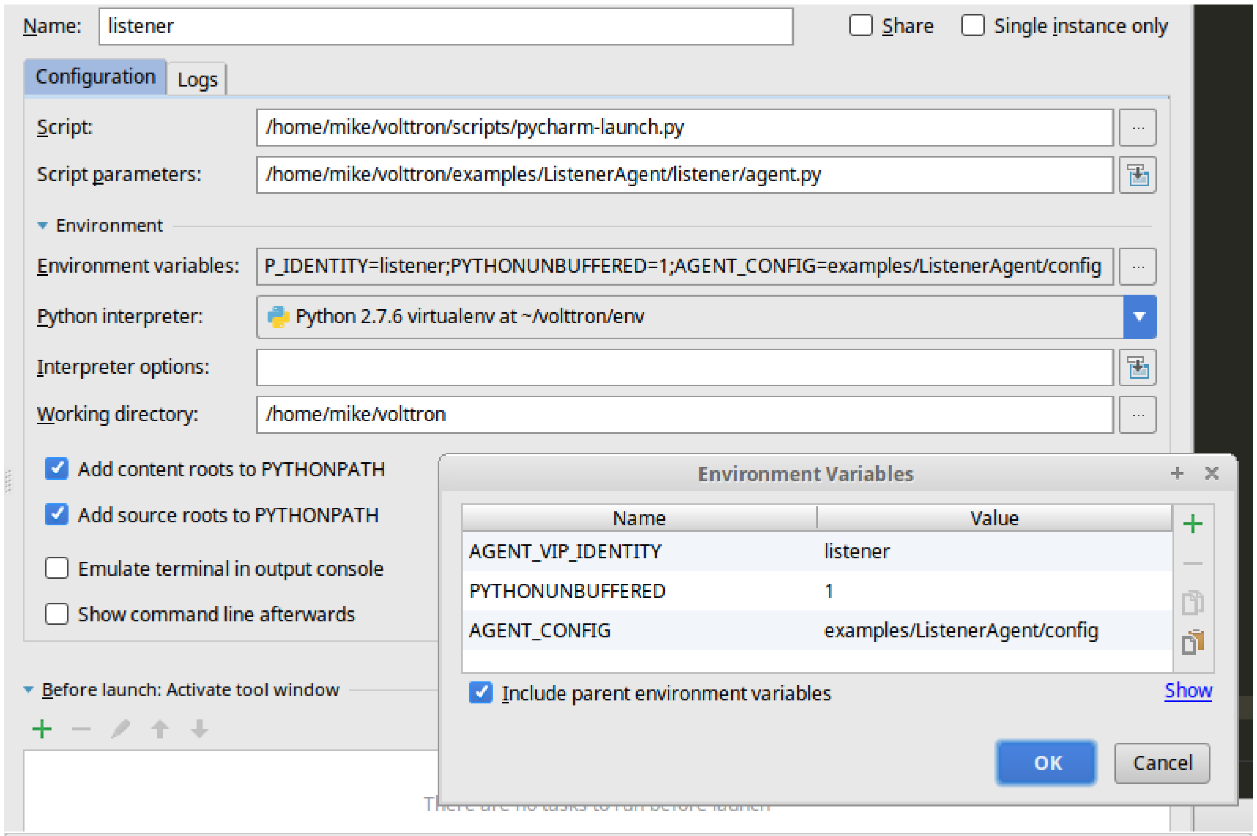
Running an Agent

Running an agent is configured similarly to running VOLTTRON proper. In **Run > Edit Configurations** add a configuration and give it the same name as your agent. The script should be the path to *scripts/pycharm-launch.py* and and the script parameter must be the path to your agent's *agent.py* file.

In the Environment Variables field add the variable *AGENT_CONFIG* that has the path to the agent's configuration file as its value, as well as *AGENT_VIP_IDENTITY*, which must be unique on the platform.

A good place to keep configuration files is in a directory called *config* in top level source directory; git will ignore changes to these files.

Note: There is an issue with imports in Pycharm when there is a secondary file (i.e. not *agent.py* but another module within the same package). When that happens right click on the directory in the file tree and select **Mark Directory As -> Source Root**


Share
Single instance only
Configuration
Logs
Environment
Before launch: Activate tool window
There are no tasks to run before launch
OK
Cancel
Show

Name: listener

Script: /home/mike/volttron/scripts/pycharm-launch.py

Script parameters: /home/mike/volttron/examples/ListenerAgent/listener/agent.py

Environment variables: P_IDENTITY=listener;PYTHONUNBUFFERED=1;AGENT_CONFIG=examples/ListenerAgent/config

Python interpreter: Python 2.7.6 virtualenv at ~/volttron/env

Interpreter options:

Working directory: /home/mike/volttron

☑ Add content roots to PYTHONPATH

☑ Add source roots to PYTHONPATH

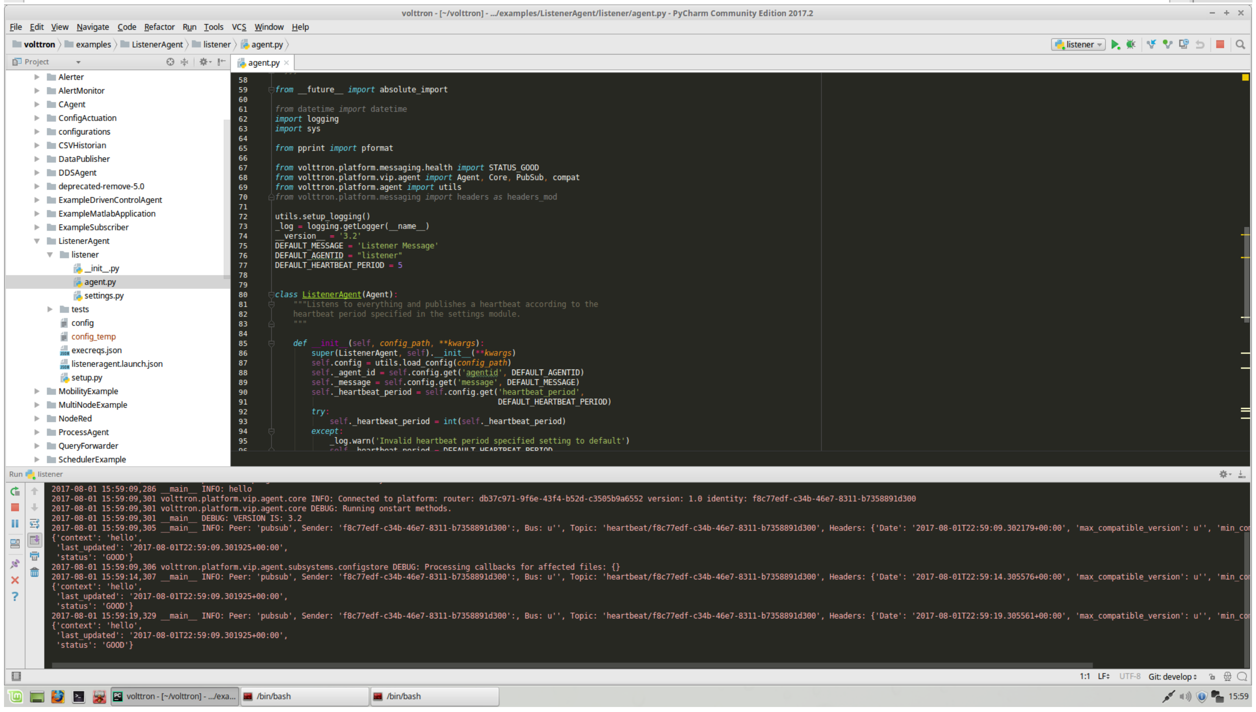
☐ Emulate terminal in output console

☐ Show command line afterwards

Environment Variables

Name	Value
AGENT_VIP_IDENTITY	listener
PYTHONUNBUFFERED	1
AGENT_CONFIG	examples/ListenerAgent/config

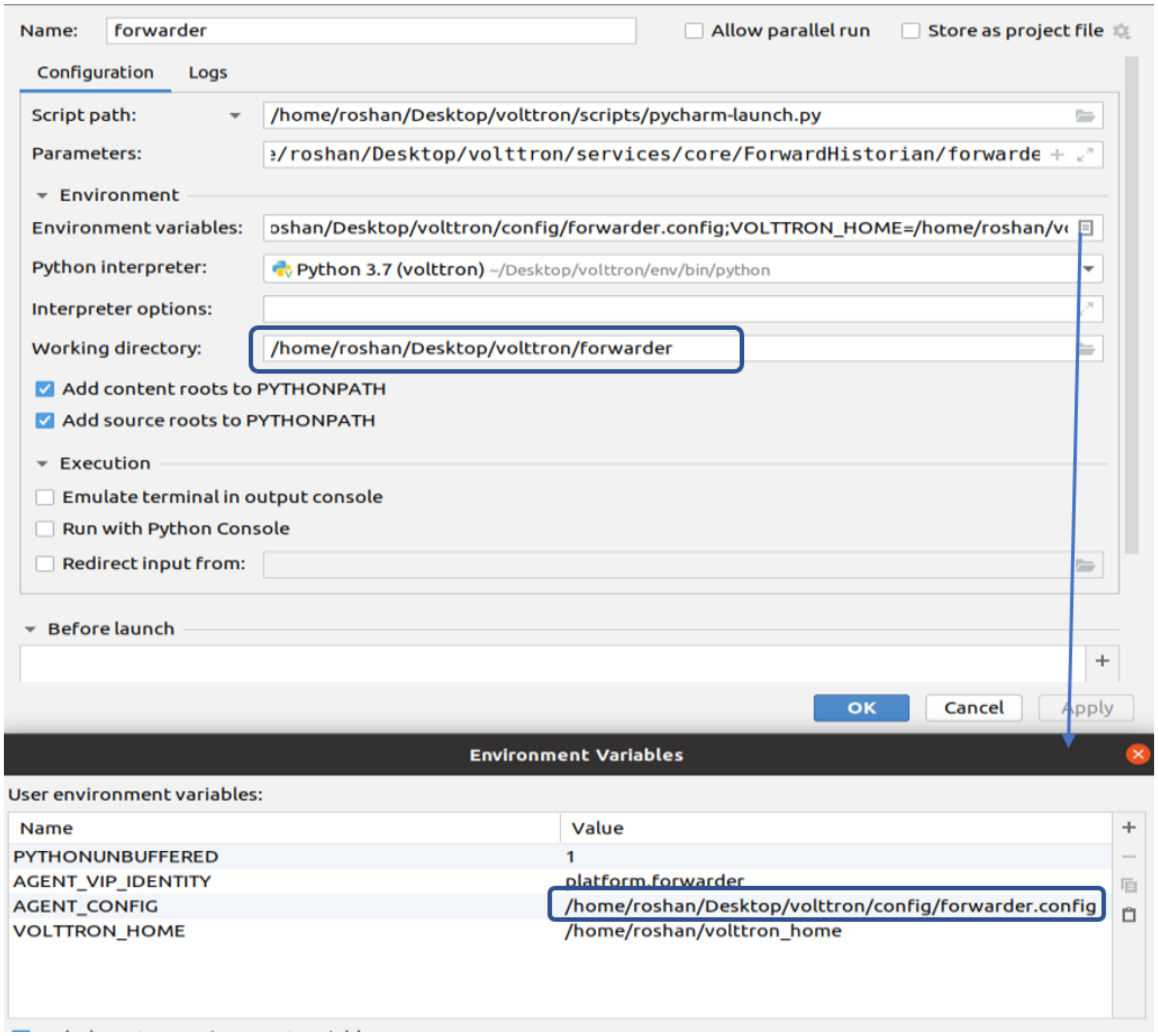
☑ Include parent environment variables



```
from __future__ import absolute_import

from datetime import datetime
import logging
import sys

from pprint import pformat

from volttron.platform.messaging.health import STATUS_GOOD
from volttron.platform.vip.agent import Agent, Core, PubSub, compat
from volttron.platform.agent import utils
from volttron.platform.messaging import headers as headers_mod

utils.setup_logging()
_log = logging.getLogger(__name__)
__version__ = '3.2'
DEFAULT_MESSAGE = 'Listener Message'
DEFAULT_AGENTID = "listener"
DEFAULT_HEARTBEAT_PERIOD = 5


class ListenerAgent(Agent):
    """Listens to everything and publishes a heartbeat according to the
    heartbeat period specified in the settings module.
    """

    def __init__(self, config_path, **kwargs):
        super(ListenerAgent, self).__init__(**kwargs)
        self.config = utils.load_config(config_path)
        self._agent_id = self.config.get('agentid', DEFAULT_AGENTID)
        self._message = self.config.get('message', DEFAULT_MESSAGE)
        self._heartbeat_period = self.config.get('heartbeat_period',
                                                 DEFAULT_HEARTBEAT_PERIOD)
        try:
            self._heartbeat_period = int(self._heartbeat_period)
        except:
            _log.warn('Invalid heartbeat period specified setting to default')
```

```
2017-08-01 15:59:09,286 __main__ INFO: hello
2017-08-01 15:59:09,301 volttron.platform.vip.agent.core INFO: Connected to platform: router: db37c971-9f6e-43f4-b52d-c3505b9a6552 version: 1.0 identity: f8c77edf-c34b-46e7-8311-b7358891d300
2017-08-01 15:59:09,301 volttron.platform.vip.agent.core DEBUG: Running onstart methods.
2017-08-01 15:59:09,301 __main__ DEBUG: VERSION IS: 3.2
2017-08-01 15:59:09,305 __main__ INFO: Peer: 'pubsub', Sender: 'f8c77edf-c34b-46e7-8311-b7358891d300':, Bus: u'', Topic: 'heartbeat/f8c77edf-c34b-46e7-8311-b7358891d300', Headers: {'Date': '2017-08-01T22:59:09.302179+00:00', 'max_compatible_version': u'', 'min_co
{'context': 'hello',
 'last_updated': '2017-08-01T22:59:09.301925+00:00',
 'status': 'GOOD'}
2017-08-01 15:59:09,306 volttron.platform.vip.agent.subsystems.configstore DEBUG: Processing callbacks for affected files: {}
2017-08-01 15:59:14,307 __main__ INFO: Peer: 'pubsub', Sender: 'f8c77edf-c34b-46e7-8311-b7358891d300':, Bus: u'', Topic: 'heartbeat/f8c77edf-c34b-46e7-8311-b7358891d300', Headers: {'Date': '2017-08-01T22:59:14.305576+00:00', 'max_compatible_version': u'', 'min_co
{'context': 'hello',
 'last_updated': '2017-08-01T22:59:09.301925+00:00',
 'status': 'GOOD'}
2017-08-01 15:59:19,329 __main__ INFO: Peer: 'pubsub', Sender: 'f8c77edf-c34b-46e7-8311-b7358891d300':, Bus: u'', Topic: 'heartbeat/f8c77edf-c34b-46e7-8311-b7358891d300', Headers: {'Date': '2017-08-01T22:59:19.305561+00:00', 'max_compatible_version': u'', 'min_co
{'context': 'hello',
 'last_updated': '2017-08-01T22:59:09.301925+00:00',
 'status': 'GOOD'}
```

Note: There will be issues if two agents create a file with the same name in the same working directory. For instance: SQLHistorian agent and Forwarder agent both create a backup.sqlite directory on the same working directory. When that happens both the agents attempt to use the same backup db and eventually lock the db. To avoid this situation, create different working directories for each agent and add the absolute path for the config file. The best way to go about this is to create a new folder and assign working directory to that folder as shown below.


Name: forwarder
☐ Allow parallel run ☐ Store as project file

Configuration | Logs

Script path: /home/roshan/Desktop/volttron/scripts/pycharm-launch.py
Parameters: ɜ/roshan/Desktop/volttron/services/core/ForwardHistorian/forwarde

Environment
Environment variables: ɔshan/Desktop/volttron/config/forwarder.config;VOLTTRON_HOME=/home/roshan/v‹
Python interpreter: Python 3.7 (volttron) ~/Desktop/volttron/env/bin/python
Interpreter options:
Working directory: /home/roshan/Desktop/volttron/forwarder
☑ Add content roots to PYTHONPATH
☑ Add source roots to PYTHONPATH

Execution
☐ Emulate terminal in output console
☐ Run with Python Console
☐ Redirect input from:

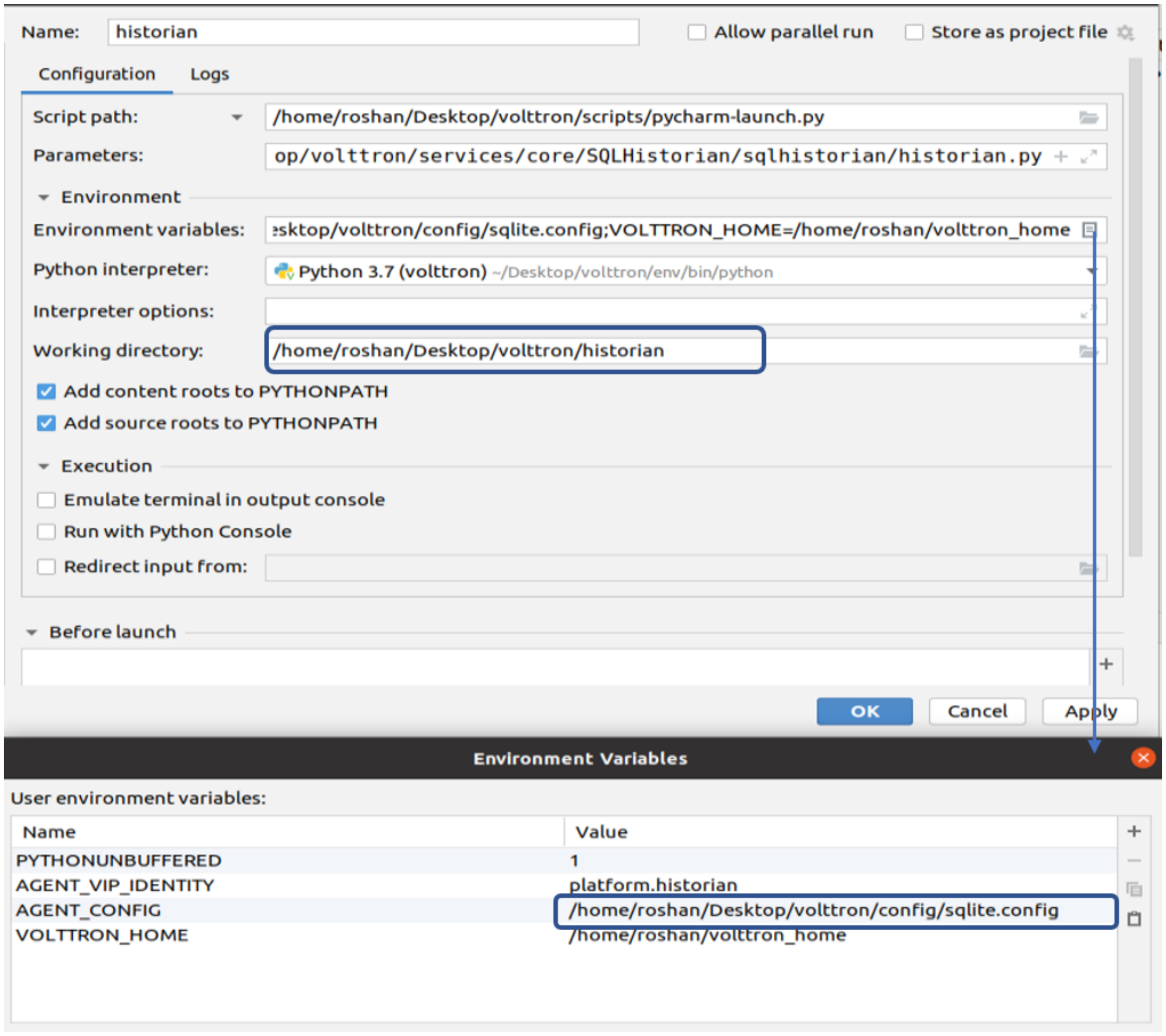
Before launch

OK | Cancel | Apply

Environment Variables

User environment variables:

Name	Value
PYTHONUNBUFFERED	1
AGENT_VIP_IDENTITY	platform.forwarder
AGENT_CONFIG	/home/roshan/Desktop/volttron/config/forwarder.config
VOLTTRON_HOME	/home/roshan/volttron_home


Name: historian

Configuration

Script path: /home/roshan/Desktop/volttron/scripts/pycharm-launch.py

Parameters: op/volttron/services/core/SQLHistorian/sqlhistorian/historian.py

Environment

Environment variables: ›sktop/volttron/config/sqlite.config;VOLTTRON_HOME=/home/roshan/volttron_home

Python interpreter: Python 3.7 (volttron) ~/Desktop/volttron/env/bin/python

Interpreter options:

Working directory: /home/roshan/Desktop/volttron/historian

[✓] Add content roots to PYTHONPATH

[✓] Add source roots to PYTHONPATH

Execution

[ ] Emulate terminal in output console

[ ] Run with Python Console

[ ] Redirect input from:

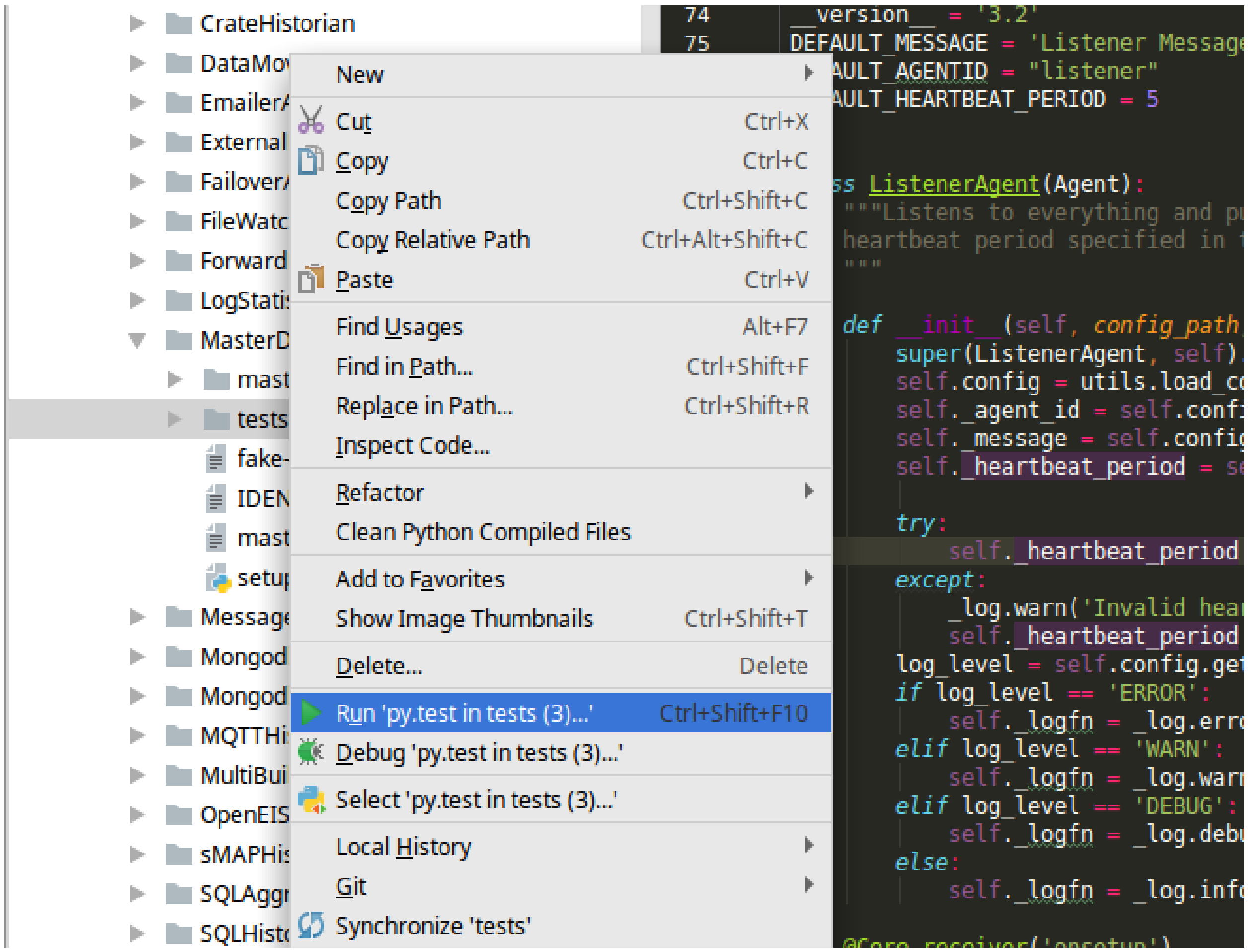
Environment Variables

User environment variables:

Name	Value
PYTHONUNBUFFERED	1
AGENT_VIP_IDENTITY	platform.historian
AGENT_CONFIG	/home/roshan/Desktop/volttron/config/sqlite.config
VOLTTRON_HOME	/home/roshan/volttron_home

Testing an Agent

Agent tests written in py.test can be run simply by right-clicking the tests directory and selecting *Run ‘py.test in tests*, so long as the root directory is set as the VOLTTRON source root.



```
__version__ = '3.2'
DEFAULT_MESSAGE = 'Listener Message
AULT_AGENTID = "listener"
AULT_HEARTBEAT_PERIOD = 5


ss ListenerAgent(Agent):
 """Listens to everything and pu
 heartbeat period specified in t
 """

 def __init__(self, config_path
     super(ListenerAgent, self)
     self.config = utils.load_co
     self._agent_id = self.confi
     self._message = self.config
     self._heartbeat_period = se

     try:
         self._heartbeat_period
     except:
         _log.warn('Invalid hear
         self._heartbeat_period
     log_level = self.config.get
     if log_level == 'ERROR':
         self._logfn = _log.erro
     elif log_level == 'WARN':
         self._logfn = _log.warn
     elif log_level == 'DEBUG':
         self._logfn = _log.debu
     else:
         self._logfn = _log.info
```

1.8 Agent Development

The VOLTTRON platform now has utilities to speed the creation and installation of new agents. To use these utilities the VOLTTRON environment must be activated.

From the project directory, activate the VOLTTRON environment with:

```
source env/bin/activate
```

1.8.1 Create Agent Code

Run the following command to start the Agent Creation Wizard:

```
vpkg init TestAgent tester
```

TestAgent is the directory that the agent code will be placed in. The directory must not exist when the command is run. *tester* is the name of the agent module created by wizard.

The Wizard will prompt for the following information:

```
Agent version number: [0.1]: 0.5
Agent author: []: VOLTTRON Team
Author's email address: []: volttron@pnnl.gov
Agent homepage: []: https://volttron.org/
Short description of the agent: []: Agent development tutorial.
```

Once the last question is answered the following will print to the console:

```
2018-08-02 12:20:56,604 () volttron.platform.packaging INFO: Creating TestAgent
2018-08-02 12:20:56,604 () volttron.platform.packaging INFO: Creating TestAgent/tester
2018-08-02 12:20:56,604 () volttron.platform.packaging INFO: Creating TestAgent/setup.
↪py
2018-08-02 12:20:56,604 () volttron.platform.packaging INFO: Creating TestAgent/config
2018-08-02 12:20:56,604 () volttron.platform.packaging INFO: Creating TestAgent/
↪tester/agent.py
2018-08-02 12:20:56,604 () volttron.platform.packaging INFO: Creating TestAgent/
↪tester/__init__.py
```

The TestAgent directory is created with the new Agent inside.

Agent Directory

At this point, the contents of the TestAgent directory should look like:

```
TestAgent/
├── setup.py
├── config
└── tester
    ├── agent.py
    └── __init__.py
```

Agent Skeleton

The *agent.py* file in the *tester* directory of the newly created agent module will contain skeleton code (below). Descriptions of the features of this code as well as additional development help are found in the rest of this document.

```
"""
Agent documentation goes here.
"""

__docformat__ = 'reStructuredText'

import logging
import sys
from volttron.platform.agent import utils
from volttron.platform.vip.agent import Agent, Core, RPC

_log = logging.getLogger(__name__)
utils.setup_logging()
__version__ = "0.1"


def tester(config_path, **kwargs):
    """Parses the Agent configuration and returns an instance of
```

(continues on next page)

(continued from previous page)

```
    the agent created using that configuration.

    :param config_path: Path to a configuration file.

    :type config_path: str
    :returns: Garbage
    :rtype: Garbage
    """
    try:
        config = utils.load_config(config_path)
    except StandardError:
        config = {}

    if not config:
        _log.info("Using Agent defaults for starting configuration.")

    setting1 = int(config.get('setting1', 1))
    setting2 = config.get('setting2', "some/random/topic")

    return Tester(setting1,
                      setting2,
                      **kwargs)


class Tester(Agent):
    """
    Document agent constructor here.
    """

    def __init__(self, setting1=1, setting2="some/random/topic",
                 **kwargs):
        super(Garbage, self).__init__(**kwargs)
        _log.debug("vip_identity: " + self.core.identity)

        self.setting1 = setting1
        self.setting2 = setting2

        self.default_config = {"setting1": setting1,
                               "setting2": setting2}


        #Set a default configuration to ensure that self.configure is called␣
↪immediately to setup
        #the agent.
        self.vip.config.set_default("config", self.default_config)
        #Hook self.configure up to changes to the configuration file "config".
        self.vip.config.subscribe(self.configure, actions=["NEW", "UPDATE"], pattern=
↪"config")

    def configure(self, config_name, action, contents):
        """
        Called after the Agent has connected to the message bus. If a configuration␣
↪exists at startup
        this will be called before onstart.

        Is called every time the configuration in the store changes.
        """
```

(continues on next page)

(continued from previous page)

```
        config = self.default_config.copy()
        config.update(contents)

        _log.debug("Configuring Agent")

        try:
            setting1 = int(config["setting1"])
            setting2 = str(config["setting2"])
        except ValueError as e:
            _log.error("ERROR PROCESSING CONFIGURATION: {}".format(e))
            return

        self.setting1 = setting1
        self.setting2 = setting2

        self._create_subscriptions(self.setting2)

    def _create_subscriptions(self, topic):
        #Unsubscribe from everything.
        self.vip.pubsub.unsubscribe("pubsub", None, None)

        self.vip.pubsub.subscribe(peer='pubsub',
                                  prefix=topic,
                                  callback=self._handle_publish)

    def _handle_publish(self, peer, sender, bus, topic, headers,
                                message):
        pass

    @Core.receiver("onstart")
    def onstart(self, sender, **kwargs):
        """
        This is method is called once the Agent has successfully connected to the␣
↪platform.
        This is a good place to setup subscriptions if they are not dynamic or
        do any other startup activities that require a connection to the message bus.
        Called after any configurations methods that are called at startup.

        Usually not needed if using the configuration store.
        """
        #Example publish to pubsub
        #self.vip.pubsub.publish('pubsub', "some/random/topic", message="HI!")

        #Exmaple RPC call
        #self.vip.rpc.call("some_agent", "some_method", arg1, arg2)

    @Core.receiver("onstop")
    def onstop(self, sender, **kwargs):
        """
        This method is called when the Agent is about to shutdown, but before it␣
↪disconnects from
        the message bus.
        """
        pass

    @RPC.export
    def rpc_method(self, arg1, arg2, kwarg1=None, kwarg2=None):
```

(continues on next page)

(continued from previous page)

```
        """
        RPC method

        May be called from another agent via self.core.rpc.call """
        return self.setting1 + arg1 - arg2

def main():
    """Main method called to start the agent."""
    utils.vip_main(garbage,
                   version=__version__)


if __name__ == '__main__':
    # Entry point for script
    try:
        sys.exit(main())
    except KeyboardInterrupt:
        pass
```

The resulting code is well documented with comments and documentation strings. It gives examples of how to do common tasks in VOLTTRON Agents. The main agent code is found in *tester/agent.py*.

1.8.2 Building an Agent

The following section includes guidance on several important components for building agents in VOLTTRON.

Parse Packaged Configuration and Create Agent Instance

The code to parse a configuration file packaged and installed with the agent is found in the *tester* function:

```
def tester(config_path, **kwargs):
    """Parses the Agent configuration and returns an instance of
    the agent created using that configuration.

    :param config_path: Path to a configuration file.

    :type config_path: str
    :returns: Tester
    :rtype: Tester
    """
    try:
        config = utils.load_config(config_path)
    except StandardError:
        config = {}

    if not config:
        _log.info("Using Agent defaults for starting configuration.")

    setting1 = int(config.get('setting1', 1))
    setting2 = config.get('setting2', "some/random/topic")

    return Tester(setting1,
                  setting2,
                  **kwargs)
```

The configuration is parsed with the *utils.load_config* function and the results are stored in the *config* variable. An instance of the Agent is created from the parsed values and is returned.

Initialization and Configuration Store Support

The *configuration store* is a powerful feature. The agent template provides a simple example of setting up default configuration store values and setting up a configuration handler.

```
class Tester(Agent):
    """
    Document agent constructor here.
    """

    def __init__(self, setting1=1, setting2="some/random/topic",
                 **kwargs):
        super(Tester, self).__init__(**kwargs)
        _log.debug("vip_identity: " + self.core.identity)

        self.setting1 = setting1
        self.setting2 = setting2

        self.default_config = {"setting1": setting1,
                               "setting2": setting2}


        #Set a default configuration to ensure that self.configure is called␣
↪immediately to setup
        #the agent.
        self.vip.config.set_default("config", self.default_config)
        #Hook self.configure up to changes to the configuration file "config".
        self.vip.config.subscribe(self.configure, actions=["NEW", "UPDATE"], pattern=
↪"config")

    def configure(self, config_name, action, contents):
        """
        Called after the Agent has connected to the message bus. If a configuration␣
↪exists at startup
        this will be called before onstart.

        Is called every time the configuration in the store changes.
        """
        config = self.default_config.copy()
        config.update(contents)

        _log.debug("Configuring Agent")

        try:
            setting1 = int(config["setting1"])
            setting2 = str(config["setting2"])
        except ValueError as e:
            _log.error("ERROR PROCESSING CONFIGURATION: {}".format(e))
            return

        self.setting1 = setting1
        self.setting2 = setting2

        self._create_subscriptions(self.setting2)
```

Note: Support for the configuration store is instantiated by subscribing to configuration changes with *self.vip.config.subscribe*.

```
self.vip.config.subscribe(self.configure_main, actions=["NEW", "UPDATE"], pattern=
↪"config")
```

Values in the default config can be built into the agent or come from the packaged configuration file. The subscribe method tells our agent which function to call whenever there is a new or updated config file. For more information on using the configuration store see *Agent Configuration Store*.

_create_subscriptions (covered in a later section) will use the value in *self.setting2* to create a new subscription.

Agent Lifecycle Events

The agent lifecycle is controlled in the agents VIP *core*. The agent lifecycle manages *scheduling and periodic function calls*, the main agent loop, and trigger a number of signals for callbacks in the concrete agent code. These callbacks are listed and described in the skeleton code below:

Note: The lifecycle signals can trigger any method. To cause a method to be triggered by a lifecycle signal, use a decorator:

```
@Core.receiver("<lifecycle_method>")
def my_callback(self, sender, **kwargs):
    # do my lifecycle method callback
    pass
```

```
@Core.receiver("onsetup")
def onsetup(self, sender, **kwargs)
    """
    This method is called after the agent has successfully connected to the platform,␣
↪but before the scheduled
    methods loop has started.  This method not often used, but is most commonly used␣
↪to define periodic
    functions or do some pre-configuration.
    """
    self.vip.core.periodic(60, send_request)

@Core.receiver("onstart")
def onstart(self, sender, **kwargs):
    """
    This method is called once the Agent has successfully connected to the platform.
    This is a good place to setup subscriptions if they are not dynamic or to
    do any other startup activities that require a connection to the message bus.
    Called after any configurations methods that are called at startup.

    Usually not needed if using the configuration store.
    """
    #Example publish to pubsub
    self.vip.pubsub.publish('pubsub', "some/random/topic", message="HI!")

    #Example RPC call
    self.vip.rpc.call("some_agent", "some_method", arg1, arg2)
```

(continues on next page)

(continued from previous page)

```
@Core.receiver("onstop")
def onstop(self, sender, **kwargs):
    """
    This method is called when the Agent is about to shutdown, but before it␣
↪disconnects from
    the message bus.  Common use-cases for this method are to stop periodic␣
↪processing, closing connections and
    setting agent state prior to cleanup.
    """
    self.publishing = False
    self.cache.close()

@Core.receiver("onfinish")
def onfinish(self, sender, **kwargs)
    """
    This method is called after all scheduled threads have concluded.  This method is␣
↪rarely used, but could be
    used to send shut down signals to other agents, etc.
    """
    self.vip.pubsub.publish('pubsub', 'some/topic', message=f'agent {self.core.
↪identity} shutdown')
```

Periodics and Scheduling

Periodic and Scheduled callback functions are callbacks made to functions in agent code from the thread scheduling in the agent core.

Scheduled Callbacks

Scheduled callback functions are often used similarly to cron jobs to perform tasks at specific times, or to schedule tasks ad-hoc as agent state is updated. There are 2 ways to schedule callbacks: using a decorator, or calling the core's scheduling function. Example usage follows.

```
# using the agent's core to schedule a task
self.core.schedule(periodic(5), self.sayhi)

def sayhi(self):
    print("Hello-World!")
```

```
# using the decorator to schedule a task
@Core.schedule(cron('0 1 * * *'))
def cron_function(self):
   print("this is a cron-scheduled function")
```

Note: Scheduled Callbacks can use CRON scheduling, a datetime object, a number of seconds (from current time), or a *periodic* which will make the schedule function as a periodic.

```
# inside some agent method
self.core.schedule(t, function)
self.core.schedule(periodic(t), periodic_function)
self.core.schedule(cron('0 1 * * *'), cron_function)
```

Periodic Callbacks

Periodic call back functions are functions which are repeatedly called at a regular interval until the periodic is cancelled in the agent code or the agent stops running. Like scheduled callbacks, periodics can be specified using either decorators or using core function calls.

```
self.core.periodic(10, self.saybye)

def saybye(self):
    print('Good-bye Cruel World!')
```

```
@Core.periodic(60)
def poll_api(self):
    return requests.get("https://lmgtfy.com").json()
```

Note: Periodic intervals are specified in seconds.

Publishing Data to the Message Bus

The agent's VIP connection can be used to publish data to the message bus. The message published and topic to publish to are determined by the agent implementation. Classes of agents already *specified by VOLTTRON* may have well-defined intended topic usage, see those agent specifications for further detail.

```
def publish_oscillating_update(self):
    self.publish_value = 1 if self.publish_value = 0 else 0
    self. vip.pubsub.publish('pubsub', 'some/topic/', message=f'{"oscillating_value":
↪"{self.publish_value}"')
```

Setting up a Subscription

The Agent creates a subscription to a topic on the message bus using the value of *self.setting2* in the method *_create_subscription*. The messages for this subscription are handled with the *_handle_publish* method:

```
def _create_subscriptions(self, topic):
    #Unsubscribe from everything.
    self.vip.pubsub.unsubscribe("pubsub", None, None)

    self.vip.pubsub.subscribe(peer='pubsub',
                              prefix=topic,
                              callback=self._handle_publish)

def _handle_publish(self, peer, sender, bus, topic, headers,
                            message):
    #By default no action is taken.
    pass
```

Alternatively, a decorator can be used to specify the function as a callback:

```
@PubSub.subscribe('pubsub', "topic_prefix")
def _handle_publish(self, peer, sender, bus, topic, headers,
                                message):
        #By default no action is taken.
        pass
```

To unsubscribe from a topic, the *self.vip.pubsub.unsubscribe* can be used:

```
self.vip.pubsub.unsubscribe(peer='pubsub',
                            prefix=topic,
                            callback=self._handle_publish)
```

Giving `None` as values for the prefix and callback argument will unsubscribe from everything on that bus. This is handy for subscriptions that must be updated base on a configuration setting.

Heartbeat

The heartbeat subsystem provides access to a periodic publish so that others can observe the agent's status. Other agents can subscribe to the *heartbeat* topic to see who is actively publishing to it. It it turned off by default.

Enabling the *heartbeat* publish:

Subscribing to the heartbeat topic:

Health

The health subsystem adds extra status information to the an agent's heartbeat. Setting the status will start the heartbeat if it wasn't already. Health is used to represent the internal state of the agent at runtime. *GOOD* health indicates that all is fine with the agent and it is operating normally. *BAD* health indicates some kind of problem, such as if an agent is unable to reach a remote web API.

Example of setting health:

Remote Procedure Calls

An agent may receive commands from other agents via a Remote Procedure Call (RPC). This is done with the *@RPC.export* decorator:

```
@RPC.export
def rpc_method(self, arg1, arg2, kwarg1=None, kwarg2=None):
    """
    RPC method

    May be called from another agent via self.core.rpc.call """
    return self.setting1 + arg1 - arg2
```

To send an RPC call to another agent running on the platform, the agent must invoke the *rpc.call* method of its VIP connection.

```
# in agent code
def send_remote_procedure_call(self):
    peer = "<agent identity>"
    peer_method = "<method in peer agent API>"
```

(continues on next page)

(continued from previous page)

```
    args = ["list", "of", "peer", "method", "arguments", "..."]
    self.vip.rpc.call(peer, peer_method, *args)
```

1.8.3 Packaging Configuration

The wizard will automatically create a *setup.py* file. This file sets up the name, version, required packages, method to execute, etc. for the agent based on your answers to the wizard. The packaging process will also use this information to name the resulting file.

```
from setuptools import setup, find_packages

MAIN_MODULE = 'agent'

# Find the agent package that contains the main module
packages = find_packages('.')
agent_package = 'tester'

# Find the version number from the main module
agent_module = agent_package + '.' + MAIN_MODULE
_temp = __import__(agent_module, globals(), locals(), ['__version__'], -1)
__version__ = _temp.__version__

# Setup
setup(
    name=agent_package + 'agent',
    version=__version__,
    author_email="volttron@pnnl.gov",
    url="https://volttron.org/",
    description="Agent development tutorial.",
    author="VOLTTRON Team",
    install_requires=['volttron'],
    packages=packages,
    entry_points={
        'setuptools.installation': [
            'eggsecutable = ' + agent_module + ':main',
        ]
    }
)
```

1.8.4 Launch Configuration

In TestAgent, the wizard will automatically create a JSON file called "config". It contains configuration information for the agent. This file contains examples of every data type supported by the configuration system:

```
{
  # VOLTTRON config files are JSON with support for python style comments.
  "setting1": 2, #Integers
  "setting2": "some/random/topic2", #Strings
  "setting3": true, #Booleans: remember that in JSON true and false are not␣
↪capitalized.
  "setting4": false,
  "setting5": 5.1, #Floating point numbers.
  "setting6": [1,2,3,4], #Lists
```

(continues on next page)

(continued from previous page)

```
  "setting7": {"setting7a": "a", "setting7b": "b"} #Objects
}
```

1.8.5 Packaging and Installation

To install the agent the platform must be running. Start the platform with the command:

```
./start-volttron
```

Note: If you are not in an activated environment, this script will start the platform running in the background in the correct environment. However the environment will not be activated for you; you must activate it yourself.

Now we must install it into the platform. Use the following command to install it and add a tag for easily referring to the agent. From the project directory, run the following command:

```
python scripts/install-agent.py -s TestAgent/ -c TestAgent/config -t testagent
```

To verify it has been installed, use the following command:

```
vctl list
```

This will result in output similar to the following:

```
    AGENT                     IDENTITY            TAG        Status     Health      PRI
df  testeragent-0.5           testeragent-0.5_1   testagent
```

- The first string is a unique portion of the full UUID for the agent
- AGENT is the "name" of the agent based on the contents of its class name and the version in its setup.py.
- IDENTITY is the agent's identity in the platform. This is automatically assigned based on class name and instance number. This agent's ID is _1 because it is the first instance.
- TAG is the name we assigned in the command above
- Status indicates the running status of an agent - running agents are *running*, agents which are not running will have no listed status
- Health is an indication of the internal state of the agent. 'Healthy' agents will have GOOD health. If an agent enters an error state, it will continue to run, but its health will be BAD.
- PRI is the priority for agents which have been "enabled" using the `vctl enable` command.

When using lifecycle commands on agents, they can be referred to by the UUID (default) or AGENT (name) or TAG.

1.8.6 Running and Testing the Agent

Now that the first pass of the agent code is complete, we can see if the agent works. It is highly-suggested to build a set of automated tests for the agent code prior to writing the agent, and running those tests after the agent is code-complete. Another quick way to determine if the agent is going the right direction is to run the agent on the platform using the VOLTTRON command line interface.

From the Command Line

To test the agent, we will start the platform (if not already running), launch the agent, and check the log file. With the VOLTTRON environment activated, start the platform by running (if needed):

```
./start-volttron
```

You can launch the agent in three ways, all of which you can find by using the *vctl list* command:

- By using the <uuid>:

```
vctl start <uuid>
```

- By name:

```
vctl start --name testeragent-0.1
```

- By tag:

```
vctl start --tag testagent
```

Check that it is *running*:

```
vctl status
```

- Start the ListenerAgent as in the *platform installation guide*.
- Check the log file for messages indicating the TestAgent is receiving the ListenerAgents messages:

```
TODO
```

Automated Test Cases and Documentation

Before contributing a new agent to the VOLTTRON source code repository, please consider adding two other essential elements.

1. Integration and unit test cases
2. README file that includes details of pre-requisite software, agent setup details (such as setting up databases, permissions, etc.) and sample configuration

VOLTTRON uses *pytest* as a framework for executing tests. All unit tests should be based on the *pytest* framework. For instructions on writing unit and integration tests with *pytest*, refer to the *Writing Agent Tests* documentation.

pytest is not installed with the distribution by default. To install py.test and it's dependencies execute the following:

```
python bootstrap.py --testing
```

Note: There are other options for different agent requirements. To see all of the options use:

```
python bootstrap.py --help
```

in the Extra Package Options section.

To run a single test module, use the command

```
pytest <testmodule.py>
```

To run all of the tests in the volttron repository execute the following in the root directory using an activated command prompt:

```
./ci-integration/run-tests.sh
```

1.8.7 Scripts

In order to make repetitive tasks less repetitive the VOLTTRON team has create several scripts in order to help. These tasks are available in the *scripts* directory.

Note: In addition to the *scripts* directory, the VOLTTRON team has added the config directory to the .gitignore file. By convention this is where we store customized scripts and configuration that will not be made public. Please feel free to use this convention in your own processes.

The *scripts/core* directory is laid out in such a way that we can build scripts on top of a base core. For example the scripts in sub-folders such as the *historian-scripts* and *demo-comms* use the scripts that are present in the core directory.

The most widely used script is *scripts/install-agent.py*. The *install_agent.py* script will remove an agent if the tag is already present, create a new agent package, and install the agent to *VOLTTRON_HOME*. This script has three required arguments and has the following signature:

Note: Agent to Package must have a setup.py in the root of the directory. Additionally, the user must be in an activated Python Virtual Environment for VOLTTRON

```
cd $VOLTTRON_ROOT
source env/bin/activate
```

```
python scripts/install_agent.py -s <agent path> -c <agent config file> -i <agent VIP␣
↪identity> --tag <Tag>
```

Note: The `--help` optional argument can be used with *scripts/install-agent.py* to view all available options for the script

The *install_agent.py* script will respect the *VOLTTRON_HOME* specified on the command line or set in the global environment. An example of setting *VOLTTRON_HOME* to */tmp/v1home* is as follows.

```
VOLTTRON_HOME=/tmp/v1home python scripts/install-agent.py -s <Agent to Package> -c
↪<Config file> --tag <Tag>
```

Agent Configuration Store Interface

The Agent Configuration Store Subsystem provides an interface for facilitating dynamic configuration via the platform configuration store. It is intended to work alongside the original configuration file to create a backwards compatible system for configuring agents with the bundled configuration file acting as default settings for the agent.

If an Agent Author does not want to take advantage of the platform configuration store they need to make no changes. To completely disable the Agent Configuration Store Subsystem an Agent may pass *enable_store=False* to the *Agent.__init__* method.

The Agent Configuration Store Subsystem caches configurations as the platform sends updates to the agent. Updates from the platform will usually trigger callbacks on the agent.

Agent access to the Configuration Store is managed through the *self.vip.config* object in the Agent class.

The "config" Configuration

The configuration name *config* is considered the canonical name of an Agents main configuration. As such the Agent will always run callbacks for that configuration first at startup and when a change to another configuration triggers any callbacks for *config*.

Configuration Callbacks

Agents may setup callbacks for different configuration events. The callback method must have the following signature:

```
my_callback(self, config_name, action, contents)
```

Note: The example above is for a class member method, however the method does not need to be a member of the agent class.

- **config_name** - The method to call when a configuration event occurs.
- **action** - The specific configuration event type that triggered the callback. Possible values are "NEW", "UPDATE", "DELETE". See *Configuration Events*
- **contents** - The actual contents of the configuration. Will be a string, list, or dictionary for the actions "NEW" and "UPDATE". None if the action is "DELETE".

Note: All callbacks which are connected to the "NEW" event for a configuration will called during agent startup with the initial state of the configuration.

Configuration Events

- **NEW** - This event happens for every existing configuration at Agent startup and whenever a new configuration is added to the Configuration Store.
- **UPDATE** - This event happens every time a configuration is changed.
- **DELETE** - The event happens every time a configuration is removed from the store.

Setting Up a Callback

A callback is setup with the *self.vip.config.subscribe* method.

Note: Subscriptions may be setup at any point in the life cycle of an Agent. Ideally they are setup in __init__.

```
subscribe(callback, actions=["NEW", "UPDATE", "DELETE"], pattern="*")
```

- **callback** - The method to call when a configuration event occurs.
- **actions** - The specific configuration event that will trigger the callback. May be a string with the name of a single action or a list of actions.
- **pattern** - The pattern used to match configuration names to trigger the callback.

Configuration Name Pattern Matching

Configuration name matching uses Unix file name matching semantics. Specifically the python module `fnmatch` is used.

Name matching is not case sensitive regardless of the platform VOLTTRON is running on.

For example, the pattern *devices/** will trigger the supplied callback for any configuration name that starts with *devices/*.

The default pattern matches all configurations.

Getting a Configuration

Once RPC methods are available to an agent (once onstart methods have been called or from any configuration callback) the contents of any configuration may be acquired with the *self.vip.config.get* method.

```
get(config_name="config")
```

If the Configuration Subsystem has not been initialized with the starting values of the agent configuration that will happen in order to satisfy the request.

If initialization occurs to satisfy the request callbacks will *not* be called before returning the results.

Typically an Agent will only obtain the contents of a configuration via a callback. This method is included for agents that want to save state in the store and only need to retrieve the contents of a configuration at startup and ignore any changes to the configuration going forward.

Setting a Configuration

Once RPC methods are available to an agent (once onstart methods have been called) the contents of any configuration may be set with the *self.vip.config.set* method.

```
set(config_name, contents, trigger_callback=False, send_update=False)
```

The contents of the configuration may be a string, list, or dictionary.

This method is intended for agents that wish to maintain a copy of their state in the store for retrieval at startup with the *self.vip.config.get* method.

Warning: This method may **not** be called from a configuration callback. The Configuration Subsystem will detect this and raise a `RuntimeError`, even if *trigger_callback* or *send_update* is False.

The platform has a locking mechanism to prevent concurrent configuration updates to the Agent. Calling *self.vip.config.set* would cause the Agent and the Platform configuration store for that Agent to deadlock until a timeout occurs.

Optionally an agent may trigger any callbacks by setting *trigger_callback* to True. If *trigger_callback* is set to False the platform will still send the updated configuration back to the agent. This ensures that a subsequent call to *self.cip.config.get* will still return the correct value. This way the agent's configuration subsystem is kept in sync with the platform's copy of the agent's configuration store at all times.

Optionally the agent may prevent the platform from sending the updated file to the agent by setting *send_update* to False. This setting is available strictly for performance tuning.

Warning: This setting will allow the agent's view of the configuration to fall out of sync with the platform. Subsequent calls to *self.vip.config.get* will return an old version of the file if it exists in the agent's view of the configuration store.

This will also affect any configurations that reference the configuration changed with this setting.

Care should be taken to ensure that the configuration is only retrieved at agent startup when using this option.

Setting a Default Configuration

In order to more easily allow agents to use both the Configuration Store while still supporting configuration via the tradition method of a bundled configuration file the *self.vip.config.set_default* method was created.

```
set_default(config_name, contents)
```

Warning: This method may **not** be called once the Agent Configuration Store Subsystem has been initialized. This method should only be called from *__init__* or an *onsetup* method.

The *set_default* method adds a temporary configuration to the Agents Configuration Subsystem. Nothing is sent to the platform. If a configuration with the same name exists in the platform store it will be presented to a callback method in place of the default configuration.

The normal way to use this is to set the contents of the packaged Agent configuration as the default contents for the configuration named *config*. This way the same callback used to process *config* configuration in the Agent will be called when the Configuration Subsystem can be used to process the configuration file packaged with the Agent.

Note: No attempt is made to merge a default configuration with a configuration from the store.

If a configuration is deleted from the store and a default configuration exists with the same name the Agent Configuration Subsystem will call the *UPDATE* callback for that configuration with the contents of the default configuration.

Other Methods

In a well thought out configuration scheme these methods should not be needed but are included for completeness.

List Configurations

A current list of all configurations for the Agent may be called with the *self.vip.config.list* method.

Unsubscribe

All subscriptions can be removed with a call to the *self.vip.config.unsubscribe_all* method.

Delete

A configuration can be deleted with a call to the *self.vip.config.delete* method.

```
delete(config_name, trigger_callback=False)
```

Note: This method may **not** be called from a callback for the same reason as the *self.vip.config.set* method.

Delete Default

A default configuration can be deleted with a call to the *self.vip.config.delete_default* method.

```
delete_default(config_name)
```

Warning: This method may **not** be called once the Agent Configuration Store Subsystem has been initialized. This method should only be called from *__init__* or an *onsetup* method.

Example Agent

The following example shows how to use set_default with a basic configuration and how to setup callbacks.

```
def my_agent(config_path, **kwargs):

    config = utils.load_config(config_path) #Now returns {} if config_path does not␣
↪exist.

    setting1 = config.get("setting1", 42)
    setting2 = config.get("setting2", 2.5)

    return MyAgent(setting1, setting2, **kwargs)

class MyAgent(Agent):
    def __init__(self, setting1=0, setting2=0.0, **kwargs):
        super(MyAgent, self).__init__(**kwargs)

        self.default_config = {"setting1": setting1,
                               "setting2": setting2}

        self.vip.config.set_default("config", self.default_config)
```

(continues on next page)

(continued from previous page)

```
        #Because we have a default config we don't have to worry about "DELETE"
        self.vip.config.subscribe(self.configure_main, actions=["NEW", "UPDATE"],␣
↪pattern="config")
        self.vip.config.subscribe(self.configure_other, actions=["NEW", "UPDATE"],␣
↪pattern="other_config/*")
        self.vip.config.subscribe(self.configure_delete, actions="DELETE", pattern=
↪"other_config/*")

    def configure_main(self, config_name, action, contents):
        #Ensure that we use default values from anything missing in the configuration.
        config = self.default_config.copy()
        config.update(contents)

        _log.debug("Configuring MyAgent")

        #Sanity check the types.
        try:
            setting1 = int(config["setting1"])
            setting2 = float(config["setting2"])
        except ValueError as e:
            _log.error("ERROR PROCESSING CONFIGURATION: {}".format(e))
            #TODO: set a health status for the agent
            return

        _log.debug("Using setting1 {}, setting2 {}". format(setting1, setting2))
        #Do something with setting1 and setting2.

    def configure_other(self, config_name, action, contents):
        _log.debug("Configuring From {}".format(config_name))
        #Do something with contents of configuration.

    def configure_delete(self, config_name, action, contents):
        _log.debug("Removing {}".format(config_name))
        #Do something in response to the removed configuration.
```

Writing Agent Tests

The VOLTTRON team strongly encourages developing agents with a set of unit and integration tests. Test-driven development can save developers significant time and effort by clearly defining behavioral expectations for agent code. We recommend developing agent tests using Pytest. Agent code contributed to VOLTTRON is expected to include a set of tests using Pytest in the agent module directory. Following are instructions for setting up Pytest, structuring your tests, how to write unit and integration tests (including some helpful tools using Pytest and Mock) and how to run your tests.

Installation

To get started with Pytest, install it in an activated environment:

```
pip install pytest
```

Or when running VOLTTRON's bootstrap process, specify the `--testing` optional argument.

```
python bootstrap.py --testing
```

Pytest on PyPI

Module Structure

We suggest the following structure for your agent module:

```
├── UserAgent
│   ├── user_agent
│   │   ├── data
│   │   │   └── user_agent_data.csv
│   │   ├── __init__.py
│   │   └── agent.py
│   ├── tests
│   │   └── test_user_agent.py
│   ├── README.md
│   ├── config.json
│   ├── contest.py
│   ├── requirements.txt
│   └── setup.py
```

The test suite should be in a *tests* directory in the root agent directory, and should contain one or more test code files (with the *test_<name of test>* convention). *conftest.py* can be used to give all agent tests access to some portion of the VOLTTRON code. In many cases, agents use *conftest.py* to import VOLTTRON testing fixtures for integration tests.

Naming Conventions

Pytest tests are discovered and run using some conventions:

- Tests will be found recursively in either the directory specified when running Pytest, or the current working directory if no argument was supplied
- Pytest will search in those directories for files called test_<name of test>.py or <name of test>_test.py
- **In those files, Pytest will test:**
  - functions and methods prefixed by "test" outside of any class
  - functions and methods prefixed by "test" inside of any class prefixed by "test"

```
├── TestDir
│   ├── MoreTests
│   │   ├── test2.py
│   ├── test1.py
│   └── file.py
```

```python
# test1.py

def helper_method():
    return 1

def test_success():
    assert helper_method()

# test2.py

def test_success():
```

(continues on next page)

(continued from previous page)

```
    assert True

def test_fail():
    assert False

# file.py

def test_success():
    assert True

def test_fail():
    assert False
```

In the above example, Pytest will run the tests *test_success* from the file test1.py and *test_success* and test_fail from test2.py. No tests will be run from file.txt, even though it contains test code, nor will it try to run *helper_method* from test1.py as a test.

Writing Unit Tests

These tests should test the various methods of the code base, checking for success and fail conditions. These tests should capture how the components of the system should function; and describe all the possible output conditions given the possible range of inputs including how they should fail if given improper input.

Pytest guide to Unit Testing

Mocking Dependencies

VOLTTRON agents include code for many platform features; these features can be mocked to allow unit tests to test only the features of the agent without having to account for the behaviors of the core platform. While there are many tools that can mock dependencies of an agent, we recommend Volttron's AgentMock or Python's Mock testing library.

AgentMock

AgentMock was specifically created to run unit tests on agents. AgentMock takes an Agent class and mocks the attributes and methods of that Agent's dependencies. AgentMock also allows you to customize the behavior of dependencies within each individual test. Below is an example:

```
# Import the Pytest, Mock, base Agent, and Agent mock utility from VOLTTRON's␣
↪repository
import pytest
import mock
from volttron.platform.vip.agent import Agent
from volttrontesting.utils.utils import AgentMock
# Import your agent code
from UserAgent import UserAgentClass

UserAgentClass.__bases__ = (AgentMock.imitate(Agent, Agent()),)
agent = UserAgentClass()

def test_success_case():
    result = agent.do_function("valid input")
    assert isinstance(result, dict)
```

(continues on next page)

(continued from previous page)

```
    for key in ['test1', 'test2']:
        assert key in result
    assert result.get("test1") == 10
    assert isinstance(result.get("test2"), str)
    # ...

def test_success_case_custom_mocks():
    agent.some_dependency.some_method.return_value = "foobar"
    agent.some_attribute = "custom, dummy value"
    result = agent.do_function_that_relies_on_custom_mocks("valid input")
    # ...

def test_failure_case()
    # pytests.raises can be useful for testing exceptions, more information about␣
↪usage below
    with pytest.raises(ValueError, match=r'Invalid input string for do_function')
        result = agent.do_function("invalid input")
```

Mock

Simliar to AgentMock, Python's Mock testing library allows a user to replace the behavior of dependencies with a user-specified behavior. This is useful for replacing VOLTTRON platform behavior, remote API behavior, modules, etc. where using them in unit or integration tests is impractical or impossible. Below is an example that uses the patch decorator to mock an Agent's web request.

Mock documentation

```
class UserAgent()

    def __init__():
        # Code here

    def get_remote_data()
        response = self._get_data_from_remote()
        return "Remote response: {}".format(response)

    # it can be useful to create private functions for use with mock for things like␣
↪making web requests
    def _get_data_from_remote():
        url = "test.com/test1"
        headers = {}
        return requests.get(url, headers)

# ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

import pytest
import mock

def get_mock_response():
    return "test response"

# here we're mocking the UserAgent's _get_data_from_remote method and replacing it␣
↪with our get_mock_response method
# to feed our test some fake remote data
@mock.patch.object(UserAgent, '_get_data_from_remote', get_mock_response)
```

(continues on next page)

(continued from previous page)

```
def test_get_remote_data():
        assert UserAgent.get_remote_Data() == "Remote response: test response"
```

Pytest Tools

Pytest includes many helpful tools for developing your tests. We'll highlight a few that have been useful for VOLTTRON core tests, but checkout the Pytest documentation for additional information on each tool as well as tools not covered in this guide.

Pytest Fixtures

Pytest fixtures can be used to create reusable code for tests that can be accessed by every test in a module based on scope. There are several kinds of scopes, but commonly used are "module" (the fixture is run once per module for all the tests of that module) or "function" (the fixture is run once per test). For fixtures to be used by tests, they should be passed as parameters.

Pytest Fixture documentation

Here is an example of a fixture, along with using it in a test:

```
# Fixtures with scope function will be run once per test if the test accepts the␣
↪fixture as a parameter
@pytest.fixture(scope="function")
def cleanup_database():
    # This fixture cleans up a sqlite database in between each test run
    sqlite_conn = sqlite.connect("test.sqlite")
    cursor = sqlite_conn.cursor()
    cursor.execute("DROP TABLE 'TEST'")
    cursor.commit()

    cursor.execute("CREATE TABLE TEST (ID INTEGER, FirstName TEXT, LastName TEXT,␣
↪Occupation Text)")
    cursor.commit()
    sqlite.conn.close()

# when we pass the cleanup function, we expect that the table will be dropped and␣
↪rebuilt before the test runs
def test_store_data(cleanup_database):
    sqlite_conn = sqlite.connect("test.sqlite")
    cursor = sqlite_conn.cursor()
    # after this insert, we'd expect to only have 1 value in the table
    cursor.execute("INSERT INTO TEST VALUES(1, 'Test', 'User', 'Developer')")
    cursor.commit()

    # validate the row count
    cursor.execute("SELECT COUNT(*) FROM TEST")
    count = cursor.fetchone()
    assert count == 1
```

Pytest.mark

Pytest marks are used to set metadata for test functions. Defining your own custom marks can allow you to run subsections of your tests. Parametrize can be used to pass a series of parameters to a test, so that it can be run many times to cover the space of potential inputs. Marks also exist to specify expected behavior for tests.

Mark documentation

Custom Marks

To add a custom mark, add the name of the mark followed by a colon then a description string to the 'markers' section of Pytest.ini (an example of this exists in the core VOLTTRON repository). Then add the appropriate decorator:

```
@pytest.mark.UserAgent
def test_success_case():
    # TODO unit test here
    pass
```

The VOLTTRON team also has a *dev* mark for running individual (or a few) one-off tests.

```
@pytest.mark.dev
@pytest.mark.UserAgent
def test_success_case():
    # TODO unit test here
    pass
```

Parametrize

Parametrize will allow tests to be run with a variety of parameters. Add the parametrize decorator, and for parameters include a list of parameter names matching the test parameter names as a comma-delimited string followed by a list of tuples containing parameters for each test.

Parametrize docs

```
@pytest.mark.parametrize("test_input1, test_input2, expected", [(1, 2, 3), (-1, 0, "
↪")])
def test_user_agent(param1, param2, param3):
    # TODO unit test here
    pass
```

Skip, skipif, and xfail

The *skip* mark can be used to skip a test for any reason every time the test suite is run:

```
# This test will be skipped!
@pytest.mark.skip
def test_user_agent():
    # TODO unit test here
    pass
```

The *skipif* mark can be used to skip a test based on some condition:

```
# This test will be skipped if RabbitMQ hasn't been set up yet!
@pytest.mark.skipif(not isRabbitMQInstalled)
def test_user_agent():
    # TODO unit test here
    pass
```

The *xfail* mark can be used to run a test, but to show that the test is currently expected to fail

```
# This test will fail, but will not cause the module tests to be considered failing!
@pytest.mark.xfail
def test_user_agent():
    # TODO unit test here
    assert False
```

Skip, skipif, and xfail docs

Writing Integration Tests

Integration tests are useful for testing the faults that occur between integrated units. In the context of VOLTTRON agents, integration tests should test the interactions between the agent, the platform, and other agents installed on the platform that would interface with the agent. It is typical for integration tests to test configuration, behavior and content of RPC calls and agent Pub/Sub, the agent subsystems, etc.

Pytest best practices for Integration Testing

Volttrontesting Directory

The *Volttrontesting* directory includes several helpful fixtures for your tests. Including the following line at the top of your tests, or in *conftest.py*, will allow you to utilize the platform wrapper fixtures, and more.

```
from volttrontesting.fixtures.volttron_platform_fixtures import *
```

Here is an example success case integration test:

```
import pytest
import mock
from volttrontesting.fixtures.volttron_platform_fixtures import *

# If the test requires user specified values, setting environment variables or having␣
↪settings files is recommended
API_KEY = os.environ.get('API_KEY')

# request object is a pytest object for managing the context of the test
@pytest.fixture(scope="module")
def Weather(request, volttron_instance):
    config = {
        "API_KEY": API_KEY
    }
    # using the volttron_instance fixture (passed in by volttrontesting fixtures), we␣
↪can install an agent
    # on the platform to test against
    agent = volttron_instance.install_agent(
        vip_identity=identity,
        agent_dir=source,
        start=False,
        config_file=config)

    volttron_instance.start_agent(agent)
    gevent.sleep(3)

    def stop_agent():
```

(continues on next page)

(continued from previous page)

```
        print("stopping weather service")
        if volttron_instance.is_running():
            volttron_instance.stop_agent(agent)
    # here we used the passed request object to add something to happen when the test␣
↪is finished
    request.addfinalizer(stop_agent)
    return agent, identity

# Here we create a really simple agent which has only the core functionality, which␣
↪we can use for Pub/Sub
# or JSON/RPC
@pytest.fixture(scope="module")
def query_agent(request, volttron_instance):
    # Create the simple agent
    agent = volttron_instance.build_agent()

    def stop_agent():
        print("In teardown method of query_agent")
        agent.core.stop()

    request.addfinalizer(stop_agent)
    return agent

# pass the 2 fixtures to our test, then we can run the test
def test_weather_success(Weather, query_agent):
    query_data = query_agent.vip.rpc.call(identity, 'get_current_weather', locations).
↪get(timeout=30)
    assert query_data.get("weather_results") = "Its sunny today!"
```

For more integration test examples, it is recommended to take a look at some of the VOLTTRON core agents, such as historian agents and weather service agents.

Using Docker for Limited-Integration Testing

If you want to run limited-integration tests which do not require the setup of a volttron system, you can use Docker containers to mimic dependencies of an agent. The *volttrontesting/fixtures/docker_wrapper.py* module provides a convenient function to create docker containers for use in limited-integration tests. For example, suppose that you had an agent with a dependency on a MySQL database. If you want to test the connection between the Agent and the MySQL dependency, you can create a Docker container to act as a real MySQL database. Below is an example:

```
from volttrontesting.fixtures.docker_wrapper import create_container
from UserAgent import UserAgentClass

def test_docker_wrapper_example():
    ports_config = {'3306/tcp': 3306}
    with create_container("mysql:5.7", ports=ports_config) as container:
        init_database(container)
        agent = UserAgent(ports_config)

        results = agent.some_method_that_talks_to_container()
```

Running your Tests and Debugging

Pytest can be run from the command line to run a test module.

```
pytest <path to module to be tested>
```

If using marks, you can add `-m <mark>` to specify your testing subset, and -s can be used to suppress standard output. For more information about optional arguments you can type *pytest –help* into your command line interface to see the full list of options.

Testing output should look something like this:

```
(volttron) <user>@<host>:~/volttron$ pytest services/core/SQLHistorian/
======================================================== test session starts␣
↪=========================================================
platform linux -- Python 3.6.9, pytest-5.4.1, py-1.8.1, pluggy-0.13.1 -- /home/<user>/
↪volttron/env/bin/python
cachedir: .pytest_cache
rootdir: /home/<user>/volttron, inifile: pytest.ini
plugins: timeout-1.3.4
timeout: 240.0s
timeout method: signal
timeout func_only: False
collected 2 items

services/core/SQLHistorian/tests/test_sqlitehistorian.py::test_sqlite_
↪timeout[volttron_3-volttron_instance0] ERROR            [ 50%]
services/core/SQLHistorian/tests/test_sqlitehistorian.py::test_sqlite_
↪timeout[volttron_3-volttron_instance1] PASSED           [100%]

=============================================================== ERRORS␣
↪==============================================================
________________________________ ERROR at setup of test_sqlite_timeout[volttron_3-
↪volttron_instance0] ________________________________

request = <SubRequest 'volttron_instance' for <Function test_sqlite_timeout[volttron_
↪3-volttron_instance0]>>, kwargs = {}
address = 'tcp://127.0.0.113:5846'

    @pytest.fixture(scope="module",
                    params=[
                        dict(messagebus='zmq', ssl_auth=False),
                        pytest.param(dict(messagebus='rmq', ssl_auth=True), marks=rmq_
↪skipif),
                    ])
    def volttron_instance(request, **kwargs):
        """Fixture that returns a single instance of volttron platform for testing

        @param request: pytest request object
        @return: volttron platform instance
        """
        address = kwargs.pop("vip_address", get_rand_vip())
        wrapper = build_wrapper(address,
                                messagebus=request.param['messagebus'],
                                ssl_auth=request.param['ssl_auth'],
>                               **kwargs)

address    = 'tcp://127.0.0.113:5846'
kwargs     = {}
request    = <SubRequest 'volttron_instance' for <Function test_sqlite_
↪timeout[volttron_3-volttron_instance0]>>
```

(continues on next page)

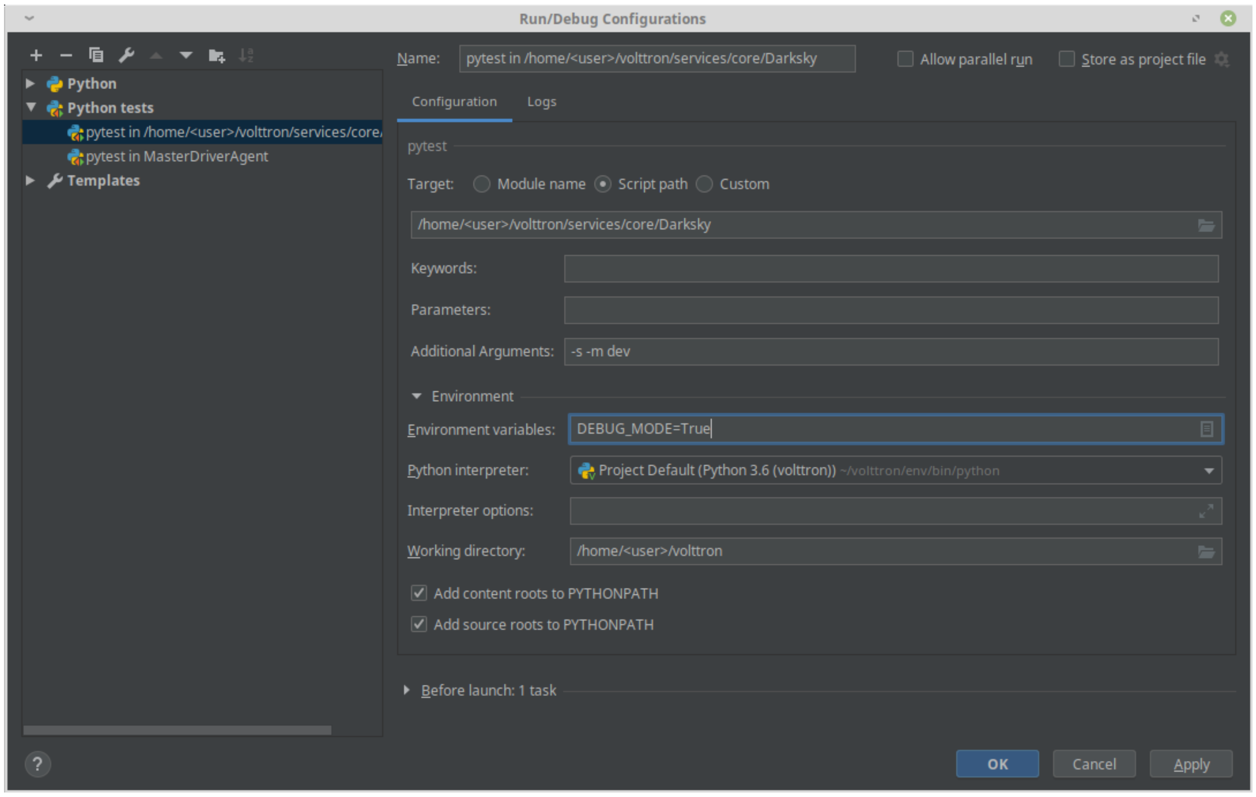
(continued from previous page)

```
volttrontesting/fixtures/volttron_platform_fixtures.py:106:
```

Running Tests Via PyCharm

To run our Pytests using PyCharm, we'll need to create a run configuration. To do so, select "edit configurations" from the "Run" menu (or if using the toolbar UI element you can click on the run configurations dropdown to select "edit configurations"). Use the plus symbol at the top right of the pop-up menu, scroll to "Python Tests" and expand this menu and select "pytest". This will create a run configuration, which will then need to be filled out. We recommend the following in general:

- Set the "Script Path" radio and fill the form with the path to your module. Pytest will run any tests in that module using the discovery process described above (and any marks if specified)
- In the interpreter dropdown, select the VOLTTRON virtual environment - this will likely be your project default
- Set the working directory to the VOLTTRON root directory
- Add any environment variables - For debugging, add variable "DEBUG_MODE" = True or "DEBUG" 1
- Add any optional arguments (-s will prevent standard output from being displayed in the console window, -m is used to specify a mark)



Name: pytest in /home/<user>/volttron/services/core/Darksky

pytest

Target: Module name (●) Script path Custom

/home/<user>/volttron/services/core/Darksky

Keywords:

Parameters:

Additional Arguments: -s -m dev

Environment

Environment variables: DEBUG_MODE=True

Python interpreter: Project Default (Python 3.6 (volttron)) ~/volttron/env/bin/python

Interpreter options:

Working directory: /home/<user>/volttron

✓ Add content roots to PYTHONPATH

✓ Add source roots to PYTHONPATH

PyCharm testing instructions

More information on testing in Python

Developing Historian Agents

VOLTTRON provides a convenient base class for developing new historian agents. The base class automatically performs a number of important functions:

- subscribes to all pertinent topics
- caches published data to disk until it is successfully recorded to a historian
- creates the public facing interface for querying results
- spells out a simple interface for concrete implementation to meet to make a working Historian Agent
- breaks data to publish into reasonably sized chunks before handing it off to the concrete implementation for publication. The size of the chunk is configurable
- sets up a separate thread for publication. If publication code needs to block for a long period of time (up to 10s of seconds) this will no disrupt the collection of data from the bus or the functioning of the agent itself

The VOLTTRON repository provides several *historians* which can be deployed without modification.

BaseHistorian

All Historians must inherit from the BaseHistorian class in volttron.platform.agent.base_historian and implement the following methods:

publish_to_historian(self, to_publish_list)

This method is called by the BaseHistorian class when it has received data from the message bus to be published. *to_publish_list* is a list of records to publish in the form:

```
[
    {
        '_id': 1,
        'timestamp': timestamp,
        'source': 'scrape',
        'topic': 'campus/building/unit/point',
        'value': 90,
        'meta': {'units':'F'}
    }
    {
        ...
    }
]
```

- **_id** - ID of the record used for internal record tracking. All IDs in the list are unique
- **timestamp** - Python datetime object of the time data was published at timezone UTC
- **source** - Source of the data: can be scrape, analysis, log, or actuator
- **topic** - Topic data was published on, topic prefix's such as "device" are dropped
- **value** - Value of the data, can be any type.
- **meta** - Metadata for the value, some sources will omit this entirely.

For each item in the list the concrete implementation should attempt to publish (or discard if non-publishable) every item in the list. Publication should be batched if possible. For every successfully published record and every record that is to be discarded because it is non-publishable the agent must call *report_handled* on those records. Records that should be published but were not for whatever reason require no action. Future calls to *publish_to'_historian* will include these unpublished records. *publish_to_historian* is always called with the oldest unhandled records. This allows the historian to no lose data due to lost connections or other problems.

As a convenience *report_all_handled* can be called if all of the items in *published_list* were successfully handled.

query_topic_list(self)

Must return a list of all unique topics published.

query_historian(self, topic, start=None, end=None, skip=0, count=None, order=None)

This function must return the results of a query in the form:

```
{"values": [(timestamp1: value1), (timestamp2: value2), ...],
 "metadata": {"key1": value1, "key2": value2, ...}}
```

metadata is not required (The caller will normalize this to { } for you if you leave it out)

- **topic** - the topic the user is querying for
- **start** - datetime of the start of the query, *None* for the beginning of time
- **end** - datetime of the end of of the query, *None* for the end of time
- **skip** - skip this number of results (for pagination)
- **count** - return at maximum this number of results (for pagination)
- **order** - *FIRST_TO_LAST* for ascending time stamps, *LAST_TO_FIRST* for descending time stamps

historian_setup(self)

Implementing this is optional. This function is run on the same thread as the rest of the concrete implementation at startup. It is meant for connection setup.

Example Historian

An example historian can be found in the *examples/CSVHistorian* directory in the VOLTTRON repository. This example historian uses a CSV file as the persistent data store. It is recommended to use this agent as a reference for developing new historian agents.

Developing Market Agents

VOLTTRON provides a convenient base class for developing new market agents. The base class automatically subscribes to all pertinent topics, and spells out a simple interface for concrete implementation to make a working Market Agent.

Markets are implemented by the Market Service Agent which is a core service agent. The Market Service Agent publishes information on several topics to which the base agent automatically subscribes. The base agent also provides all the methods you will need to interact with the Market Service Agent to implement your market transactions.

MarketAgent

All Market Agents must inherit from the MarketAgent class in *volttron.platform.agent.base_market_agent* and call the following method:

```
self.join_market(market_name, buyer_seller, reservation_callback, offer_callback,␣
↪aggregate_callback, price_callback, error_callback)
```

This method causes the market agent to join a single market. If the agent wishes to participate in several markets it may be called once for each market. The first argument is the name of the market to join and this name must be unique across the entire volttron instance because all markets are implemented by a single market service agent for each volttron instance. The second argument describes the role that this agent wished to play in this market. The value is imported as:

```
from volttron.platform.agent.base_market_agent.buy_sell import BUYER, SELLER
```

Arguments 3-7 are callback methods that the agent may implement as needed for the agent's participation in the market.

The Reservation Callback

```
reservation_callback(self, timestamp, market_name, buyer_seller)
```

This method is called when it is time to reserve a slot in the market for the current market cycle. If this callback is not registered a slot is reserved for every market cycle. If this callback is registered it is called for each market cycle and returns *True* if a reservation is wanted and *False* if a reservation is not wanted.

The name of the market and the roll being played are provided so that a single callback can handle several markets. If the agent joins three markets with the same reservation callback routine it will be called three times with the appropriate market name and buyer/seller role for each call. The MeterAgent example illustrates the use of this of this method and how to determine whether to make an offer when the reservation is refused.

A market will only exist if there are reservations for at least one buyer or one seller. If the market fails to achieve the minimum participation the error callback will be called. If only buyers or only sellers make reservations any offers will be rejected with the reason that the market has not formed.

The Offer Callback

```
offer_callback(self, timestamp, market_name, buyer_seller)
```

If the agent has made a reservation for the market and a callback has been registered this callback is called. If the agent wishes to make an offer at this time the market agent computes either a supply or a demand curve as appropriate and offers the curve to the market service by calling the `make_offer` method.

The name of the market and the roll being played are provided so that a single callback can handle several markets.

For each market joined either an offer callback, an aggregate callback, or a cleared price callback is required.

The Aggregate Callback

```
aggregate_callback(self, timestamp, market_name, buyer_seller, aggregate_curve)
```

When a market has received all its buy offers it calculates an aggregate demand curve. When the market receives all of its sell offers it calculates an aggregate supply curve. This callback delivers the aggregate curve to the market agent whenever the appropriate curve becomes available.

If the market agent wants to use this opportunity to make an offer on this or another market it would do that using the `make_offer` method.

- If the aggregate demand curve is received, only a supply offer may be submitted for this market
- If the aggregate supply curve is received, only make a demand offer will be accepted by this market.

You may use this information to make an offer on another market; The example AHUAgent does this. The name of the market and the roll being played are provided so that a single callback can handle several markets.

For each market joined, either an offer callback, an aggregate callback, or a cleared price callback is required.

The Price Callback

```
price_callback(self, timestamp, market_name, buyer_seller, price, quantity)
```

This callback is called when the market clears. If the market agent wants to use this opportunity to make an offer on this or another market it would do that using the `make_offer` method.

Once the market has cleared you can not make an offer on that market. Again, you may use this information to make an offer on another market as in the example AHUAgent. The name of the market and the roll being played are provided so that a single callback can handle several markets.

For each market joined either an offer callback, an aggregate callback, or a cleared price callback is required.

The Error Callback

```
error_callback(self, timestamp, market_name, buyer_seller, error_code, error_message,␣
↪aux)
```

This callback is called when an error occurs isn't in response to an RPC call. The error codes are documented in:

```
from volttron.platform.agent.base_market_agent.error_codes import NOT_FORMED, SHORT_
↪OFFERS, BAD_STATE, NO_INTERSECT
```

- NOT_FORMED - If a market fails to form this will be called at the offer time.
- SHORT_OFFERS - If the market doesn't receive all its offers this will be called while clearing the market.
- BAD_STATE - This indicates a bad state transition while clearing the market and should never happen, but may be called while clearing the market.
- NO_INTERSECT - If the market fails to clear this would be called while clearing the market and an auxillary array will be included. The auxillary array contains comparisons between the supply max, supply min, demand max and demand min. They allow the market client to make determinations about why the curves did not intersect that may be useful.

The error callback is optional, but highly recommended.

Example Agents

Some example agents are included with the platform to help explore its features. These agents represent concrete implementations of important agent sub-types such as Historians or Weather Agents, or demonstrate a development pattern for accomplishing common tasks.

More complex agents contributed by other researchers can also be found in the examples directory. It is recommended that developers new to VOLTTRON understand the example agents first before diving into the other agents.

C Agent

The C Agent uses the *ctypes* module to load a shared object into memory so its functions can be called from Python.

There are two versions of the C Agent:

- A standard agent that can be installed with the agent installation process
- A driver which can can be controlled using the Master Driver Agent

Building the Shared Object

The shared object library must be built before installing C Agent examples. Running `make` in the C Agent source directory will compile the provided C code using the position independent flag, a requirement for creating shared objects.

Files created by make can be removed by running

```
make clean
```

Agent Installation

After building the shared object library the standard agent can be installed with the `scripts/install-agent.py` script:

```
python scripts/install-agent.py -s examples/CAgent
```

The other is a driver interface for the Master Driver. To use the C driver, the driver code file must be moved into the Master Driver's *interfaces* directory:

```
examples/CAgent/c_agent/driver/cdriver -> services/core/MasterDriverAgent/
↪master_driver/interfaces
```

The C Driver configuration tells the interface where to find the shared object. An example is available in the C Agent's *driver* directory.

Config Actuation Example

The Config Actuation example attempts to set points on a device when files are added or updated in its *configuration store*.

Configuration

The name of a configuration file must match the name of the device to be actuated. The configuration file is a JSON dictionary of point name and value pairs. Any number of points on the device can be listed in the config.

```
{
    "point0": value,
    "point1": value
}
```

CSV Historian

The CSV Historian Agent is an example historian agent that writes device data to the CSV file specified in the configuration file.

Explanation of CSV Historian

The Utils module of the VOLTTRON platform includes functions for setting up global logging for the platform:

```
utils.setup_logging()
_log = logging.getLogger(__name__)
```

The `historian` method is called by `utils.vip_main` when the agents is started (see below). `utils.vip_main` expects a callable object that returns an instance of an Agent. This method of dealing with a configuration file and instantiating an Agent is common practice.

```
def historian(config_path, **kwargs):
    if isinstance(config_path, dict):
        config_dict = config_path
    else:
        config_dict = utils.load_config(config_path)

    output_path = config_dict.get("output", "~/historian_output.csv")

    return CSVHistorian(output_path = output_path, **kwargs)
```

All historians must inherit from *BaseHistorian*. The *BaseHistorian* class handles the capturing and caching of all device, logging, analysis, and record data published to the message bus.

```
class CSVHistorian(BaseHistorian):
```

The Base Historian creates a separate thread to handle publishing data to the data store. In this thread the Base Historian calls two methods on the created historian, `historian_setup` and `publish_to_historian`.

The Base Historian created the new thread in it's `__init__` method. This means that any instance variables must assigned in `__init__` before calling the Base Historian's `__init__` method.

```
def __init__(self, output_path="", **kwargs):
    self.output_path = output_path
    self.csv_dict = None
    super(CSVHistorian, self).__init__(**kwargs)
```

Historian setup is called shortly after the new thread starts. This is where a Historian sets up a connect the first time. In our example we create the *Dictwriter* object that we will use to create and add lines to the CSV file.

We keep a reference to the file object so that we may flush its contents to disk after writing the header and after we have written new data to the file.

The CSV file we create will have 4 columns: *timestamp*, *source*, *topic*, and *value*.

```
def historian_setup(self):
    self.f = open(self.output_path, "wb")
    self.csv_dict = csv.DictWriter(self.f, ["timestamp", "source", "topic", "value"])
    self.csv_dict.writeheader()
    self.f.flush()
```

`publish_to_historian` is called when data is ready to be published. It is passed a list of dictionaries. Each dictionary contains a record of a single value that was published to the message bus.

The dictionary takes the form:

```
{
    '_id': 1,
    'timestamp': timestamp1.replace(tzinfo=pytz.UTC), #Timestamp in UTC
    'source': 'scrape', #Source of the data point.
    'topic': "pnnl/isb1/hvac1/thermostat", #Topic that published to without prefix.
    'value': 73.0, #Value that was published
    'meta': {"units": "F", "tz": "UTC", "type": "float"} #Meta data published with␣
↪the topic
}
```

Once the data is written to the historian we call `self.report_all_handled()` to inform the *BaseHistorian* that all data we received was successfully published and can be removed from the cache. Then we can flush the file to ensure that the data is written to disk.

```
def publish_to_historian(self, to_publish_list):
    for record in to_publish_list:
        row = {}
        row["timestamp"] = record["timestamp"]

        row["source"] = record["source"]
        row["topic"] = record["topic"]
        row["value"] = record["value"]

        self.csv_dict.writerow(row)

    self.report_all_handled()
    self.f.flush()
```

This agent does not support the Historian Query interface.

Agent Testing

The CSV Historian can be tested by running the included *launch_my_historian.sh* script.

Agent Installation

This Agent may be installed on the platform using the standard method.

Data Publisher

This is a simple agent that plays back data either from the config store or a CSV to the configured topic. It can also provide basic emulation of the Actuator Agent for testing agents that expect to be able to set points on a device in response to device publishes.

Installation notes

In order to simulate the actuator you must install the agent with the VIP identity of *platform.actuator*. If an an actuator is already installed on the platform, this will cause VIP identity conflicts. To install the agent, the agent install script can be used:

```
python scripts/install-agent.py -s examples/DataPublisher -c <config file>
```

Configuration

```
{
    # basetopic can be devices, analysis, or custom base topic
    "basepath": "devices/PNNL/ISB1",

    # use_timestamp uses the included in the input_data if present.
    # Currently the column must be named `Timestamp`.
    "use_timestamp": true,

    # Only publish data at most once every max_data_frequency seconds.
    # Extra data is skipped.
    # The time windows are normalized from midnight.
    # ie 900 will publish one value for every 15 minute window starting from
    # midnight of when the agent was started.
    # Only used if timestamp in input file is used.
    "max_data_frequency": 900,

    # The meta data published with the device data is generated
    # by matching point names to the unittype_map.
    "unittype_map": {
        ".*Temperature": "Farenheit",
        ".*SetPoint": "Farenheit",
        "OutdoorDamperSignal": "On/Off",
        "SupplyFanStatus": "On/Off",
        "CoolingCall": "On/Off",
        "SupplyFanSpeed": "RPM",
        "Damper*.": "On/Off",
        "Heating*.": "On/Off",
        "DuctStatic*.": "On/Off"
    },
    # Path to input CSV file.
    # May also be a list of records or reference to a CSV file in the config store.
    # Large CSV files should be referenced by file name and not
    # stored in the config store.
    "input_data": "econ_test2.csv",
    # Publish interval in seconds
    "publish_interval": 1,
```

(continues on next page)

(continued from previous page)

```
    # Tell the playback to maintain the location a the file in the config store.
    # Playback will be resumed from this point
    # at agent startup even if this setting is changed to false before restarting.
    # Saves the current line in line_marker in the DataPublishers's config store
    # as plain text.
    # default false
    "remember_playback": true,

    # Start playback from 0 even if the line_marker configuration is set a non 0␣
↪value.
    # default false
    "reset_playback": false,

    # Repeat data from the start if this flag is true.
    # Useful for data that does not include a timestamp and is played back in real␣
↪time.
    "replay_data": false
}
```

CSV File Format

The CSV file must have a single header line. The column names are appended to the *basepath* setting in the configuration file and the resulting topic is normalized to remove extra‘ / characters. The values are all treated as floating point values and converted accordingly.

The corresponding device for each point is determined and the values are combined together to create an *all* topic publish for each device.

If a *Timestamp* column is in the input it may be used to set the timestamp in the header of the published data.

Timestamp	centrifugal_chiller/OutsideAirTemperature	centrifugal_chiller/DischargeAirTemperatureSetPoint
2012/05/19 05:07:00	0	56
2012/05/19 05:08:00	0	56
2012/05/19 05:09:00	0	56
2012/05/19 05:10:00	0	56
2012/05/19 05:11:00	0	56
2012/05/19 05:12:00	0	56
2012/05/19 05:13:00	0	56
2012/05/19 05:14:00	0	56
2012/05/19 05:15:00	0	56
2012/05/19 05:16:00	0	56
2012/05/19 05:17:00	0	56
2012/05/19 05:18:00	0	56
2012/05/19 05:19:00	0	56
2012/05/19 05:20:00	0	56
2012/05/19 05:21:00	0	56
2012/05/19 05:22:00	0	56
2012/05/19 05:23:00	0	56
2012/05/19 05:24:00	0	56
2012/05/19 05:25:00	48.78	56
2012/05/19 05:26:00	48.88	56

Timestamp	centrifugal_chiller/OutsideAirTemperature	centrifugal_chiller/DischargeAirTemperatureSetPoint
2012/05/19 05:27:00	48.93	56
2012/05/19 05:28:00	48.95	56
2012/05/19 05:29:00	48.92	56
2012/05/19 05:30:00	48.88	56
2012/05/19 05:31:00	48.88	56
2012/05/19 05:32:00	48.99	56
2012/05/19 05:33:00	49.09	56
2012/05/19 05:34:00	49.11	56
2012/05/19 05:35:00	49.07	56
2012/05/19 05:36:00	49.05	56
2012/05/19 05:37:00	49.09	56
2012/05/19 05:38:00	49.13	56
2012/05/19 05:39:00	49.09	56
2012/05/19 05:40:00	49.01	56
2012/05/19 05:41:00	48.92	56
2012/05/19 05:42:00	48.86	56
2012/05/19 05:43:00	48.92	56
2012/05/19 05:44:00	48.95	56
2012/05/19 05:45:00	48.92	56
2012/05/19 05:46:00	48.86	56
2012/05/19 05:47:00	48.78	56
2012/05/19 05:48:00	48.69	56
2012/05/19 05:49:00	48.65	56
2012/05/19 05:50:00	48.65	56
2012/05/19 05:51:00	48.65	56
2012/05/19 05:52:00	48.61	56
2012/05/19 05:53:00	48.59	56
2012/05/19 05:54:00	48.55	56
2012/05/19 05:55:00	48.63	56
2012/05/19 05:56:00	48.76	56
2012/05/19 05:57:00	48.95	56
2012/05/19 05:58:00	49.24	56
2012/05/19 05:59:00	49.54	56
2012/05/19 06:00:00	49.71	56
2012/05/19 06:01:00	49.79	56
2012/05/19 06:02:00	49.94	56
2012/05/19 06:03:00	50.13	56
2012/05/19 06:04:00	50.18	56
2012/05/19 06:05:00	50.15	56

DDS Agent

The DDS example agent demonstrates VOLTTRON's capacity to be extended with tools and libraries not used in the core codebase. DDS is a messaging platform that implements a publish-subscribe system for well defined data types.

This agent example is meant to be run the command line, as opposed to installing it like other agents. From the *examples/DDSAgent* directory, the command to start it is:

```
$ AGENT_CONFIG=config python -m ddsagent.agent
```

The *rticonnextdds-connector* library needs to be installed for this example to function properly. We'll retrieve it from GitHub since it is not available through Pip. Download the source with:

```
$ wget https://github.com/rticommunity/rticonnextdds-connector/archive/master.zip
```

and unpack it in *examples/DDSAgent/ddsagent* with:

```
$ unzip master.zip
```

The `demo_publish()` output can be viewed with the *rtishapesdemo* available from RTI.

Configuration

Each data type that this agent will have access to needs to have an XML document defining its structure. The XML will include a participant name, publisher name, and a subscriber name. These are recorded in the configuration with the location on disk of the XML file.

```
{
    "square": {
        "participant_name": "MyParticipantLibrary::Zero",
        "xml_config_path": "./ddsagent/rticonnextdds-connector-master/examples/python/
↪ShapeExample.xml",
        "publisher_name": "MyPublisher::MySquareWriter",
        "subscriber_name": "MySubscriber::MySquareReader"
    }
}
```

Listener Agent

The ListenerAgent subscribes to all topics and is useful for testing that agents being developed are publishing correctly. It also provides a template for building other agents as it expresses the requirements of a platform agent.

Explanation of Listener Agent Code

Use `utils` to setup logging, which we'll use later.

```
utils.setup_logging()
_log = logging.getLogger(__name__)
```

The Listener agent extends (inherits from) the Agent class for its default functionality such as responding to platform commands:

```
class ListenerAgent(Agent):
    '''
    Listens to everything and publishes a heartbeat according to the
    heartbeat period specified in the settings module.
    '''
```

After the class definition, the Listener agent reads the configuration file, extracts the configuration parameters, and initializes any Listener agent instance variable. This is done through the agent's `__init__` method:

```
def __init__(self, config_path, **kwargs):
    super(ListenerAgent, self).__init__(**kwargs)
    self.config = utils.load_config(config_path)
    self._agent_id = self.config.get('agentid', DEFAULT_AGENTID)
    log_level = self.config.get('log-level', 'INFO')
    if log_level == 'ERROR':
        self._logfn = _log.error
    elif log_level == 'WARN':
        self._logfn = _log.warn
    elif log_level == 'DEBUG':
        self._logfn = _log.debug
    else:
        self._logfn = _log.info
```

Next, the Listener agent will run its setup method. This method is tagged to run after the agent is initialized by the decorator `@Core.receiver('onsetup')`. This method accesses the configuration parameters, logs a message to the platform log, and sets the agent ID.

```
@Core.receiver('onsetup')
def onsetup(self, sender, **kwargs):
    # Demonstrate accessing a value from the config file
    _log.info(self.config.get('message', DEFAULT_MESSAGE))
    self._agent_id = self.config.get('agentid')
```

The Listener agent subscribes to all topics published on the message bus. Publish and subscribe interactions with the message bus are handled by the *PubSub* module located at *~/volttron/volttron/platform/vip/agent/subsystems/pubsub.py*.

The Listener agent uses an empty string to subscribe to all messages published. This is done in a decorator for simplifying subscriptions.

```
@PubSub.subscribe('pubsub', '')
def on_match(self, peer, sender, bus,  topic, headers, message):
    '''Use match_all to receive all messages and print them out.'''
    if sender == 'pubsub.compat':
        message = compat.unpack_legacy_message(headers, message)
    self._logfn(
    "Peer: %r, Sender: %r:, Bus: %r, Topic: %r, Headers: %r, "
    "Message: %r", peer, sender, bus, topic, headers, message)
```

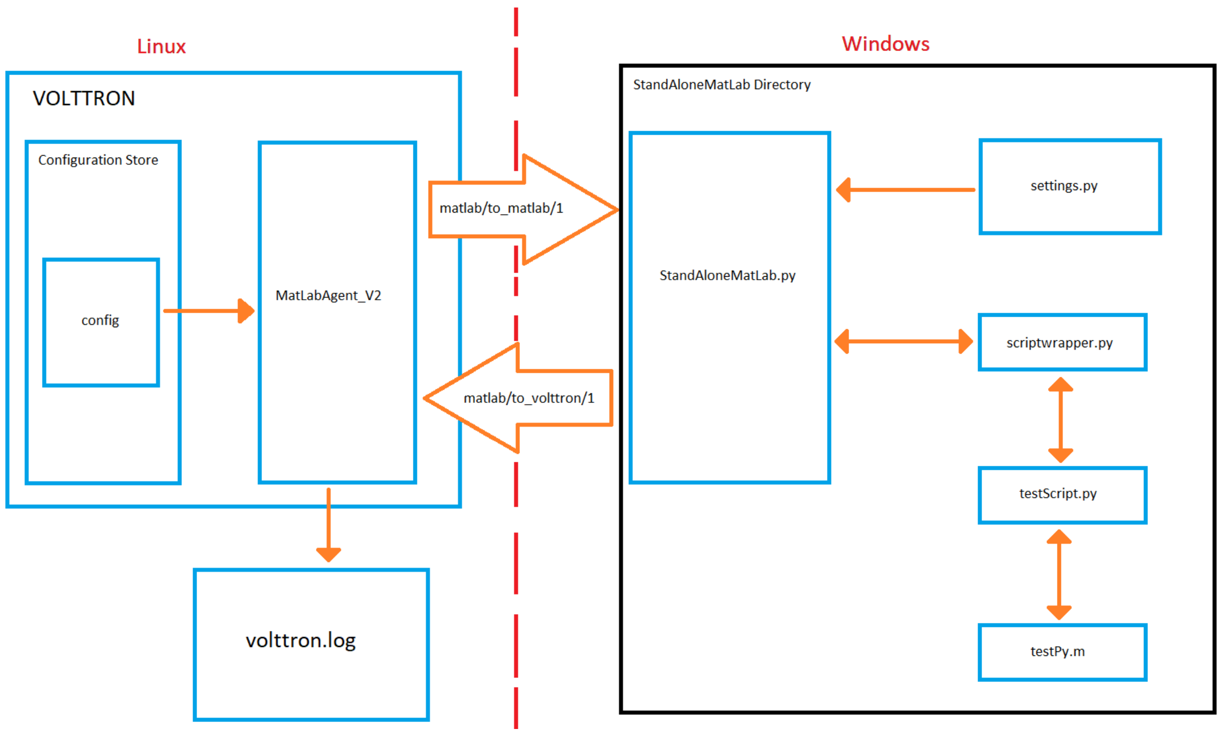
MatLab Agent

The MatLab agent and Matlab Standalone Agent together are example agents that allow for MatLab scripts to be run in a Windows environment and interact with the VOLTTRON platform running in a Linux environment.

The MatLab agent takes advantage of the config store to dynamically send scripts and commandline arguments across the message bus to one or more Standalone Agents in Windows. The Standalone Agent then executes the requested script and arguments, and sends back the results to the MatLab agent.

Overview of Matlab Agents

There are multiple components that are used for the MatLab agent. This diagram is to represent the components that are connected to the MatLab Agents. In this example, the scripts involved are based on the default settings in the MatLab Agent.



MatLabAgentV2

MatLabAgentV2 publishes the name of a python script along with any command line arguments that are needed for the script to the appropriate topic. The agent then listens on another topic, and whenever anything is published on this topic, it stores the message in the log file chosen when the VOLTTRON instance is started. If there are multiple standalone agents, the agent can send a a script to each of them, along with their own set of command line arguments. In this case, each script name and set of command line arguments should be sent to separate subtopics. This is done so that no matter how many standalone agents are in use, MatLabAgentV2 will record all of their responses.

```
class MatlabAgentV2(Agent):

    def __init__(self,script_names=[], script_args=[], topics_to_matlab=[],
            topics_to_volttron=None,**kwargs):

        super(MatlabAgentV2, self).__init__(**kwargs)
        _log.debug("vip_identity: " + self.core.identity)

        self.script_names = script_names
        self.script_args = script_args
        self.topics_to_matlab = topics_to_matlab
        self.topics_to_volttron = topics_to_volttron
        self.default_config = {"script_names": script_names,
                               "script_args": script_args,
                               "topics_to_matlab": topics_to_matlab,
                               "topics_to_volttron": topics_to_volttron}


        #Set a default configuration to ensure that self.configure is called␣
↪immediately to setup
        #the agent.
```

(continues on next page)

(continued from previous page)

```
        self.vip.config.set_default("config", self.default_config)
        #Hook self.configure up to changes to the configuration file "config".
        self.vip.config.subscribe(self.configure, actions=["NEW", "UPDATE"], pattern=
↪"config")

    def configure(self, config_name, action, contents):
        """
        Called after the Agent has connected to the message bus.
        If a configuration exists at startup this will be
        called before onstart.
        Is called every time the configuration in the store changes.
        """
        config = self.default_config.copy()
        config.update(contents)

        _log.debug("Configuring Agent")

        try:
            script_names = config["script_names"]
            script_args = config["script_args"]
            topics_to_matlab = config["topics_to_matlab"]
            topics_to_volttron = config["topics_to_volttron"]

        except ValueError as e:
            _log.error("ERROR PROCESSING CONFIGURATION: {}".format(e))
            return

        self.script_names = script_names
        self.script_args = script_args
        self.topics_to_matlab = topics_to_matlab
        self.topics_to_volttron = topics_to_volttron
        self._create_subscriptions(self.topics_to_volttron)

        for script in range(len(self.script_names)):
            cmd_args = ""
            for x in range(len(self.script_args[script])):
                cmd_args += ",{}".format(self.script_args[script][x])
            _log.debug("Publishing on: {}".format(self.topics_to_matlab[script]))
            self.vip.pubsub.publish('pubsub', topic=self.topics_to_matlab[script],
                    message="{}{}".format(self.script_names[script],cmd_args))
            _log.debug("Sending message: {}{}".format(self.script_names[script],cmd_
↪args))

        _log.debug("Agent Configured!")
```

For this example, the agent is publishing to the *matlab/to_matlab/1* topic, and is listening to the *matlab/to_volttron* topic. It is sending the script name *testScript.py* with the argument 20. These are the default values found in the agent, if no configuration is loaded.

```
script_names = config.get('script_names', ["testScript.py"])
script_args = config.get('script_args', [["20"]])
topics_to_matlab = config.get('topics_to_matlab', ["matlab/to_matlab/1"])
topics_to_volttron = config.get('topics_to_volttron', "matlab/to_volttron/")
```

StandAloneMatLab.py

The *StandAloneMatLab.py* script is a standalone agent designed to be able to run in a Windows environment. Its purpose is to listen to a topic, and when something is published to this topic, it takes the message, and sends it to the `script_runner` function in *scriptwrapper.py*. This function processes the inputs, and then the output is published to another topic.

```
class StandAloneMatLab(Agent):
    '''The standalone version of the MatLab Agent'''

    @PubSub.subscribe('pubsub', _topics['volttron_to_matlab'])
    def print_message(self, peer, sender, bus, topic, headers, message):
        print('The Message is: ' + str(message))
        messageOut = script_runner(message)
        self.vip.pubsub.publish('pubsub', _topics['matlab_to_volttron'],␣
↪message=messageOut)
```

settings.py

The topic to listen to and the topic to publish to are defined in *settings.py*, along with the information needed to connect the Standalone Agent to the primary VOLTTRON instance. These should be the same topics that the MatLabAgentV2 is publishing and listening to, so that the communication can be successful. To connect the Standalone Agent to the primary VOLTTRON instance, the IP address and port of the instance are needed, along with the server key.

```
_topics = {
        'volttron_to_matlab': 'matlab/to_matlab/1',
        'matlab_to_volttron': 'matlab/to_volttron/1'
        }

# The parameters dictionary is used to populate the agent's
# remote vip address.
_params = {
        # The root of the address.
        # Note:
        # 1. volttron instance should be configured to use tcp. use command vcfg
        # to configure
        'vip_address': 'tcp://192.168.56.101',
        'port': 22916,

        # public and secret key for the standalone_matlab agent.
        # These can be created using the command:  volttron-ctl auth keypair
        # public key should also be added to the volttron instance auth
        # configuration to enable standalone agent access to volttron instance. Use
        # command 'vctl auth add' Provide this agent's public key when prompted
        # for credential.

        'agent_public': 'dpu13XKPvGB3XJNVUusCNn2U0kIWcuyDIP5J8mAgBQ0',
        'agent_secret': 'Hlya-6BvfUot5USdeDHZ8eksDkWgEEHABs1SELmQhMs',

        # Public server key from the remote platform.  This can be
        # obtained using the command:
        # volttron-ctl auth serverkey
        'server_key': 'QTIzrRGQ0-b-37AbEYDuMA0l2ETrythM2V1ac0v9CTA'

}
```

(continues on next page)

(continued from previous page)

```
def remote_url():
        return "{vip_address}:{port}?serverkey={server_key}" \
                "&publickey={agent_public}&" \
                "secretkey={agent_secret}".format(**_params)
```

The primary VOLTTRON instance will then need to add the public key from the Standalone Agent. In this example, the topic that the Standalone Agent is listening to is *matlab/to_matlab/1*, and the topic it is publishing to is *matlab/to_volttron/1*.

scriptwrapper.py

Scriptwrapper.py contains the script_runner function. The purpose of this function is to take in a string that contains a Python script and command line arguments separated by commas. This string is parsed and passed to the system arguments, which allows the script sent to the function to use the command line arguments. The function then redirects standard output to a *StringIO* file object, and then attempts to execute the script. If there are any errors with the script, the error that is generated is returned to the standalone agent. Otherwise, the file object stores the output from the script, is converted to a string, and is sent to the standalone agent. In this example, the script that is to be run is *testScript.py*.

```
#Script to take in a string, run the program,
#and output the results of the command as a string.

import time
import sys
from io import StringIO


def script_runner(message):
    original = sys.stdout
#     print(message)
#     print(sys.argv)
    sys.argv = message.split(',')
#     print(sys.argv)

    try:
        out = StringIO()
        sys.stdout = out
        exec(open(sys.argv[0]).read())
        sys.stdout = original
        return out.getvalue()
    except Exception as ex:
        out = str(ex)
        sys.stdout = original
        return out
```

Note: The script that is to be run needs to be in the same folder as the agent and the *scriptwrapper.py* script. The *script_runner* function needs to be edited if it is going to call a script at a different location.

testScript.py

This is a very simple test script designed to demonstrate the calling of a MatLab function from within Python. First it initializes the MatLab engine for Python. It then takes in a single command line argument, and passes it to the MatLab function *testPy.m*. If no arguments are sent, it will send 0 to the *testPy.m* function. It then prints the result of the *testPy.m* function. In this case, since standard output is being redirected to a file object, this is how the result is passed from this function to the Standalone Agent.

```
import matlab.engine
import sys


eng = matlab.engine.start_matlab()

if len(sys.argv) == 2:
    result = eng.testPy(float(sys.argv[1]))
else:
    result = eng.testPy(0.0)

print(result)
```

testPy.m

This MatLab function is a very simple example, designed to show a function that takes an argument, and produces an array as the output. The input argument is added to each element in the array, and the entire array is then returned.

```
function out = testPy(z)
x = 1:100
out = x + z
end
```

Setup on Linux

1. Setup and run VOLTTRON from develop branch using instructions *here*.
2. Configure volttron instance using the `vcfg` command. When prompted for the vip address use `tcp://<ip address of the linux machine>`. This is necessary to enable volttron communication with external processes.

   Note: If you are running VOLTTRON from within VirtualBox, jit would be good to set one of your adapters as a *Host-only* adapter. This can be done within the VM's settings, under the *Network* section. Once this is done, use this IP for the VIP address.

3. Update the configuration for MatLabAgent_v2 at <*volttron source dir*>*/example/MatLabAgent_v2/config*.

   The configuration file for the MatLab agent has four variables.

   1. script_names
   2. script_args
   3. topics_to_matlab
   4. topics_to_volttron

An example config file included with the folder.

```
{
  # VOLTTRON config files are JSON with support for python style comments.
  "script_names": ["testScript.py"],
  "script_args": [["20"]],
  "topics_to_matlab": ["matlab/to_matlab/1"],
  "topics_to_volttron": "matlab/to_volttron/"
}
```

To edit the configuration, the format should be as follows:

```
{
  "script_names": ["script1.py", "script2.py", "..."],
  "script_args": [["arg1","arg2"], ["arg1"], ["..."]],
  "topics_to_matlab": ["matlab/to_matlab/1", "matlab/to_matlab/2", "..."],
  "topics_to_volttron": "matlab/to_volttron/"
}
```

The config requires that each script name lines up with a set of commandline arguments and a topic. A commandline argument must be included, even if it is not used. The placement of brackets are important, even when only communicating with one standalone agent.

For example, if only one standalone agent is used, and no command line arguments are in place, the config file may look like this.

```
{
  "script_names": ["testScript.py"],
  "script_args": [["0"]],
  "topics_to_matlab": ["matlab/to_matlab/1"],
  "topics_to_volttron": "matlab/to_volttron/"
}
```

4. Install MatLabAgent_v2 and start agent (from volttron root directory)

```
python ./scripts/install-agent.py -s examples/MatLabAgent_v2 --start
```

Note: The MatLabAgent_v2 publishes the command to be run to the message bus only on start or on a configuration update. Once we configure the *standalone_matlab* agent on the Windows machine, we will send a configuration update to the running MatLabAgent_v2. The configuration would contain the topics to which the Standalone Agent is listening to and will be publishing result to.

See also:

The MatLab agent uses the configuration store to dynamically change inputs. More information on the config store and how it used can be found here.

- *VOLTTRON Configuration Store*
- *Agent Configuration Store*
- *Agent Configuration Store Interface*

5. Run the below command and make a note of the server key. This is required for configuring the stand alone agent on Windows. (This is run on the linux machine)

```
vctl auth serverkey
```

Setup on Windows

Install pre-requisites

1. Install Python3.6 64-bit from the Python website.
2. Install the MatLab engine from MathWorks.

> **Warning:** The MatLab engine for Python only supports certain version of Python depending on the version of MatLab used. Please check here to see if the current version of MatLab supports your version of Python.

Note: At this time, you may want to verify that you are able to communicate with your Linux machine across your network. The simplest method would be to open up the command terminal and use `ping <ip of Linux machine>`, and `telnet <ip of Linux machine> <port of volttron instance, default port is 22916>`. Please make sure that the port is opened for outside access.

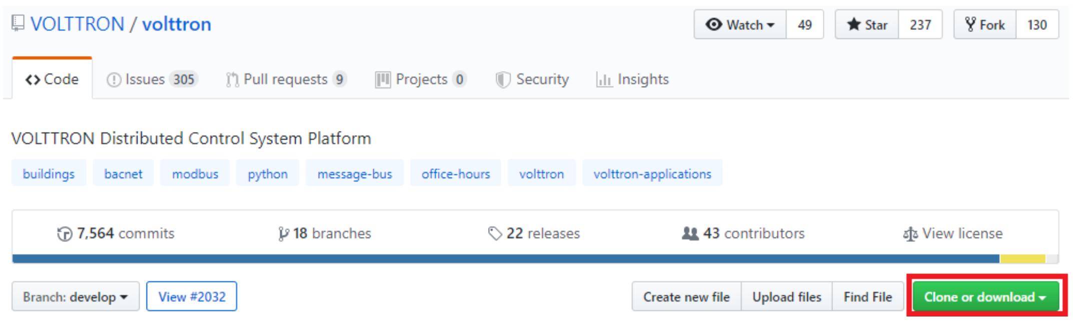
Install Standalone MatLab Agent

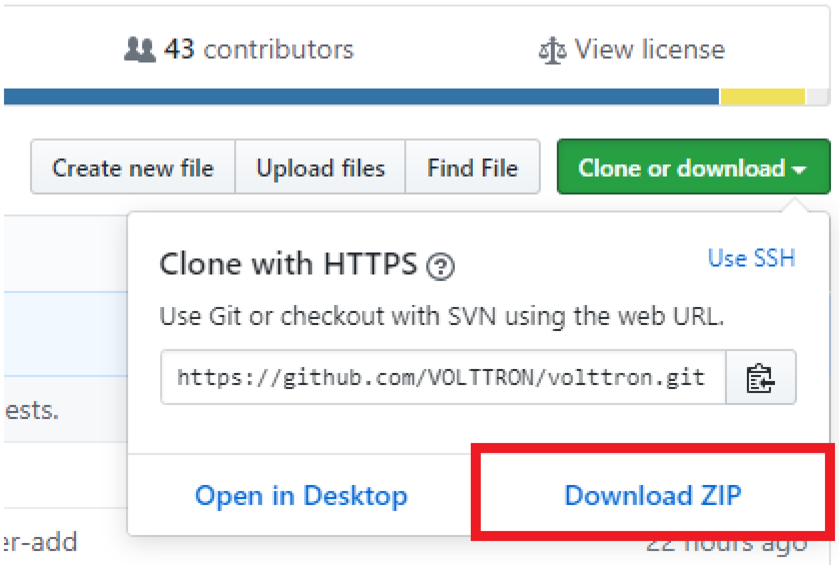
The standalone MatLab agent is designed to be usable in a Windows environment.

> **Warning:** VOLTTRON is not designed to run in a Windows environment. Outside of cases where it is stated to be usable in a Windows environment, it should be assumed that it will **NOT** function as expected.

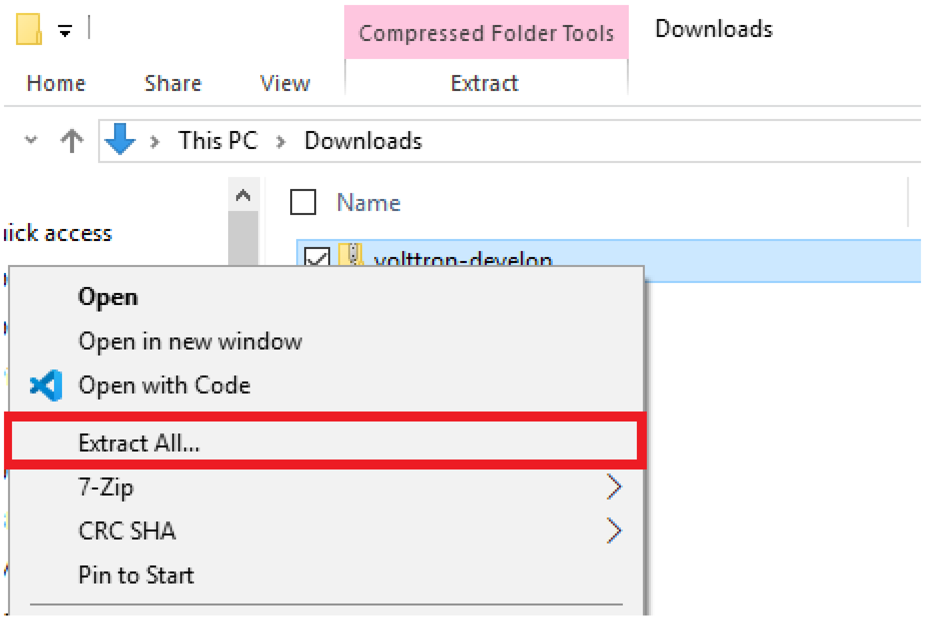
1. Download VOLTTRON

   Download the VOLTTRON develop repository from Github. Download the zip from GitHub.

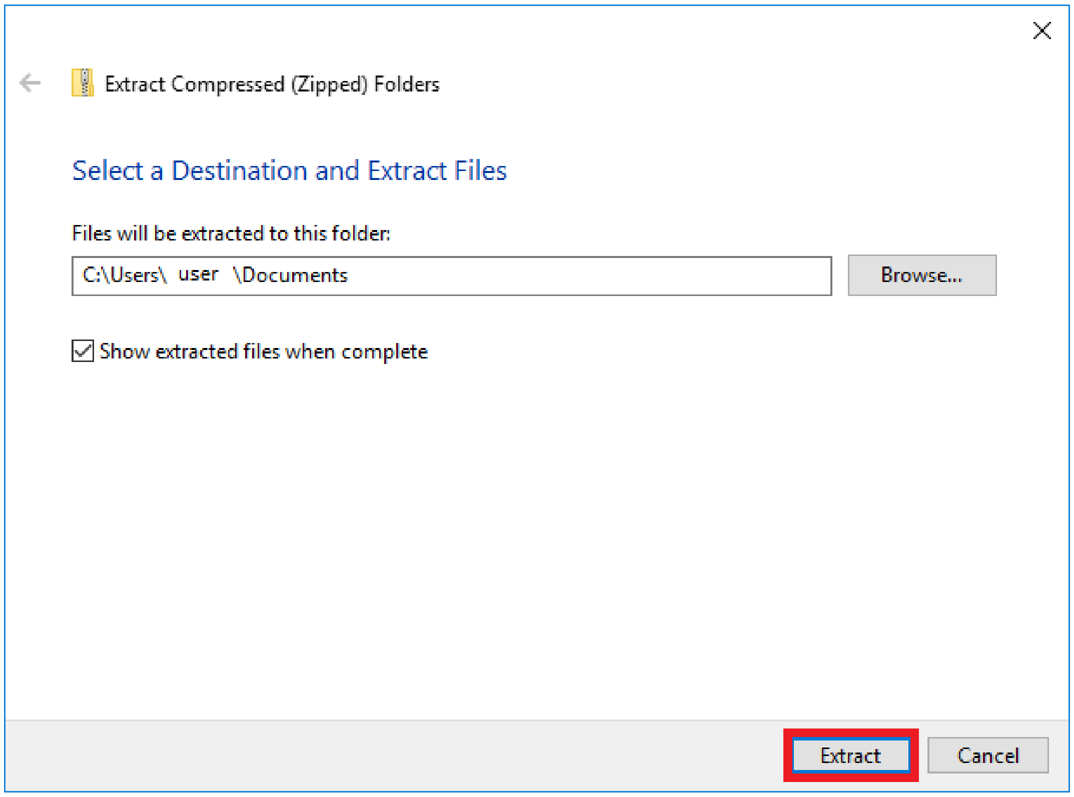




Once the zipped file has been downloaded, go to your *Downloads* folder, right-click on the file, and select *Extract All. . .*



Choose a location for the extracted folder, and select "Extract"



Select a Destination and Extract Files

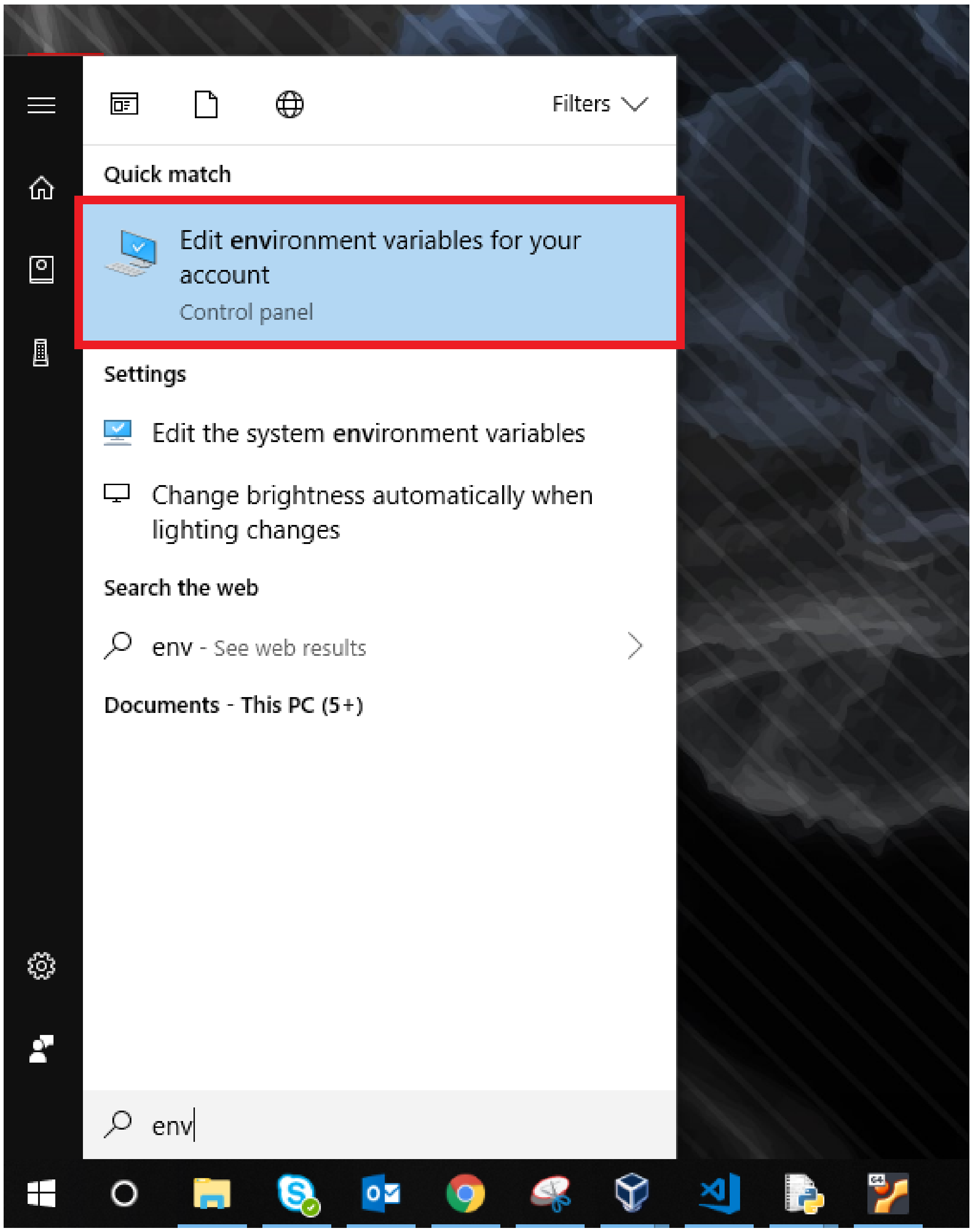
Files will be extracted to this folder:

C:\Users\ user \Documents

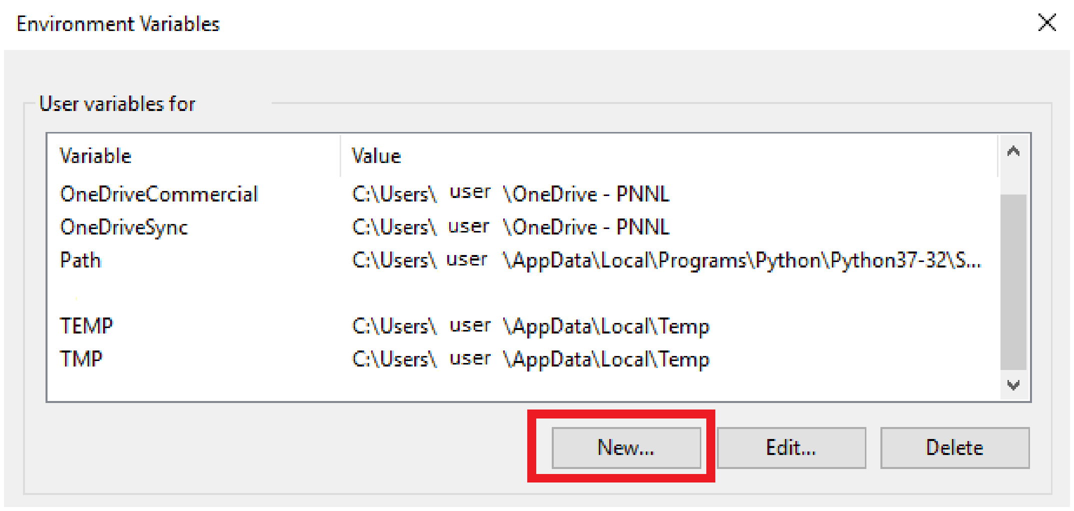
Show extracted files when complete: ✓

2. Setup the *PYTHONPATH*

   Open the Windows explorer, and navigate to *Edit environment variables for your account*.


Filters
Quick match
Edit environment variables for your account
Control panel
Settings
Edit the system environment variables
Change brightness automatically when lighting changes
Search the web
env - See web results
Documents - This PC (5+)
env

Select "New"

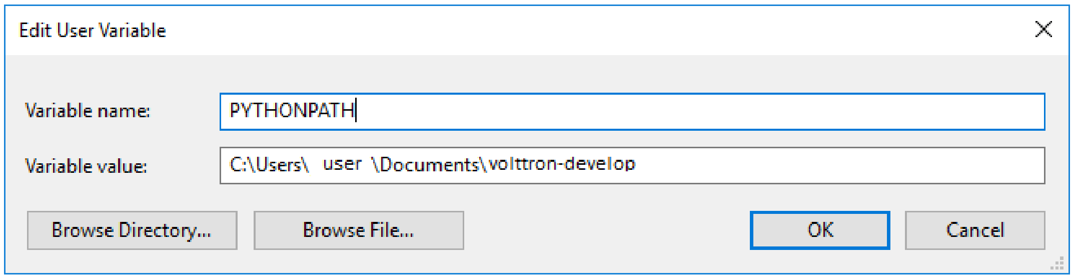


Environment Variables

User variables for

Variable	Value
OneDriveCommercial	C:\Users\ user \OneDrive - PNNL
OneDriveSync	C:\Users\ user \OneDrive - PNNL
Path	C:\Users\ user \AppData\Local\Programs\Python\Python37-32\S...
TEMP	C:\Users\ user \AppData\Local\Temp
TMP	C:\Users\ user \AppData\Local\Temp

For "Variable name" enter: `PYTHONPATH` For "Variable value" either browse to your VOLTTRON installation, or enter in the path to your VOLTTRON installation.



Edit User Variable

Variable name: PYTHONPATH

Variable value: C:\Users\ user \Documents\volttron-develop

Select *OK* twice.

3. Set Python version in MatLab

   Open your MatLab application. Run the command:

```
pyversion
```

   This should print the path to Python2.7. If you have multiple versions of python on your machine and *pyversion* points to a different version of Python, use:

```
pyversion /path/to/python.exe
```

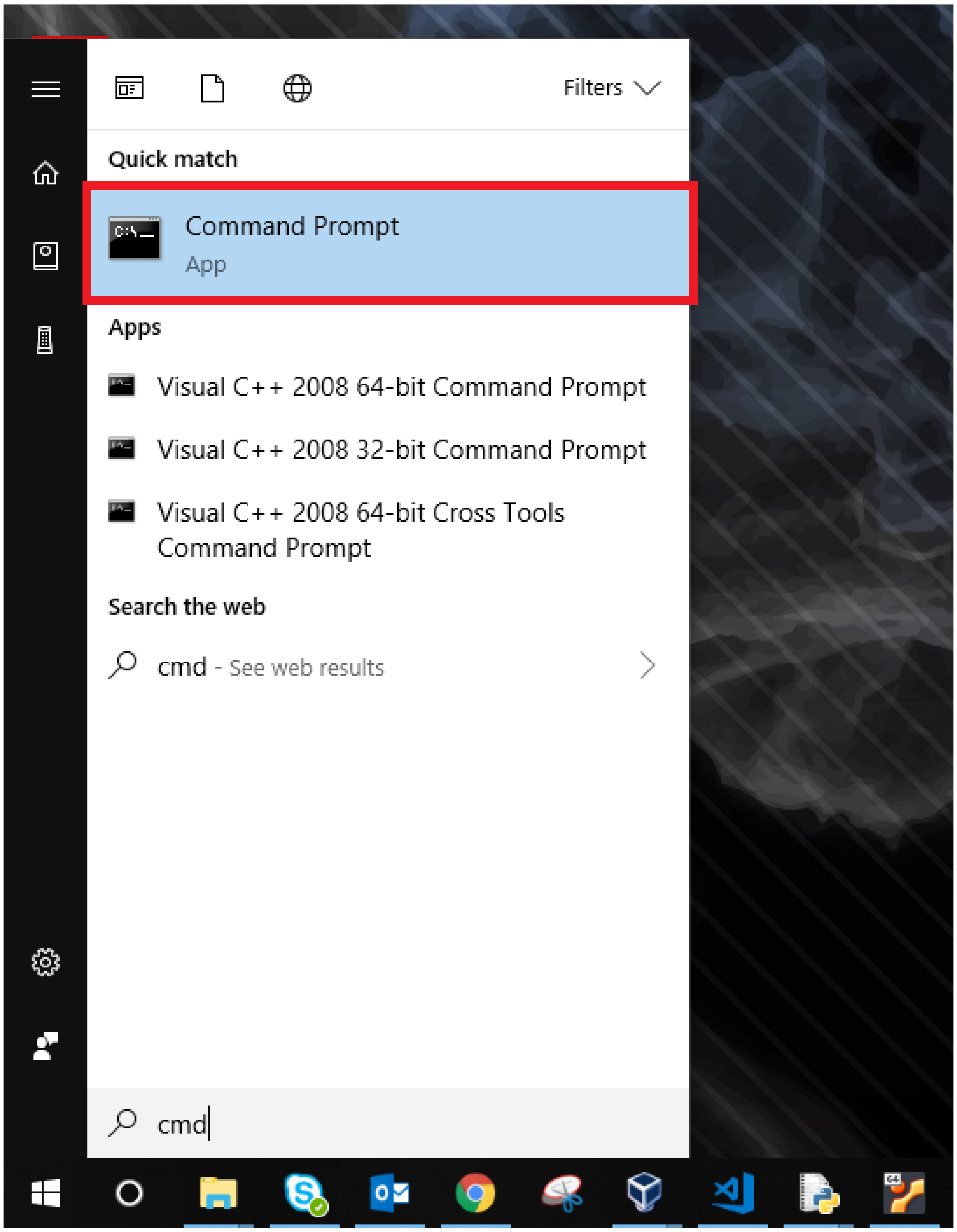
   to set the appropriate version of python for your system.

   For example, to use python 3.6 with MatLab:

```
pyversion C:\Python36\python.exe
```

4. Set up the environment.

   Open up the command prompt


Filters
Quick match
Command Prompt
App
Apps
Visual C++ 2008 64-bit Command Prompt
Visual C++ 2008 32-bit Command Prompt
Visual C++ 2008 64-bit Cross Tools Command Prompt
Search the web
cmd - See web results
cmd

Navigate to your VOLTTRON installation

```
cd \Your\directory\path\to\volttron-develop
```

Use pip to install and setup dependencies.

```
pip install -r examples\StandAloneMatLab\requirements.txt
```

```
pip install -e .
```

Note: If you get the error doing the second step because of an already installed volttron from a different directory, manually delete the *volttron-egg.* link file from your *<python path>\Lib\site-*

packages directory (for example:

```
del C:\\Python27\\lib\\site-packages\\volttron-egg.link
```

and re-run the second command

5. Configure the agent

   The configuration settings for the standalone agent are in setting.py (located in *volttron-develop\examples\StandAloneMatLab\*)

   settings.py

   - *volttron_to_matlab* needs to be set to the topic that will send your script and command line arguments to your stand alone agent. This was defined in the config.
   - *matlab_to_volttron* needs to be set to the topic that will send your script's output back to your volttron platform. This was defined in config.
   - *vip_address* needs to be set to the address of your volttron instance
   - *port* needs to be set to the port of your volttron instance
   - *server_key* needs to be set to the public server key of your primary volttron platform. This can be obtained from the primary volttron platform using `vctl auth serverkey` (VOLTTRON must be running to use this command.)

   It is possible to have multiple standalone agents running. In this case, copy the *StandAloneMatLab* folder, and make the necessary changes to the new *settings.py* file. Unless it is connecting to a separate VOLTTRON instance, you should only need to change the *volttron_to_matlab* setting.

   Note: It is recommended that you generate a new "agent_public" and "agent_private" key for your standalone agent. This can be done using the `vctl auth keypair` command on your primary VOLTTRON platform on Linux. If you plan to use multiple standalone agents, they will each need their own keypair.

6. Add standalone agent key to VOLTTRON platform

   - Copy the public key from *settings.py* in the StandAloneMatLab folder.
   - While the primary VOLTTRON platform is running on the linux machine, add the agent public key using the `vctl auth` command on the Linux machine. This will make VOLTTRON platform allow connections from the standalone agent

```
vctl auth add --credentials <standalone agent public key>
```

7. Run standalone agent

   At this point, the agent is ready to run. To use the agent, navigate to the example folder and use python to start the agent. The agent will then wait for a message to be published to the selected topic by the MatLab agent.

```
cd examples\StandAloneMatLab\

python standalone_matlab.py
```

   The output should be similar to this:

```
2019-08-01 10:42:47,592 volttron.platform.vip.agent.core DEBUG: identity:␣
↪standalone_matlab
2019-08-01 10:42:47,592 volttron.platform.vip.agent.core DEBUG: agent_
↪uuid: None
2019-08-01 10:42:47,594 volttron.platform.vip.agent.core DEBUG:␣
↪serverkey: None
2019-08-01 10:42:47,596 volttron.platform.vip.agent.core DEBUG: AGENT␣
↪RUNNING on ZMQ Core standalone_matlab
2019-08-01 10:42:47,598 volttron.platform.vip.zmq_connection DEBUG: ZMQ␣
↪connection standalone_matlab
2019-08-01 10:42:47,634 volttron.platform.vip.agent.core INFO: Connected␣
↪to platform: router: ebae9efa-5e8f-49e3-95a0-2020ddff9e8a version: 1.0␣
↪identity: standalone_matlab
2019-08-01 10:42:47,634 volttron.platform.vip.agent.core DEBUG: Running␣
↪onstart methods.
```

Note: If you have Python3 as your default Python run the command `python -2 standalone_matlab.py`

8. On the Linux machine configure the Matlab Agent to publish commands to the topic standalone agent is listening to. To load a new configuration or to change the current configuration enter

```
vctl config store <agent vip identity> config <path\to\configfile>
```

Whenever there is a change in the configuration in the config store, or whenever the agent starts, the MatLab Agent sends the configured command to the topic configured. As long as the standalone agent has been started and is listening to the appropriate topic, the output in the log should look similar to this:

```
2019-08-01 10:43:18,925 (matlab_agentV2agent-0.3 3539) matlab_agentV2.agent␣
↪DEBUG: Configuring Agent
2019-08-01 10:43:18,926 (matlab_agentV2agent-0.3 3539) matlab_agentV2.agent␣
↪DEBUG: Publishing on: matlab/to_matlab/1
2019-08-01 10:43:18,926 (matlab_agentV2agent-0.3 3539) matlab_agentV2.agent␣
↪DEBUG: Sending message: testScript2.py,20
2019-08-01 10:43:18,926 (matlab_agentV2agent-0.3 3539) matlab_agentV2.agent␣
↪DEBUG: Agent Configured!
2019-08-01 10:43:18,979 (matlab_agentV2agent-0.3 3539) matlab_agentV2.agent␣
↪INFO: Agent: matlab/to_volttron/1
Message:
'20'
```

Once the matlab agent publishes the message (in the above case, "testScript2.py,20") on the windows command prompt running the standalone agent, you should see the message that was received by the standalone agent.

```
2019-08-01 10:42:47,671 volttron.platform.vip.agent.subsystems.configstore␣
↪DEBUG: Processing callbacks for affected files: {}
The Message is: testScript2.py,20
```

Note: If MatLabAgent_v2 has been installed and started, and you have not started the *standalone_matlab agent*, you will need to either restart the matlab_agentV2, or make a change to the configuration in the config store to send command to the topic standalone agent is actively listening to.

Node Red Example

Node Red is a visual programming language wherein users connect small units of functionality "nodes" to create "flows".

There are two example nodes that allow communication between Node-Red and VOLTTRON. One node reads subscribes to messages on the VOLTTRON message bus and the other publishes to it.

Dependencies

The example nodes depend on *python-shell* to be installed and available to the Node Red environment.

Installation

Copy all files from *volttron/examples/NodeRed* to your *~/.node-red/nodes* directory. *~/.node-red* is the default directory for Node Red files. If you have set a different directory use that instead.

Set the variables at the beginning of the *volttron.js* file to be a valid VOLTTRON environment, VOLTTRON home, and Python PATH.

Valid CURVE keys need to be added to the *settings.py* file. If they are generated with the *vctl auth keypair* command then the public key should be added to VOLTTRON's authorization file with the following:

```
$ vctl auth add
```

The serverkey can be found with:

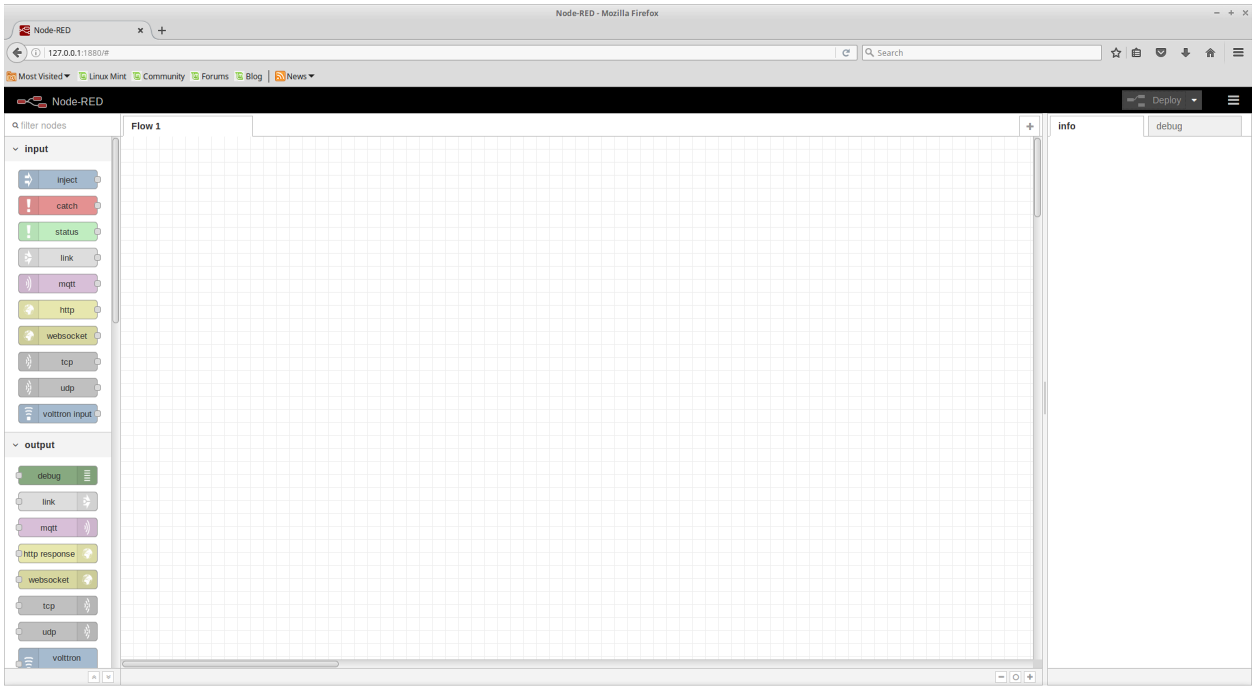
```
$ vctl auth serverkey
```

Usage

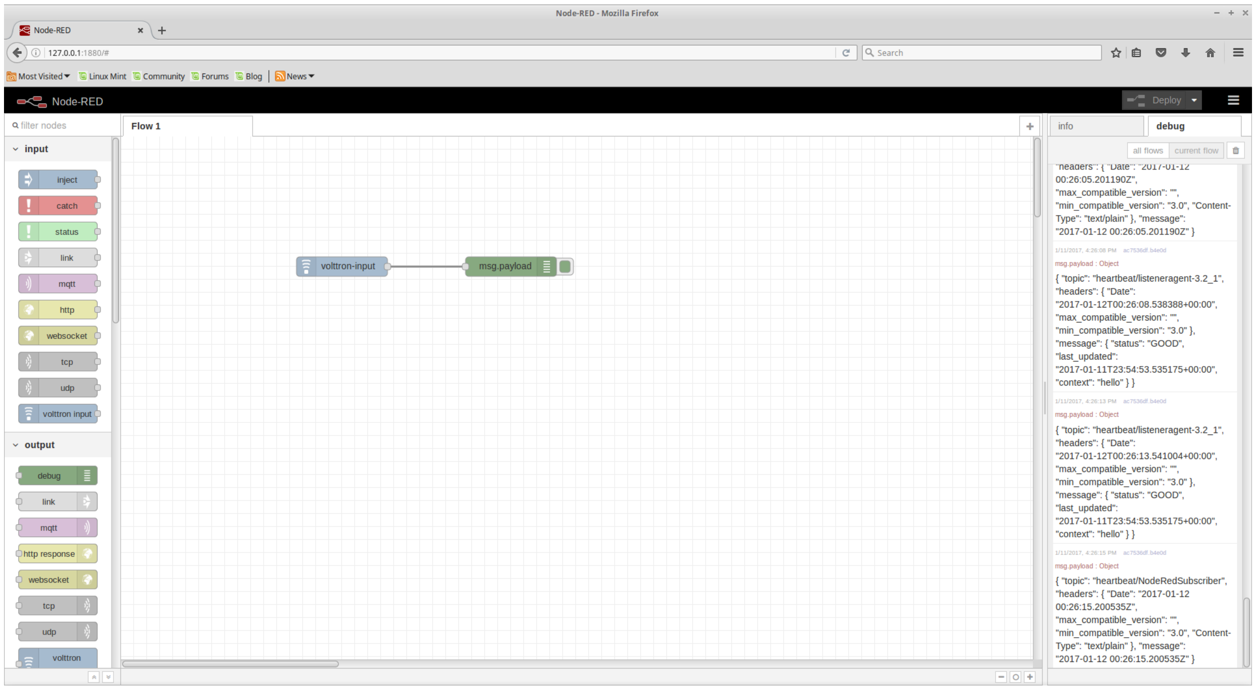
Start VOLTTRON and Node Red.

```
$ node-red


Welcome to Node-RED
===================

11 Jan 15:26:49 - [info] Node-RED version: v0.14.4
11 Jan 15:26:49 - [info] Node.js  version: v0.10.25
11 Jan 15:26:49 - [info] Linux 3.16.0-38-generic x64 LE
11 Jan 15:26:49 - [info] Loading palette nodes
11 Jan 15:26:49 - [warn] ------------------------------------------------------
11 Jan 15:26:49 - [warn] [rpi-gpio] Info : Ignoring Raspberry Pi specific node
11 Jan 15:26:49 - [warn] ------------------------------------------------------
11 Jan 15:26:49 - [info] Settings file  : /home/volttron/.node-red/settings.js
11 Jan 15:26:49 - [info] User directory : /home/volttron/.node-red
11 Jan 15:26:49 - [info] Flows file     : /home/volttron/.node-red/flows_volttron.json
11 Jan 15:26:49 - [info] Server now running at http://127.0.0.1:1880/
11 Jan 15:26:49 - [info] Starting flows
11 Jan 15:26:49 - [info] Started flows
```

The output from the Node Red command indicates the address of its web interface. Nodes available for use are in the left sidebar.



We can now use the VOLTTRON nodes to read from and write to VOLTTRON.



```
"headers": { "Date": "2017-01-12
00:26:05.201190Z",
"max_compatible_version": "",
"min_compatible_version": "3.0", "Content-
Type": "text/plain" }, "message":
"2017-01-12 00:26:05.201190Z" }

1/11/2017, 4:26:08 PM   ac7536df.b4e0d
msg.payload : Object
{ "topic": "heartbeat/listeneragent-3.2_1",
"headers": { "Date":
"2017-01-12T00:26:08.538388+00:00",
"max_compatible_version": "",
"min_compatible_version": "3.0" },
"message": { "status": "GOOD",
"last_updated":
"2017-01-11T23:54:53.535175+00:00",
"context": "hello" } }

1/11/2017, 4:26:13 PM   ac7536df.b4e0d
msg.payload : Object
{ "topic": "heartbeat/listeneragent-3.2_1",
"headers": { "Date":
"2017-01-12T00:26:13.541004+00:00",
"max_compatible_version": "",
"min_compatible_version": "3.0" },
"message": { "status": "GOOD",
"last_updated":
"2017-01-11T23:54:53.535175+00:00",
"context": "hello" } }

1/11/2017, 4:26:15 PM   ac7536df.b4e0d
msg.payload : Object
{ "topic": "heartbeat/NodeRedSubscriber",
"headers": { "Date": "2017-01-12
00:26:15.200535Z",
"max_compatible_version": "",
"min_compatible_version": "3.0", "Content-
Type": "text/plain" }, "message":
"2017-01-12 00:26:15.200535Z" }
```

Scheduler Example Agent

The Scheduler Example Agent demonstrates how to use the scheduling feature of the :ref‘Actuator Agent <Actuator-Agent>‘ as well as how to send a command. This agent publishes a request for a reservation on a (fake) device then takes an action when it's scheduled time appears. The ActuatorAgent must be running to exercise this example.

Note: Since there is no actual device, an error is produced when the agent attempts to take its action.

```
def publish_schedule(self):
    '''Periodically publish a schedule request'''
    headers = {
        'AgentID': agent_id,
        'type': 'NEW_SCHEDULE',
        'requesterID': agent_id, #The name of the requesting agent.
        'taskID': agent_id + "-ExampleTask", #The desired task ID for this task. It␣
↪must be unique among all other scheduled tasks.
        'priority': 'LOW', #The desired task priority, must be 'HIGH', 'LOW', or 'LOW_
↪PREEMPT'
        }

    start = str(datetime.datetime.now())
    end = str(datetime.datetime.now() + datetime.timedelta(minutes=1))


    msg = [
       ['campus/building/unit',start,end]
    ]
    self.vip.pubsub.publish(
    'pubsub', topics.ACTUATOR_SCHEDULE_REQUEST, headers, msg)
```

The agent listens to schedule announcements from the actuator and then issues a command:

```
@PubSub.subscribe('pubsub', topics.ACTUATOR_SCHEDULE_ANNOUNCE(campus='campus',
                                     building='building',unit='unit'))
def actuate(self, peer, sender, bus,  topic, headers, message):
    print ("response:",topic,headers,message)
    if headers[headers_mod.REQUESTER_ID] != agent_id:
        return
    '''Match the announce for our fake device with our ID
    Then take an action. Note, this command will fail since there is no
    actual device'''
    headers = {
                'requesterID': agent_id,
               }
    self.vip.pubsub.publish(
    'pubsub', topics.ACTUATOR_SET(campus='campus',
                                     building='building',unit='unit',
                                     point='point'),
                                     headers, 0.0)
```

Simple Web Agent Walk-through

A simple web enabled agent that will hook up with a VOLTTRON message bus and allow interaction between it via HTTP. This example agent shows a simple file serving agent, a JSON-RPC based call, and a websocket based connection mechanism.

Starting VOLTTRON Platform

Note: Activate the environment first *active the environment*

In order to start the simple web agent, we need to bind the VOLTTRON instance to the a web server. We need to specify the address and the port for the web server. For example, if we want to bind the *localhost:8080* as the web server we start the VOLTTRON platform as follows:

```
./start-volttron --bind-web-address http://127.0.0.1:8080
```

Once the platform is started, we are ready to run the Simple Web Agent.

Running Simple Web Agent

Note: The following assumes the shell is located at the *VOLTTRON_ROOT*.

Copy the following into your shell (save it to a file for executing it again later):

```
python scripts/install-agent.py \
    --agent-source examples/SimpleWebAgent \
    --tag simpleWebAgent \
    --vip-identity webagent \
    --force \
    --start
```

This will create a web server on `http://localhost:8080`. The *index.html* file under *simpleweb/webroot/simpleweb/* can be any HTML page which binds to the VOLTTRON message bus .This provides a simple example of providing a web endpoint in VOLTTRON.

Path based registration examples

- Files will need to be in *webroot/simpleweb* in order for them to be browsed from `http://localhost:8080/simpleweb/index.html`
- Filename is required as we don't currently auto-redirect to any default pages as shown in `self.vip.web.register_path("/simpleweb", os.path.join(WEBROOT))`

The following two examples show the way to call either a JSON-RPC (default) endpoint and one that returns a different content-type. With the JSON-RPC example from volttron central we only allow post requests, however this is not required.

- Endpoint will be available at *http://localhost:8080/simple/text* `self.vip.web.register_endpoint("/simple/text", self.text)`
- Endpoint will be available at *http://localhost:8080/simple/jsonrpc* `self.vip.web.register_endpoint("/simpleweb/jsonrpc", self.rpcendpoint)`
- `text/html` content type specified so the browser can act appropriately like `[("Content-Type", "text/html")]`
- The default response is `application/json so our` endpoint returns appropriately with a JSON based response.

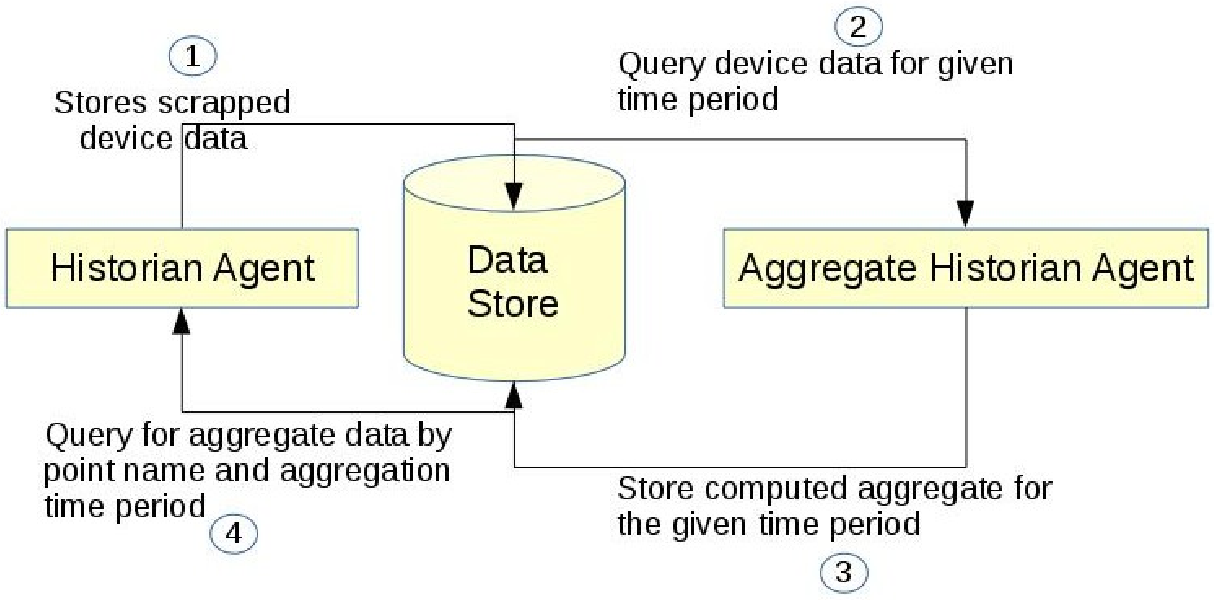
Agent Specifications

Documents included below are intended to provide a specification to classes of agents which include a base class in the VOLTTRON repository and have a well defined set of functions and services.

Aggregate Historian

Description

An aggregate historian computes aggregates of data stored in a given volttron historian's data store. It runs periodically to compute aggregate data and store it in new tables/collections in the historian's data store. Each regular historian (BaseHistorian) needs a corresponding aggregate historian to compute and store aggregates of the data collected by the regular historian.


1
Stores scrapped device data
2
Query device data for given time period
Historian Agent
Data Store
Aggregate Historian Agent
Query for aggregate data by point name and aggregation time period
4
Store computed aggregate for the given time period
3

Software Interfaces

Data Collection - Data store that the aggregate historian uses as input source needs to be up. Access to it should be provided using an account that has create, read, and write privileges. For example, a MongoAggregateHistorian needs to be able to connect to the mongodb used by MongoHistorian using an account that has read and write access to the db used by the MongoHistorian.

Data retrieval Aggregate Historian Agent does not provide api for retrieving the aggregate data collected. Use Historian agent's query interface. Historian's query api will be modified as below

1. topic_name can now be a list of topic names or a single topic
2. Two near optional parameters have been added to the query api - agg_type (aggregation type), agg_period (aggregation time period). Both these parameters are mandatory for query aggregate data.
3. New api to get the list of aggregate topics available for querying

User Interfaces

Aggregation agent requires user to configure the following details as part of the agent configuration file

1. Connection details for historian's data store (same as historian agent configuration)
2. **List of aggregation groups where each group contains:**
    1. Aggregation period - integer followed by m/h/d/w/M (minutes, hours, days, weeks or months)
    2. Boolean parameter to indicate if aggregation periods should align to calendar times
    3. Optional collection start time in utc. If not provided, aggregation collection will start from current time
    4. List of aggregation points with topic name, type of aggregation (sum, avg, etc.), and minimum number of records that should be available for the aggregate to be computed
    5. Topic name can be specified either as a list of specific topic names (topic_names=[topic1, topic2]) or a regular expression pattern (topic_name_pattern="Building1/device_*/Zone*temperature")
    6. When aggregation is done for a single topic then name of topic will be used for the computed aggregation as well. You could optionally provide a unique aggregation_topic_name
    7. When topic_name_pattern or multiple topics are specified a unique aggregate topic name should be specified for the collected aggregate. Users can query for the collected aggregate data using this aggregate topic name.
    8. User should be able to configure multiple aggregations done with the same time period/time interval and these should be time synchronized.

Functional Capabilities

1. Should run periodically to compute aggregate data.
2. Same instance of the agent should be able to collect data at more than one time interval
3. For each configured time period/interval agent should be able to collect different type of aggregation for different topics/points
4. Support aggregation over multiple topics/points
5. Agent should be able to handle and normalize different time units such as minutes, hours, days, weeks and months
6. Agent should be able to compute aggregate both based on wall clock based time intervals and calendar based time interval. For example, agent should be able to calculate daily average based on 12.00AM to 11.59PM of a calendar day or between current time and the same time the previous day.
7. Data should be stored in such a way that users can easily retrieve multiple aggregate topics data within a given time interval

Data Structure

Collected aggregate data should be stored in the historian data store into new collection or tables and should be accessible by historian agent's query interface. Users should easily be able to query aggregate data of multiple points for which data is time synchronized.

Use Cases

Collect monthly average of multiple topic using data from MongoDBHistorian

1. Create a configuration file with connection details from Mongo Historian configuration file and add additional aggregation specific configuration

```
{
    # configuration from mongo historian - START
    "connection": {
        "type": "mongodb",
        "params": {
            "host": "localhost",
            "port": 27017,
            "database": "mongo_test",
            "user": "test",
            "passwd": "test"
        }
    },
    # configuration from mongo historian - START
    "aggregations":[
        # list of aggregation groups each with unique aggregation_period and
        # list of points that needs to be collected
        {
        "aggregation_period": "1M",
        "use_calendar_time_periods": true,
        "utc_collection_start_time":"2016-03-01T01:15:01.000000",
        "points": [
            {
             "topic_names": ["Building/device/point1", "Building/device/point2"],
             "aggregation_topic_name":"building/device/point1_2/month_sum",
             "aggregation_type": "avg",
             "min_count": 2
            }
        ]
        }
    ]
}
```

In the above example configuration, here is what each field under "aggregations" represent

- **aggregation_period**: can be minutes(m), hours(h), weeks(w), or months(M)
- **use_calendar_time_periods: true or false - Should aggregation period align to calendar time periods. Default False. Exam**
  - if "aggregation_period":"1h" and "use_calendar_time_periods": false, example periods: 10.15-11.15, 11.15-12.15, 12.15-13.15 etc.
  - if "aggregation_period":"1h" and "use_calendar_time_periods": true, example periods: 10.00-11.00, 11.00-12.00, 12.00-13.00 etc.
  - if "aggregation_period":"1M" and "use_calendar_time_periods": true, aggregation would be computed from the first day of the month to last day of the month
  - if "aggregation_period":"1M" and "use_calendar_time_periods": false, aggregation would be computed with a 30 day interval based on aggregation collection start time
- **utc_collection_start_time**: The time from which aggregation computation should start. If not provided this would default to current time.

- **points: List of points, its aggregation type and min_count** **topic_names**: List of topic_names across which aggregation should be computed. **aggregation_topic_name**: Unique name given for this aggregate. Optional if aggregation is for a single topic. **aggregation_type**: Type of aggregation to be done. Please see *Constraints and Limitations*

  min_count: Optional. Minimum number of records that should exist within the configured time period for a aggregation to be computed.

2. install and starts the aggregate historian using the above configuration

3. Query aggregate data: Query using historian's query api by passing two additional parameters - agg_type and agg_period

```
result1 = query_agent.vip.rpc.call('platform.historian',
                                   'query',
                                   topic='building/device/point1_2/month_sum',
                                   agg_type='avg',
                                   agg_period='1M',
                                   count=20,
                                   order="FIRST_TO_LAST").get(10)
```

Collect weekly average(sunday to saturday) of single topic using data from MongoDBHistorian

1. **Create a configuration file with connection details from Mongo Historian configuration file and add additional aggregatio**

   - **aggregation_period**: "1w",
   - **topic_names**: ["Building/device/point1"], #topic for which you want to compute aggregation
   - **aggregation_topic_name** need not be provided

2. install and starts the aggregate historian using the above configuration

3. Query aggregate data: Query using historian's query api by passing two additional parameters - agg_type and agg_period. topic_name will be the same as the point name for which aggregation is collected

```
result1 = query_agent.vip.rpc.call('platform.historian',
                                   'query',
                                   topic='Building/device/point1',
                                   agg_type='avg',
                                   agg_period='1w',
                                   count=20,
                                   order="FIRST_TO_LAST").get(10)
```

Collect hourly average for multiple topics based on topic_name pattern

1. **Create a configuration file with connection details from Mongo Historian configuration file and add additional aggregatio**

   - **aggregation_period**: "1h",
   - Insetead of topic_names provide **topic_name_pattern**. For example, **"topic_name_pattern":"Building1/device_a*/point1"**
   - **aggregation_topic_name** provide a unique aggregation topic name

2. install and starts the aggregate historian using the above configuration

3. Query aggregate data: Query using historian's query api by passing two additional parameters - agg_type and agg_period. topic_name will be the same as the point name for which aggregation is collected

```
result1 = query_agent.vip.rpc.call('platform.historian',
                                   'query',
                                   topic="unique aggregation_topic_name provided in␣
↪configuration",
                                   agg_type='avg',
                                   agg_period='1h',
                                   count=20,
                                   order="FIRST_TO_LAST").get(10)
```

Collect 7 day average of two topics and time synchronize them for easy comparison

1. Create a configuration file with connection details from Mongo Historian configuration file and add additional aggregation specific configuration. The configuration file should be similar to the below example

```
{
    # configuration from mongo historian - START
    "connection": {
        "type": "mongodb",
        "params": {
            "host": "localhost",
            "port": 27017,
            "database": "mongo_test",
            "user": "test",
            "passwd": "test"
        }
    },
    # configuration from mongo historian - START
    "aggregations":[
        # list of aggregation groups each with unique aggregation_period and
        # list of points that needs to be collected
        {
        "aggregation_period": "1w",
        "use_calendar_time_periods": false, #compute for last 7 days, then the next␣
↪and so on..
        "points": [
            {
             "topic_names": ["Building/device/point1"],
             "aggregation_type": "avg",
             "min_count": 2
            },
            {
             "topic_names": ["Building/device/point2"],
             "aggregation_type": "avg",
             "min_count": 2
            }
        ]
        }
    ]
}
```

2. install and starts the aggregate historian using the above configuration

3. Query aggregate data: Query using historian's query api by passing two additional parameters - agg_type and agg_period. provide the list of topic names for which aggregate was configured above. Since both the points were

configured within a single "aggregations" array element, their aggregations will be time synchronized

```
result1 = query_agent.vip.rpc.call('platform.historian',
                                   'query',
                                   topic=['Building/device/point1''Building/device/
↪point2'],
                                   agg_type='avg',
                                   agg_period='1w',
                                   count=20,
                                   order="FIRST_TO_LAST").get(10)
```

Results will be of the format

```
{'values': [
   ['Building/device/point1', '2016-09-06T23:31:27.679910+00:00', 2],
   ['Building/device/point1', '2016-09-15T23:31:27.679910+00:00', 3],
   ['Building/device/point2', '2016-09-06T23:31:27.679910+00:00', 2],
   ['Building/device/point2', '2016-09-15T23:31:27.679910+00:00', 3]],
'metadata': {}}
```

Qurey list of aggregate data collected

```
result = query_agent.vip.rpc.call('platform.historian',
                                  'get_aggregate_topics').get(10)
```

The result will be of the format:

```
[(aggregate topic name, aggregation type, aggregation time period, configured list of␣
↪topics or topic name pattern), ...]
```

This shows the list of aggregation currently being computed periodically

Qurey list of supported aggregation types

```
result = query_agent.vip.rpc.call(
    AGG_AGENT_VIP,
    'get_supported_aggregations').get(timeout=10)
```

Constraints and Limitations

1. Initial implementation of this agent will not support any data filtering for raw data before computing data aggregation
2. Initial implementation should support all aggregation types directly supported by underlying data store. End user input is needed to figure out what additional aggregation methods are to be supported

    MySQL

Name	Description
AVG()	Return the average value of the argument
BIT_AND()	Return bitwise AND
BIT_OR()	Return bitwise OR
BIT_XOR()	Return bitwise XOR
COUNT()	Return a count of the number of rows returned
GROUP_CONCAT()	Return a concatenated string
MAX()	Return the maximum value
MIN()	Return the minimum value
STD()	Return the population standard deviation
STDDEV()	Return the population standard deviation
STDDEV_POP()	Return the population standard deviation
STDDEV_SAMP()	Return the sample standard deviation
SUM()	Return the sum
VAR_POP()	Return the population standard variance
VAR_SAMP()	Return the sample variance
VARIANCE()	Return the population standard variance

SQLite

Name	Description
AVG()	Return the average value of the argument
COUNT()	Return a count of the number of rows returned
GROUP_CONCAT()	Return a concatenated string
MAX()	Return the maximum value
MIN()	Return the minimum value
SUM()	Return sum of all non-NULL values in the group. If there are no non-NULL input rows then returns NULL .
TOTAL()	Return sum of all non-NULL values in the group.If there are no non-NULL input rows returns 0.0

MongoDB

Name	Description
SUM	Returns a sum of numerical values. Ignores non-numeric values
AVG	Returns a average of numerical values. Ignores non-numeric values
MAX	Returns the highest expression value for each group.
MIN	Returns the lowest expression value for each group.
FIRST	Returns a value from the first document for each group. Order is only defined if the documents are in a defined order.
LAST	Returns a value from the last document for each group. Order is only defined if the documents are in a defined order.
PUSH	Returns an array of expression values for each group
AD-DTOSET	Returns an array of unique expression values for each group. Order of the array elements is undefined.
STDDE-VPOP	Returns the population standard deviation of the input values
STDDE-VSAMP	Returns the sample standard deviation of the input values

Tagging Service

Description

Tagging service provides VOLTTRON users the ability to add semantic tags to different topics so that topic can be queried by tags instead of specific topic name or topic name pattern.

Taxonomy

VOLLTTRON will use tags from Project Haystack. Tags defined in haystack will be imported into VOLTTRON and grouped by categories to tag topics and topic name prefix.

Dependency

Once data in VOLTTRON has been tagged, users will be able to query topics based on tags and use the resultant topics to query the historian

Features

1. User should be able to tag individual components of a topic such as campus, building, device, point etc.
2. Using the tagging service users should only be able to add tags already defined in the volttron tagging schema. New tags should be explicitly added to the tagging schema before it can be used to tag topics or topic prefix
3. Users should be able batch process and tag multiple topic names or topic prefix using a template. At the end of this, users should be notified about the list of topics that did not confirm to the template. This will help users to individually add or edit tags for those specific topics
4. When users query for topics based on a tag, the results would correspond to the current metadata values. It is up to the calling agent/application to periodically query for latest updates if needed.
5. Users should be able query based on tags on a specific topic or its topic prefix/parents
6. Allow for count and skip parameters in queries to restrict count and allow pagination

API

1. Get the list of tag categories available

rpc call to tagging service method **'get_categories'** with optional parameters:

1. **include_description** - set to True to return available description for each category. Default = False
2. **skip** - number of categories to skip. this parameter along with count can be used for paginating results
3. **count** - limit the total number of tag categories returned to given count
4. **order** - ASCENDING or DESCENDING. By default, it will be sorted in ascending order

2. Get the list of tags for a specific category

rpc call to tagging service method **'get_tags_by_category'** with parameter:

1. **category** - <category name>

and optional parameters:

2. **include_kind - indicate if result should include the** kind/data type for tags returned. Defaults to False
3. **include_description - indicate if result should include** available description for tags returned. Defaults to False
4. **skip** - number of tags to skip. this parameter along with count can be used for paginating results
5. **count** - limit the total number of tags returned to given count
6. **order** - ASCENDING or DESCENDING. By default, it will be sorted in ascending order

3. Get the list of tags for a topic_name or topic_name_prefix

rpc call to tagging service method **get_tags_by_topic**

with parameter

1. **topic_prefix** - topic name or topic name prefix

and optional parameters:

2. **include_kind - indicate if result should include the** kind/data type for tags returned. Defaults to False
3. **include_description - indicate if result should include** available description for tags returned. Defaults to False
4. **skip** - number of tags to skip. this parameter along with count can be used for paginating results
5. **count** - limit the total number of tags returned to given count
6. **order** - ASCENDING or DESCENDING. By default, it will be sorted in ascending order

4. Find topic names by tags

rpc call to tagging service method **'get_topics_by_tags'** with the one or more of the following parameters

1. **and_condition** - dictionary of tag and its corresponding values that should be matched using equality operator or a list of tags that should exists/be true. Tag conditions are combined with AND condition. Only topics that match all the tags in the list would be returned
2. **or_condition** - dictionary of tag and its corresponding values that should be matched using equality operator or a list tags that should exist/be true. Tag conditions are combined with OR condition. Topics that match any of the tags in the list would be returned. If both **and_condition** and **or_condition** are provided then they are combined using AND operator.
3. **condition** - conditional statement to be used for matching tags. If this parameter is provided the above two parameters are ignored. The value for this parameter should be an expression that contains one or more query conditions combined together with an "AND" or "OR". Query conditions can be grouped together using parenthesis. Each condition in the expression should conform to one of the following format:
    1. <tag name/ parent.tag_name> <binary_operator> <value>

2. <tag name/ parent.tag_name>
3. <tag name/ parent.tag_name> LIKE <regular expression within single quotes
4. the word NOT can be prefixed before any of the above three to negate the condition.
5. expressions can be grouped with parenthesis.

For example

```
condition="tag1 = 1 and not (tag2 < '' and tag2 > '') and tag3␣
↪and NOT tag4 LIKE '^a.*b$'"
condition="NOT (tag5='US' OR tag5='UK') AND NOT tag3 AND NOT␣
↪(tag4 LIKE 'a.*')"
condition="campusRef.geoPostalCode='20500' and equip and boiler"
```

6. **skip** - number of topics to skip. this parameter along with count can be used for paginating results
7. **count** - limit the total number of tag topics returned to given count
8. **order** - ASCENDING or DESCENDING. By default, it will be sorted in ascending order

5. Query data based on tags

Use above api to get topics by tags and then use the result to query historian's query api.

6. Add tags to specific topic name or topic name prefix

rpc call to to tagging service method **'add_topic_tags'** with parameters:

1. **topic_prefix** - topic name or topic name prefix
2. **tags** - {<valid tag>:value, <valid_tag>: value,. . . }
3. **update_version** - True/False. Default to False. If set to True and if any of the tags update an existing tag value the older value would be preserved as part of tag version history. **NOTE:** This is a placeholder. Current version does not support versioning.

7. Add tags to multiple topics

rpc call to to tagging service method **'add_tags'** with parameters:

1. **tags** - dictionary object containing the topic and the tag details. format:

```
<topic_name or prefix or topic_name pattern>: {<valid tag>:<value>, ... }, ... }
```

2. **update_version** - True/False. Default to False. If set to True and if any of the tags update an existing tag value the older value would be preserved as part of tag version history

Use case examples

1. Loading news tags for an existing VOLTTRON instance

Current topic names:

/campus1/building1/deviceA1/point1
/campus1/building1/deviceA1/point2
/campus1/building1/deviceA1/point3
/campus1/building1/deviceA2/point1
/campus1/building1/deviceA2/point2
/campus1/building1/deviceA2/point3
/campus1/building1/deviceB1/point1
/campus1/building1/deviceB1/point2
/campus1/building1/deviceB2/point1
/campus1/building1/deviceB1/point2

Step 1:

Create a python dictionary object contains topic name pattern and its corresponding tag/value pair. Use topic pattern names to fill out tags that can be applied to more than one topic or topic prefix. Use specific topic name and topic prefix for tags that apply only to a single entity. For example:

```
{
# tags specific to building1
'/campus1/building1':
    {
    'site': true,
    'dis': ": 'some building description',
    'yearBuilt': 2015,
    'area': '24000sqft'
    },
# tags that apply to all device of a specific type
'/campus1/building1/deviceA*':
    {
    'dis': "building1 chilled water system - CHW",
    'equip': true,
    'campusRef':'campus1',
    'siteRef': 'campus1/building1',
    'chilled': true,
    'water' : true,
    'secondaryLoop': true
    }
# tags that apply to point1 of all device of a specific type
'/campus1/building1/deviceA*/point1':
    {
    'dis': "building1 chilled water system - point1",
    'point': true,
    'kind': 'Bool',
    'campusRef':'campus1',
    'siteRef': 'campus1/building1'
    }
# tags that apply to point2 of all device of a specific type
'/campus1/building1/deviceA*/point2':
    {
    'dis': "building1 chilled water system - point2",
    'point': true,
    'kind': 'Number',
    'campusRef':'campus1',
    'siteRef': 'campus1/building1'
```

(continues on next page)

(continued from previous page)

```
    }
# tags that apply to point3 of all device of a specific type
'/campus1/building1/deviceA*/point3':
    {
    'dis': "building1 chilled water system - point3",
    'point': true,
    'kind': 'Number',
    'campusRef':'campus1',
    'siteRef': 'campus1/building1'
    }
# tags that apply to all device of a specific type
'/campus1/building1/deviceB*':
    {
    'dis': "building1 device of type B",
    'equip': true,
    'chilled': true,
    'water' : true,
    'secondaryLoop': true,
    'campusRef':'campus1',
    'siteRef': 'campus1/building1'
    }
# tags that apply to point1 of all device of a specific type
'/campus1/building1/deviceB*/point1':
    {
    'dis': "building1 device B - point1",
    'point': true,
    'kind': 'Bool',
    'campusRef':'campus1',
    'siteRef': 'campus1/building1',
    'command':true
    }
# tags that apply to point1 of all device of a specific type
'/campus1/building1/deviceB*/point2':
    {
    'dis': "building1 device B - point2",
    'point': true,
    'kind': 'Number',
    'campusRef':'campus1',
    'siteRef': 'campus1/building1'
    }
}
```

Step 2: Create tags using template above

Make an RPC call to the add_tags method and pass the python dictionary object

Step 3: Create tags specific to a point or device

Any tags that were not included in step one and needs to be added later can be added using the rpc call to tagging service either the method **'add_topic_tags' 'add_tags'**

For example:

```
agent.vip.rpc.call(
        'platform.tagging',
        'add_topic_tags',
        topic_prefix='/campus1/building1/deviceA1',
        tags={'tag1':'value'})
```

```
agent.vip.rpc.call(
        'platform.tagging',
        'add_topic_tags',
        tags={
            '/campus1/building1/deviceA2':
                {'tag1':'value'},
            '/campus1/building1/deviceA2/point1':
                {'equipRef':'campus1/building1/deviceA2'}
             }
        )
```

2. Querying based on a topic's tag and it parent's tags

Query - Find all points that has the tag 'command' and belong to a device/unit that has a tag 'chilled'

```
agent.vip.rpc.call(
        'platform.tagging',
        'get_topics_by_tags',
        condition='temperature and equip.chilled)
```

In the above code block 'command' and 'chilled' are the tag names that would be searched, but since the tag 'chilled' is prefixed with 'equip.' the tag in a parent topic

The above query would match the topic '/campus1/building1/deviceB1/point1' if tags in the system are as follows

'/campus1/building1/deviceB1/point1' tags:

```
{
'dis': "building1 device B - point1",
'point': true,
'kind': 'Bool',
'campusRef':'campus1',
'siteRef': 'campus1/building1',
'equipRef': 'campus1/building1/deviceB1',
'command':true
}
```

'/campus1/building1/deviceB1' tags

```
{
'dis': "building1 device of type B",
'equip': true,
'chilled': true,
'water' : true,
'secondaryLoop': true,
'campusRef':'campus1',
'siteRef': 'campus1/building1'
}
```

Possible future improvements

1. Versioning - When a value of a tag is changed, users should be prompted to verify if this change denotes a new version or a value correction. If this value denotes a new version, then older value of the tag should preserved in a history/audit store
2. Validation of tag values based on data type
3. Support for units validation and conversions
4. Processing and saving geologic coordinates that can enable users to do geo-spatial queries in databases that support it.

Weather Service

Description

The weather service agent provides API to access current weather data, historical data and weather forecast data. There are several weather data providers, some paid and some free. Weather data providers differs from one and other

1. In the kind of features provided - current data, historical data, forecast data
2. The data points returned
3. The naming schema used to represent the data returned
4. Units of data returned
5. Frequency of data updates

The weather service agent has a design similar to historians. There is a single base weather service that defines the api signatures and the ontology of the weather data points. There is one concrete weather service agents for each weather provider. Users can install one or more provider specific agent to access weather data.

The initial implementation is for NOAA and would support current and forecast data requests. NOAA does not support accessing historical weather data through their api. This agent implements request data caching.

The second implementation is for darksky.net.

Features

Base weather agent features:

1. Caching

   The weather service provides basic caching capability so that repeated request for same data can be returned from cache instead of network round trip to the weather data provider. This is also useful to limit the number of request made to the provider as most weather data provider have restrictions on number of requests for developer/free api keys. The size of the cache can be restricted by setting an optional configuration parameter 'max_size_gb'

2. Name mapping

   Data points returned by concrete weather agents is mapped to standard names based on CF standard names table Name mapping is done using a CSV file. See *Configuration* section for an example configuration

3. Unit conversion

   If data returned from the provider is of the format {"data_point_name":value}, base weather agent can do unit conversions on the value. Both name mapping and unit conversions can be specified as a csv file and packaged with the concrete implementing agent. This feature is not mandatory. See *Configuration* section for an example configuration

Core weather data retrieval features :

1. Retrieve current weather data.
2. Retrieve hourly weather forecast data.
3. Retrieve historical weather data.
4. Periodic polling of current weather data for one or more locations. Users can configure one or more locations in a config file and weather agent will periodically poll for current weather data for the configured locations and publish the results to message bus.

The set of points returned from the above queries depends on the specific weather data provider, however the point names returned are from the standard schema.

Note:

1. Since individual weather data provider can support slightly different sets of features, users are able to query for the list of available features. For example a provider could provide daily weather forecast in addition to the hourly forecast data.

API

1. Get available features

rpc call to weather service method **'get_api_features'**

Parameters - None

Returns - dictionary of api features that can be called for this weather agent.

2. Get current weather data

rpc call to weather service method **'get_current_weather'**

Parameters:

1. **locations** - dictionary containing location details. The format of location accepted differs between different weather providers and even different APIs supported by the same provider For example the location input could be either {"zipcode":value} or {"region":value, "country": value}.

Returns: List of dictionary objects containing current weather data. The actual data points returned depends on the weather service provider.

3. Get hourly forecast data

rpc call to weather service method **'get_hourly_forecast'**

Parameters:

1. **locations** - dictionary containing location details. The format of location accepted differs between different weather providers and even different APIs supported by the same provider For example the location input could be either {"zipcode":value} or {"region":value, "country": value}.

optional parameters:

2. **hours** - The number of hours for which forecast data are returned. By default, it is 24 hours.

Returns: List of dictionary objects containing forecast data. If weather data provider returns less than requested number of hours result returned would contain a warning message in addition to the result returned by the provider

4. Get historical weather data

rpc call to weather service method **'get_hourly_historical'**

Parameters:

1. **locations** - dictionary containing location details. For example the location input could be either {"zip-code":value} or {"region":value, "country": value}.
2. **start_date** - start date of requested data
3. **end_date** - end date of requested data

Returns: List of dictionary objects containing historical data.

Note: Based on the weather data provider this api could do multiple calls to the data provider to get the requested data. For example, darksky.net allows history data query by a single date and not a date range.

5. Periodic polling of current weather data

This can be achieved by configuring the locations for which data is requested in the agent's configuration file along with polling interval. Results for each location configured, is published to its corresponding result topic. is no result topic prefix is configured, then results for all locations are posted to the topic weather/poll/current/all. poll_topic_suffixes when provided should be a list of string with the same length as the number of poll_locations. When topic prefix is specified, each location's result is published to weather/poll/current/<poll_topic_suffix for that location> topic_prefix.

Configuration

Example configuration:

```
{
    poll_locations: [
        {"zip": "22212"},
        {"zip": "99353"}
    ],
    poll_topic_suffixes: ["result_22212", "result_99353"],
    poll_interval: 20 #seconds,

    #optional cache arguments
    max_cache_size: ...

}
```

Example configuration for mapping point names returned by weather provider to a standard name and units:

```
Service_Point_Name,Standard_Point_Name,Service_Units,Standard_Units
temperature,air_temperature,fahrenheit,celsius
```

Caching

Weather agent will cache data until the configured size limit is reached (if provided).

1. Current and forecast data:

   If current/forecast weather data exists in cache and if the request time is within the update time period of the api (specified by a concrete implementation) then by default cached data would be returned otherwise a new request is made for it. If hours is provided and the amount of cached data records is less than hours, this will also result in a new request.

2. Historical data cache:

   Weather api will query the cache for available data for the given time period and fill and missing time period with data from the remote provider.

3. Clearing of cache:

   Users can configure the maximum size limit for cache. For each api call, before data is inserted in cache, weather agent will check for this size limit and purge records in this order. - Current data older than update time period - Forecast data older than update time period - History data starting with the oldest cached data

Assumptions

1. User has api key for accessing weather api for a specific weather data provider, if a key is required.
2. Different weather agent might have different requirement for how input locations are specified. For example NOAA expects a station id for querying current weather and requires either a lat/long or gridpoints to query for forecast. weatherbit.io accepts zip code.
3. Not all features might be implemented by a specific weather agent. For example NOAA doesn't make history data available using their weather api.
4. Concrete agents could expose additional api features
5. Optionally, data returned will be based on standard names provided by the CF standard names table (see Ontology). Any points with a name not mapped to a standard name would be returned as is.

1.9 Driver Development

In order for VOLTTRON agents to gather data from a device or to set device values, agents send requests to the Master Driver Agent to read or set points. The Master Driver Agent then sends these requests on to the appropriate driver for interfacing with that device based on the topic specified in the request and the configuration of the Master Driver. Drivers provide an interface between the device and the master driver by implementing portions of the devices' protocols needed to serve the functions of setting and reading points.

As a demonstration of developing a driver a driver can be made to read and set points in a CSV file. This driver will only differ from a real device driver in terms of the specifics of the protocol.

1.9.1 Create a Driver and Register class

When a new driver configuration is added to the Master Driver, the Master Driver will look for a file or directory in its interfaces directory (services/core/MasterDriverAgent/master_driver/interfaces) that shares the name of the value specified by "driver_type" in the configuration file. For the CSV Driver, create a file named csvdriver.py in that directory.

```
├── master_driver
│       ├── agent.py
│       ├── driver.py
│       ├── __init__.py
│       ├── interfaces
│       │       ├── __init__.py
│       │       ├── bacnet.py
|       |       ├── csvdriver.py
│       │       └── modbus.py
│       └── socket_lock.py
├── master-driver.agent
└── setup.py
```

Following is an example using the directory type structure:

```
├── master_driver
│       ├── agent.py
│       ├── driver.py
│       ├── __init__.py
│       ├── interfaces
│       │       ├── __init__.py
│       │       ├── bacnet.py
|       |       ├── csvdriver.py
│       │       ├── modbus.py
│       │       ├── modbus_tk.py
│       │       |   ├── __init__.py
│       │       |   ├── tests
│       │       |   ├── requirements.txt
│       │       |   └── README.rst
```

Note: Using this format, the directory must be the name specified by "driver_type" in the configuration file and the *Interface* class must be in the *__init__.py* file in that directory.

This format is ideal for including additional code files as well as requirements files, tests and documentation.

Interface Basics

A complete interface consists of two parts: the interface class and one or more register classes.

Interface Class Skeleton

When the Master Driver processes a driver configuration file it creates an instance of the interface class found in the interface file (such as the one we've just created). The interface class is responsible for managing the communication between the Volttron Platform, and the device. Each device has many registers which hold the values Volttron agents are interested in so generally the interface manages reading and writing to and from a device's registers. At a minimum, the interface class should be configurable, be able to read and write registers, as well as read all registers with a single request. First create the csv interface class boilerplate.

```python
class Interface(BasicRevert, BaseInterface):
    def __init__(self, **kwargs):
        super(Interface, self).__init__(**kwargs)

```

(continues on next page)

(continued from previous page)

```
    def configure(self, config_dict, registry_config_str):
        pass

    def get_point(self, point_name):
        pass

    def _set_point(self, point_name, value):
        pass

    def _scrape_all(self):
        pass
```

This class should inherit from the BaseInterface and at a minimum implement the configure, get_point, set_point, and scrape_all methods.

Note: In some sense, drivers are sub-agents running under the same process as the Master Driver. They should be instantiated following the agent pattern, so a function to handle configuration and create the Driver object has been included.

Register Class Skeleton

The interface needs some information specifying the communication for each register on the device. For each different type of register a register class should be defined which will help identify individual registers and determine how to communicate with them. Our CSV driver will be fairly basic, with one kind of "register", which will be a column in a CSV file. Other drivers may require many kinds of registers; for instance, the Modbus protocol driver has registers which store data in byte sized chunks and registers which store individual bits, therefore the Modbus driver has bit and byte registers.

For the CSV driver, create the register class boilerplate:

```
class CsvRegister(BaseRegister):
    def __init__(self, csv_path, read_only, pointName, units, reg_type,
                 default_value=None, description=''):
        super(CsvRegister, self).__init__("byte", read_only, pointName, units,␣
↪description=description)
```

This class should inherit from the BaseRegister. The class should keep register metadata, and depending upon the requirements of the protocol/device, may perform the communication.

The BACNet and Modbus drivers may be used as examples of more specific implementations. For the purpose of this demonstration writing and reading points will be done in the register, however, this may not always be the case (as in the case of the BACNet driver).

Filling out the Interface class

The CSV interface will be writing to and reading from a CSV file, so the device configuration should include a path specifying a CSV file to use as the "device". The CSV "device: path value is set at the beginning of the agent loop which runs the configure method when the Master Driver starts. Since this Driver is for demonstration, we'll create the CSV with some default values if the configured path doesn't exist. The CSV device will consist of 2 columns: "Point Name" specifying the name of the register, and "Point Value", the current value of the register.

```
_log = logging.getLogger(__name__)

CSV_FIELDNAMES = ["Point Name", "Point Value"]
CSV_DEFAULT = [
    {
        "Point Name": "test1",
        "Point Value": 0
    },
    {
        "Point Name": "test2",
        "Point Value": 1
    },
    {
        "Point Name": "test3",
        "Point Value": "testpoint"
    }
]
type_mapping = {"string": str,
                "int": int,
                "integer": int,
                "float": float,
                "bool": bool,
                "boolean": bool}

class Interface(BasicRevert, BaseInterface):
def __init__(self, **kwargs):
    super(Interface, self).__init__(**kwargs)
    self.csv_path = None

def configure(self, config_dict, registry_config_str):
    self.csv_path = config_dict.get("csv_path", "csv_device.csv")
    if not os.path.isfile(self.csv_path):
        _log.info("Creating csv 'device'")
        with open(self.csv_path, "w+") as csv_device:
            writer = DictWriter(csv_device, fieldnames=CSV_FIELDNAMES)
            writer.writeheader()
            writer.writerows(CSV_DEFAULT)
    self.parse_config(registry_config_str)
```

At the end of the configuration method, the Driver parses the registry configuration. The registry configuration is a csv which is used to tell the Driver which register the user wishes to communicate with and includes a few meta-data values about each register, such as whether the register can be written to, if the register value uses a specific measurement unit, etc. After each register entry is parsed from the registry config a register is added to the driver's list of active registers.

```
def parse_config(self, config_dict):
    if config_dict is None:
        return

    for index, regDef in enumerate(config_dict):
        # Skip lines that have no point name yet
        if not regDef.get('Point Name'):
            continue

        read_only = regDef.get('Writable', "").lower() != 'true'
        point_name = regDef.get('Volttron Point Name')
        if not point_name:
```

(continues on next page)

(continued from previous page)

```
            point_name = regDef.get("Point Name")
        if not point_name:
            raise ValueError("Registry config entry {} did not have a point name or␣
↪volttron point name".format(
                index))
        description = regDef.get('Notes', '')
        units = regDef.get('Units', None)
        default_value = regDef.get("Default Value", "").strip()
        if not default_value:
            default_value = None
        type_name = regDef.get("Type", 'string')
        reg_type = type_mapping.get(type_name, str)

        register = CsvRegister(
            self.csv_path,
            read_only,
            point_name,
            units,
            reg_type,
            default_value=default_value,
            description=description)

        if default_value is not None:
            self.set_default(point_name, register.value)

        self.insert_register(register)
```

Since the driver's registers will be doing the work of parsing the registers the interface only needs to select the correct register to read from or write to and instruct the register to perform the corresponding unit of work.

```
def get_point(self, point_name):
    register = self.get_register_by_name(point_name)
    return register.get_state()

def _set_point(self, point_name, value):
    register = self.get_register_by_name(point_name)
    if register.read_only:
        raise IOError("Trying to write to a point configured read only: " + point_
↪name)
    register.set_state(value)
    return register.get_state()

def _scrape_all(self):
    result = {}
    read_registers = self.get_registers_by_type("byte", True)
    write_registers = self.get_registers_by_type("byte", False)
    for register in read_registers + write_registers:
        result[register.point_name] = register.get_state()
    return result
```

Writing the Register class

The CSV driver's register class is responsible for parsing the CSV, reading the corresponding rows to return the register's current value and writing updated values into the CSV for the register. On a device which communicates via a protocol such as Modbus the same units of work would be done, but using pymodbus to perform the reads and writes. Here, Python's CSV library will be used as our "protocol implementation".

The Register class determines which file to read based on values passed from the Interface class.

```
class CsvRegister(BaseRegister):
    def __init__(self, csv_path, read_only, pointName, units, reg_type,
                 default_value=None, description=''):
        super(CsvRegister, self).__init__("byte", read_only, pointName, units,
                                          description=description)
        self.csv_path = csv_path
```

To find its value the register will read the CSV file, iterate over each row until a row with the point name the same as the register name at which point it extracts the point value, and returns it. The register should be written to handle problems which may occur, such as no correspondingly named row being present in the CSV file.

```
def get_state(self):
    if os.path.isfile(self.csv_path):
        with open(self.csv_path, "r") as csv_device:
            reader = DictReader(csv_device)
            for point in reader:
                if point.get("Point Name") == self.point_name:
                    point_value = point.get("Point Value")
                    if not point_value:
                        raise RuntimeError("Point {} not set on CSV Device".
↪format(self.point_name))
                    else:
                        return point_value
        raise RuntimeError("Point {} not found on CSV Device".format(self.point_name))
    else:
        raise RuntimeError("CSV device at {} does not exist".format(self.csv_path))
```

Likewise to overwrite an existing value, the register will iterate over each row until the point name matches the register name, saving the output as it goes. When it finds the correct row it instead saves the output updated with the new value then continues on. Finally it writes the output back to the csv.

```
def set_state(self, value):
    _log.info("Setting state for {} on CSV Device".format(self.point_name))
    field_names = []
    points = []
    found = False
    with open(self.csv_path, "r") as csv_device:
        reader = DictReader(csv_device)
        field_names = reader.fieldnames
        for point in reader:
            if point["Point Name"] == self.point_name:
                found = True
                point_copy = point
                point_copy["Point Value"] = value
                points.append(point_copy)
            else:
                points.append(point)

    if not found:
        raise RuntimeError("Point {} not found on CSV Device".format(self.point_name))
    else:
        with open(self.csv_path, "w") as csv_device:
            writer = DictWriter(csv_device, fieldnames=field_names)
            writer.writeheader()
            writer.writerows([dict(row) for row in points])
    return self.get_state()
```

At this point we should be able to scrape the CSV device using the Master Driver and set points using the actuator.

Creating Driver Configurations

The configuration files for the CSV driver are very simple, but in general, the device configuration should specify the parameters which the interface requires to communicate with the device and the registry configuration contains rows which correspond to registers and specifies their usage.

Here's the driver configuration for the CSV driver:

```
{
    "driver_config": {"csv_path": "csv_driver.csv"},
    "driver_type": "csvdriver",
    "registry_config":"config://csv_registers.csv",
    "interval": 30,
    "timezone": "UTC"
}
```

Note: The "driver_type" value must match the name of the driver's python file as this is what the Master Driver will look for when searching for the correct interface.

And here's the registry configuration:

Volttron Point Name	Point Name	Writable
test1	test1	true
test2	test2	true
test3	test3	true

The BACNet and Modbus driver docs and example configurations can be used to compare these configurations to more complex configurations.

1.9.2 Testing your driver

To test the driver's scrape all functionality, one can install a ListenerAgent and Master Driver with the driver's configurations, and run them. To do so for the CSV driver using the configurations above: activate the Volttron environment start the platform, tail the platform's log file, then try the following:

```
python scripts/install-agent.py -s examples/ListenerAgent
python scripts/install-agent.py -s services/core/MasterDriverAgent -c services/core/
↪MasterDriverAgent/master-driver.agent
vctl config store platform.driver devices/<campus>/<building>/csv_driver <path to␣
↪driver configuration>
vctl config store platform.driver <registry config path from driver configuration>
↪<path to registry configuration>
```

Note: *vctl config list platform.driver* will list device and registry configurations stored for the master driver and *vctl config delete platform.driver <config in configs list>* can be used to remove a configuration entry - these commands are very useful for debugging

---

After the Master Driver starts the driver's output should appear in the logs at regular intervals based on the Master Driver's configuration.

Here is some sample CSV driver output:

```
2019-11-15 10:32:00,010 (listeneragent-3.3 22996) listener.agent INFO: Peer: pubsub,␣
↪Sender: platform.driver:, Bus:
, Topic: devices/pnnl/isb1/csv_driver/all, Headers: {'Date': '2019-11-15T18:32:00.
↪001360+00:00', 'TimeStamp':
'2019-11-15T18:32:00.001360+00:00', 'SynchronizedTimeStamp': '2019-11-15T18:32:00.
↪000000+00:00',
'min_compatible_version': '3.0', 'max_compatible_version': ''}, Message:
[{'test1': '0', 'test2': '1', 'test3': 'testpoint'},
 {'test1': {'type': 'integer', 'tz': 'UTC', 'units': None},
  'test2': {'type': 'integer', 'tz': 'UTC', 'units': None},
  'test3': {'type': 'integer', 'tz': 'UTC', 'units': None}}]
```

This output is an indication of the basic scrape all functionality working in the Interface class - in our implementation this is also an indication of the basic functionality of the Interface class "get_point" method and Register class "get_state" methods working (although edge cases should still be tested!).

To test the Interface's "set_point" method and Register's "set_state" method we'll need to use the Actuator agent. The following agent code can be used to alternate a point's value on a schedule using the actuator, as well as perform an action based on a pubsub subscription to a single point:

```python
def CsvDriverAgent(config_path, **kwargs):
    """Parses the Agent configuration and returns an instance of
    the agent created using that configuration.

    :param config_path: Path to a configuration file.

    :type config_path: str
    :returns: Csvdriveragent
    :rtype: Csvdriveragent
    """
    _log.debug("Config path: {}".format(config_path))
    try:
        config = utils.load_config(config_path)
    except Exception:
        config = {}

    if not config:
        _log.info("Using Agent defaults for starting configuration.")
    _log.debug("config_dict before init: {}".format(config))
    utils.update_kwargs_with_config(kwargs, config)
    return Csvdriveragent(**kwargs)


class Csvdriveragent(Agent):
    """
    Document agent constructor here.
    """

    def __init__(self, csv_topic="", **kwargs):
        super(Csvdriveragent, self).__init__(**kwargs)
        _log.debug("vip_identity: " + self.core.identity)

        self.agent_id = "csv_actuation_agent"
        self.csv_topic = csv_topic

        self.value = 0
```

(continues on next page)

(continued from previous page)

```
        self.default_config = {
            "csv_topic": self.csv_topic
        }

        # Set a default configuration to ensure that self.configure is called␣
↪immediately to setup
        # the agent.
        self.vip.config.set_default("config", self.default_config)

        # Hook self.configure up to changes to the configuration file "config".
        self.vip.config.subscribe(self.configure, actions=["NEW", "UPDATE"], pattern=
↪"config")

    def configure(self, config_name, action, contents):
        """
        Called after the Agent has connected to the message bus. If a configuration␣
↪exists at startup
        this will be called before onstart.

        Is called every time the configuration in the store changes.
        """
        config = self.default_config.copy()
        config.update(contents)

        _log.debug("Configuring Agent")
        _log.debug(config)

        self.csv_topic = config.get("csv_topic", "")

        # Unsubscribe from everything.
        self.vip.pubsub.unsubscribe("pubsub", None, None)

        self.vip.pubsub.subscribe(peer='pubsub',
                                  prefix="devices/" + self.csv_topic + "/all",
                                  callback=self._handle_publish)

    def _handle_publish(self, peer, sender, bus, topic, headers, message):
        _log.info("Device {} Publish: {}".format(self.csv_topic, message))

    @Core.receiver("onstart")
    def onstart(self, sender, **kwargs):
        """
        This is method is called once the Agent has successfully connected to the␣
↪platform.
        This is a good place to setup subscriptions if they are not dynamic or
        do any other startup activities that require a connection to the message bus.
        Called after any configurations methods that are called at startup.

        Usually not needed if using the configuration store.
        """
        self.core.periodic(30, self.actuate_point)

    def actuate_point(self):
        _now = get_aware_utc_now()
        str_now = format_timestamp(_now)
        _end = _now + td(seconds=10)
        str_end = format_timestamp(_end)
```

(continues on next page)

(continued from previous page)

```
        schedule_request = [[self.csv_topic, str_now, str_end]]
        result = self.vip.rpc.call(
            'platform.actuator', 'request_new_schedule', self.agent_id, 'my_test',
↪'HIGH', schedule_request).get(
            timeout=4)
        point_topic = self.csv_topic + "/" + "test1"
        result = self.vip.rpc.call(
            'platform.actuator', 'set_point', self.agent_id, point_topic, self.value).
↪get(
            timeout=4)
        self.value = 0 if self.value is 1 else 1

    @Core.receiver("onstop")
    def onstop(self, sender, **kwargs):
        """
        This method is called when the Agent is about to shutdown, but before it␣
↪disconnects from
        the message bus.
        """
        pass


def main():
    """Main method called to start the agent."""
    utils.vip_main(CsvDriverAgent,
                   version=__version__)


if __name__ == '__main__':
    # Entry point for script
    try:
        sys.exit(main())
    except KeyboardInterrupt:
        pass
```

While this code runs, since the Actuator is instructing the Interface to set points on the device, the pubsub all publish can be used to check that the values are changing as expected.

1.10 Contributing Code

As an open source project VOLTTRON requires input from the community to keep development focused on new and useful features. To that end we are revising our commit process to hopefully allow more contributors to be apart of the community. The following document outlines the process for source code and documentation to be submitted. There are GUI tools that may make this process easier, however this document will focus on what is required from the command line.

The only requirements for contributing are Git (Linux version control software) and your favorite web browser.

Note: The following guide assumes the user has already created a fork of the core VOLTTRON repository. Please review the *docs* if you have not yet created a fork.

1.10.1 Reviewing Changes

Okay, we've written a cool new *foo.py* script to service *bar* in our deployment. Let's make sure our code is up-to-snuff.

Code

First, go through the code.

Note: We on the VOLTTRON team would recommend an internal code review - it can be really hard to catch small mistakes, typos, etc. for code you just finished writing.

- Does the code follow best-practices for Python, object-oriented programming, unit and integration testing, etc.?
- Does the code contain any typos and does it follow Pep8 guidelines?
- Does the code follow the guidelines laid out in the VOLTTRON documentation?

Docs

Next, Check out the documentation.

- Is it complete?
  - Has an introduction describing purpose
  - Describes configuration including all parameters
  - Includes installation instructions
  - Describes behavior at runtime
  - Describes all available endpoints (JSON-RPC, pub/sub messages, Web-API endpoints, etc.)
- Does it follow the *VOLTTRON documentation guidelines*?

Tests

You've included tests, right? Unit and integration tests show users that *foo.py* is better than their wildest dreams - all of the features work, and include components they hadn't even considered themselves!

- Are the unit tests thorough?
  - Success and failure cases
  - Tests for each independent component of the code
- Do the integration tests capture behavior with a running VOLTTRON platform?
  - Success and Failure cases
  - Tests for each endpoint
  - Tests for interacting with other agents if necessary
  - Are status, health, etc. updating as expected when things go wrong or the code recovers?
- Can the tests be read to describe the behavior of the code?

Structure

For agents and drivers, the VOLTTRON team has some really simple structure recommendations. These make your project structure nice and tidy, and integrate nicely with the core repository.

For agents:

```
TestAgent/
├── setup.py
├── config
├── README.rst
├── tester
|   ├── agent.py
|   └── __init__.py
└── tests
    └── test_agent.py
```

For drivers, the interface should be a file named after the driver in the Master Driver's interfaces directory:

```
├── master_driver
│         ├── agent.py
│         ├── driver.py
│         ├── __init__.py
│         ├── interfaces
│         │         ├── __init__.py
│         │         ├── bacnet.py
|         |         ├── csvdriver.py
│         │         └── new_driver.py
```

Or in the *__init__.py* file in a directory named after the driver in the Master Driver's interfaces directory:

```
├── master_driver
│         ├── agent.py
│         ├── driver.py
│         ├── __init__.py
│         ├── interfaces
│         │         ├── __init__.py
│         │         ├── bacnet.py
│         │         ├── new_driver
│         │         |   └── __init__.py
```

This option is ideal for adding additional code files, and including documentation and tests.

1.10.2 Creating a Pull Request to the main VOLTTRON repository

After reviewing changes to our fork of the VOLTTRON repository, we want our changes to be added into the main VOLTTRON repository. After all, our *foo.py* can cure a lot of the world's problems and of course it is always good to have a copyright with the correct year. Open your browser to https://github.com/VOLTTRON/volttron/compare/develop...YOUR_USERNAME:develop.

On that page the base fork should always be VOLTTRON/volttron with the base develop, the head fork should be <YOUR USERNAME>/volttron and the compare should be the branch in your repository to pull from. Once you have verified that you have got the right changes made then, click on create pull request, enter a title and description that represent your changes and submit the pull request.

The VOLTTRON repository has a description template to use to format your PR:

```
# Description

Please include a summary of the change and which issue is fixed. Please also include␣
↪relevant motivation and context. List any dependencies that are required for this␣
↪change.

Fixes # (issue)

## Type of change

Please delete options that are not relevant.

- [ ] Bug fix (non-breaking change which fixes an issue)
- [ ] New feature (non-breaking change which adds functionality)
- [ ] Breaking change (fix or feature that would cause existing functionality to not␣
↪work as expected)
- [ ] This change requires a documentation update

# How Has This Been Tested?

Please describe the tests that you ran to verify your changes. Provide instructions␣
↪so we can reproduce. Please also list any relevant details for your test␣
↪configuration

- [ ] Test A
- [ ] Test B

**Test Configuration**:
* Firmware version:
* Hardware:
* Toolchain:
* SDK:

# Checklist:

- [ ] My code follows the style guidelines of this project
- [ ] I have performed a self-review of my own code
- [ ] I have commented my code, particularly in hard-to-understand areas
- [ ] I have made corresponding changes to the documentation
- [ ] My changes generate no new warnings
- [ ] I have added tests that prove my fix is effective or that my feature works
- [ ] New and existing unit tests pass locally with my changes
- [ ] Any dependent changes have been merged and published in downstream modules
```

Note: The VOLTTRON repository includes a stub for completing your pull request. Please follow the stub to facilitate the reviewing and merging processes.

1.10.3 What happens next?

Once you create a pull request, one or more VOLTTRON team members will review your changes and either accept them as is ask for modifications in order to have your commits accepted. Typical response time is approximately two weeks; please be patient, your pull request will be reviewed. You will be automatically emailed through the GitHub notification system when this occurs (assuming you haven't changed your GitHub preferences).

Merging changes from the main VOLTTRON repository

As time goes on the VOLTTRON code base will continually be modified so the next time you want to work on a change to your files the odds are your local and remote repository will be out of date. In order to get your remote VOLTTRON repository up to date with the main VOLTTRON repository you could simply do a pull request to your remote repository from the main repository. To do so, navigate your browser to https://github.com/YOUR_USERNAME/volttron/compare/develop...VOLTTRON:develop.

Click the 'Create Pull Request' button. On the following page click the 'Create Pull Request' button. On the next page click 'Merge Pull Request' button.

Once your remote is updated you can now pull from your remote repository into your local repository through the following command:

```
git pull
```

The other way to get the changes into your remote repository is to first update your local repository with the changes from the main VOLTTRON repository and then pushing those changes up to your remote repository. To do that you need to first create a second remote entry to go along with the origin. A remote is simply a pointer to the url of a different repository than the current one. Type the following command to create a new remote called 'upstream':

```
git remote add upstream https://github.com/VOLTTRON/volttron
```

To update your local repository from the main VOLTTRON repository then execute the following command where upstream is the remote and develop is the branch to pull from:

```
git pull upstream develop
```

Finally to get the changes into your remote repository you can execute:

```
git push origin
```

Other commands to know

At this point in time you should have enough information to be able to update both your local and remote repository and create pull requests in order to get your changes into the main VOLTTRON repository. The following commands are other commands to give you more information that the preceding tutorial went through

Viewing what the remotes are in our local repository

```
git remote -v
```

Stashing changed files so that you can do a merge/pull from a remote

```
git stash save 'A comment to be listed'
```

Applying the last stashed files to the current repository

```
git stash pop
```

Finding help about any git command

```
git help
git help branch
git help stash
git help push
git help merge
```

Creating a branch from the branch and checking it out

```
git checkout -b newbranchname
```

Checking out a branch (if not local already will look to the remote to checkout)

```
git checkout branchname
```

Removing a local branch (cannot be current branch)

```
git branch -D branchname
```

Determine the current and show all local branches

```
git branch
```

Using Travis Continuous Integration Tools

The main VOLTTRON repository is hooked into an automated build tool called travis-ci. Your remote repository can be automatically built with the same tool by hooking your account into travis-ci's environment. To do this go to https://travis-ci.org and create an account. You can using your GitHub login directly to this service. Then you will need to enable the syncing of your repository through the travis-ci service. Finally you need to push a new change to the repository. If the build fails you will receive an email notifying you of that fact and allowing you to modify the source code and then push new changes out.

1.11 Contributing Documentation

The Community is encouraged to contribute documentation back to the project as they work through use cases the developers may not have considered or documented. By contributing documentation back, the community can learn from each other and build up a more extensive knowledge base.

VOLTTRON™ documentation utilizes ReadTheDocs: http://volttron.readthedocs.io/en/develop/ and is built using the Sphinx Python library with static content in Restructured Text.

1.11.1 Building the Documentation

Static documentation can be found in the *docs/source* directory. Edit or create new .rst files to add new content using the Restructured Text format. To see the results of your changes the documentation can be built locally through the command line using the following instructions:

If you've already *bootstrapped* VOLTTRON™, do the following while activated. If not, this will also pull down the necessary VOLTTRON™ libraries.

```
python bootstrap.py --documentation
cd docs
make html
```

Then, open your browser to the created local files:

```
file:///home/<USER>/git/volttron/docs/build/html/overview/index.html
```

When complete, changes can be contributed back using the same process as code *contributions* by creating a pull request. When the changes are accepted and merged, they will be reflected in the ReadTheDocs site.

1.11.2 Documentation Styleguide

Naming Conventions

- File names and directories should be all lower-case and use only dashes/minus signs (-) as word separators

```
index.rst
├── first-document.rst
├── more-documents
│    ├──second-document.rst
```

- Reference Labels should be Capitalized and dash/minus separated:

```
.. _Reference-Label:
```

- Headings and Sub-headings should be written like book titles:

```
==============
The Page Title
==============
```

Headings

Each page should have a main title:

```
==================================
This is the Main Title of the Page
==================================
```

It can be useful to include reference labels throughout the document to use to refer back to that section of documentation. Include reference labels above titles and important headings:

```
.. _Main-Title:

==================================
This is the main title of the page
==================================
```

Heading Levels

- Page titles and documentation parts should use over-line and underline hashes:

```
=====
Title
=====
```

- Chapter headings should be over-lined and underlined with asterisks

```
*******
Chapter
*******
```

- For sections, subsections, sub-subsections, etc. underline the heading with the following:
  - =, for sections
  - -, for subsections
  - ^, for sub-subsections
  - “, for paragraphs

In addition to following guidelines for styling, please separate headers from previous content by two newlines.

```
=====
Title
=====

    Content


Subheading
==========
```

Example Code Blocks

Use bash for commands or user actions:

```
ls -al
```

Use this for the results of a command:

```
total 5277200
drwxr-xr-x 22 volttron volttron       4096 Oct 20 09:44 .
drwxr-xr-x 23 volttron volttron       4096 Oct 19 18:39 ..
-rwxr-xr-x  1 volttron volttron        164 Sep 29 17:08 agent-setup.sh
drwxr-xr-x  3 volttron volttron       4096 Sep 29 17:13 applications
```

Use this when Python source code is displayed

```
@RPC.export
def status_agents(self):
    return self._aip.status_agents()
```

Directives

Danger: Something very bad!

Tip: This is something good to know

Some other directives

"attention", "caution", "danger", "error", "hint", "important", "note", "tip", "warning", "admonition"

Links

Linking to external sites is simple:

```
Link to `Google <www.google.com>`_
```

References

You can reference other sections of documentation using the *ref* directive:

```
This will reference the :ref:`platform installation <Platform-Installation>`
```

Other resources

- http://pygments.org/docs/lexers/
- http://documentation-style-guide-sphinx.readthedocs.io/en/latest/style-guide.html
- http://www.sphinx-doc.org/en/stable/markup/code.html

1.12 Jupyter Notebooks

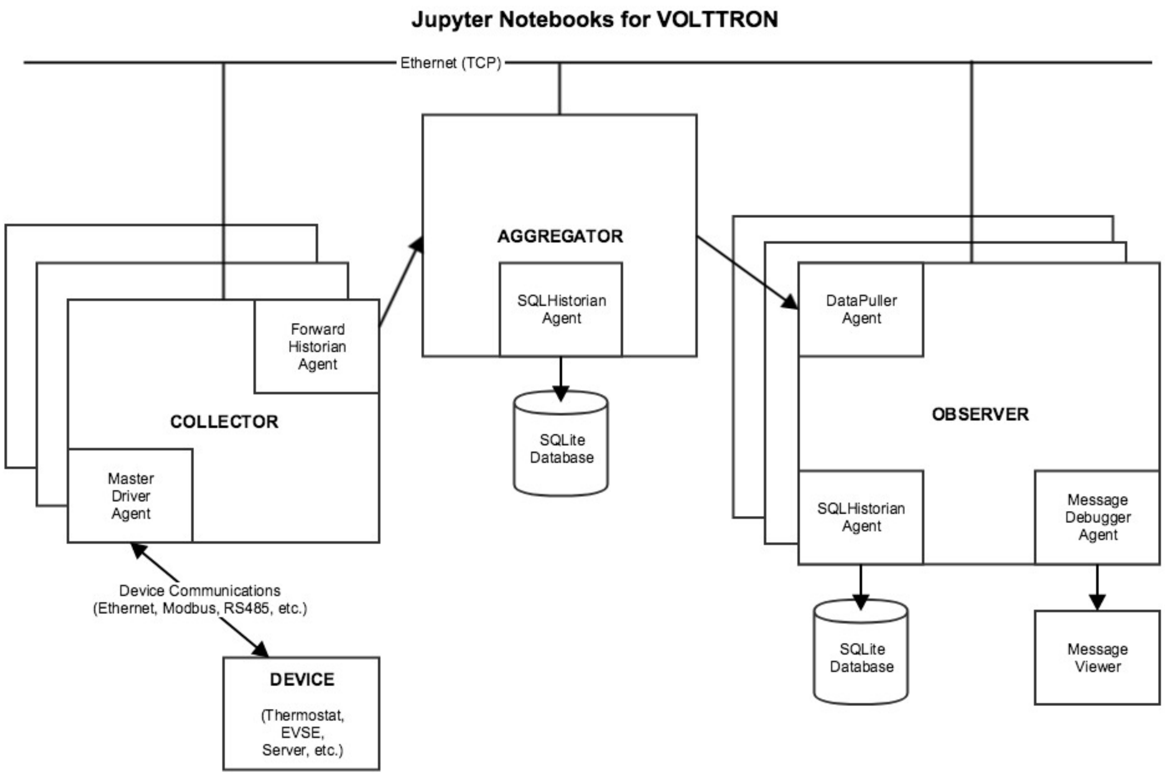
Jupyter is an open-source web application that lets you create and share "notebook" documents. A notebook displays formatted text along with live code that can be executed from the browser, displaying the execution output and preserving it in the document. Notebooks that execute Python code used to be called *iPython Notebooks*. The iPython Notebook project has now merged into Project Jupyter.

1.12.1 Using Jupyter to Manage a Set of VOLTTRON Servers

The following Jupyter notebooks for VOLTTRON have been provided as examples:

- **Collector notebooks**. Each Collector notebook sets up a particular type of device driver and forwards device data to another VOLTTRON instance, the Aggregator.
  - **SimulationCollector notebook**. This notebook sets up a group of Simulation device drivers and forwards device data to another VOLTTRON instance, the Aggregator.
  - **BacnetCollector notebook**. This notebook sets up a Bacnet (or Bacnet gateway) device driver and forwards device data to another VOLTTRON instance, the Aggregator.
  - **ChargePointCollector notebook**. This notebook sets up a ChargePoint device driver and forwards device data to another VOLTTRON instance, the Aggregator.
  - **SEP2Collector notebook**. This notebook sets up a SEP2.0 (IEEE 2030.5) device driver and forwards device data to another VOLTTRON instance, the Aggregator. The Smart Energy Profile 2.0 ("SEP2") protocol implements IEEE 2030.5, and is capable of connecting a wide array of smart energy devices to the Smart Grid. The standard is designed to run over TCP/IP and is physical layer agnostic.
- **Aggregator notebook**. This notebook sets up and executes aggregation of forwarded data from other VOLTTRON instances, using a historian to record the data.
- **Observer notebook**. This notebook sets up and executes a DataPuller that captures data from another VOLTTRON instance, using a Historian to record the data. It also uses the Message Debugger agent to monitor messages flowing across the VOLTTRON bus.

Each notebook configures and runs a set of VOLTTRON Agents. When used as a set they implement a multiple-VOLTTRON-instance architecture that captures remote device data, aggregates it, and reports on it, routing the data as follows:



1.12.2 Install VOLTTRON and Jupyter on a Server

The remainder of this guide describes how to set up a host for VOLTTRON and Jupyter. Use this setup process on a server in order to prepare it to run Jupyter notebook for VOLTTRON.

Set Up the Server and Install VOLTTRON

The following is a complete, but terse, description of the steps for installing and running VOLTTRON on a server. For more detailed, general instructions, see *Installing Volttron*.

The VOLTTRON server should run on the same host as the Jupyter server.

- Load third-party software:

```
$ sudo apt-get update
$ sudo apt-get install build-essential python-dev openssl libssl-dev libevent-dev git
$ sudo apt-get install sqlite3
```

- Clone the VOLTTRON repository from github:

```
$ cd ~
$ mkdir repos
$ cd repos
$ git clone https://github.com/VOLTTRON/volttron/
```

- Check out the develop (or master) branch and bootstrap the development environment:

```
$ cd volttron
$ git checkout develop
$ python bootstrap.py
```

- Activate and initialize the VOLTTRON virtual environment:

  Run the following each time you open a new command-line shell on the server:

  ```
  $ export VOLTTRON_ROOT=~/repos/volttron
  $ export VOLTTRON_HOME=~/.volttron
  $ cd $VOLTTRON_ROOT
  $ source env/bin/activate
  ```

Install Extra Libraries

- Add Python libraries to the VOLTTRON virtual environment:

These notebooks use third-party software that's not included in VOLTTRON's standard distribution that was loaded by *bootstrap.py*. The following additional packages are required:

- Jupyter
- SQLAlchemy (for the Message Debugger)
- Suds (for the ChargePoint driver, if applicable)
- Numpy and MatPlotLib (for plotted output)

Note: A Jupyter installation also installs and/or upgrades many dependent libraries. Doing so could disrupt other work on the OS, so it's safest to load Jupyter (and any other library code) in a virtual environment. VOLTTRON runs in a virtual environment during normal operation, so if you're using Jupyter in conjunction with VOLTTRON, it should be installed in your VOLTTRON virtual environment (In other words, be sure to use *cd $VOLTTRON_ROOT* and *source env/bin/activate* to activate the virtual environment before running `pip install`.)

- Install the third-party software:

```
$ pip install SQLAlchemy==1.1.4
$ pip install suds-jurko==0.6
$ pip install numpy
$ pip install matplotlib
$ pip install jupyter
```

Note: If *pip install* fails due to an untrusted cert, try using this command instead:

```
$ pip install --trusted-host pypi.python.org <libraryname>
```

An InsecurePlatformWarning may be displayed, but it typically won't stop the installation from proceeding.

1.12.3 Configure VOLTTRON

Use the *vcfg* wizard to configure the VOLTTRON instance. By default, the wizard configures a VOLTTRON instance that communicates with agents only on the local host (ip 127.0.0.1). This set of notebooks manages communications among multiple VOLTTRON instances on different hosts. To enable this cross-host communication on VOLTTRON's web server, replace 127.0.0.1 with the host's IP address, as follows:

```
$ vcfg
```

Accept all defaults, except as follows:

- If a prompt defaults to 127.0.0.1 as an IP address, substitute the *host's IP address* (this may happen multiple times).
- When asked whether this is a volttron central, answer *Y*.
- When prompted for a username and password, use *admin* and *admin*.

1.12.4 Start VOLTTRON

Start the main VOLTTRON process, logging to $VOLTTRON_ROOT/volttron.log:

```
$ volttron -vv -l volttron.log --msgdebug
```

This runs VOLTTRON as a foreground process. To run it in the background, use:

This also enables the Message Debugger, a non-production VOLTTRON debugging aid that's used by some notebooks. To run with the Message Debugger disabled (VOLTTRON's normal state), omit the `--msgdebug` flag.

Now that VOLTTRON is running, it's ready for agent configuration and execution. Each Jupyter notebook contains detailed instructions and executable code for doing that.

1.12.5 Configure Jupyter

More detailed information about installing, configuring and using Jupyter Notebooks is available on the Project Jupyter site, http://jupyter.org/.

- Create a Jupyter configuration file:

```
$ jupyter notebook --generate-config
```

- Revise the Jupyter configuration:

Open *~/.jupyter/jupyter_notebook_config.py* in your favorite text editor. Change the configuration to accept connections from any IP address (not just from localhost) and use a specific, non-default port number:

- Un-comment `c.NotebookApp.ip` and set it to: `*` instead of `localhost`
- Un-comment `c.NotebookApp.port` and set it to: `8891` instead of `8888`

Save the config file.

- Open ports for TCP connections:

Make sure that your Jupyter server host's security rules allow inbound TCP connections on port *8891*.

If the VOLTTRON instance needs to receive TCP requests, for example ForwardHistorian or DataPuller messages from other VOLTTRON instances, make sure that the host's security rules also allow inbound TCP communications on VOLTTRON's port, which is usually *22916*.

1.12.6 Launch Jupyter

- Start the Jupyter server:

In a separate command-line shell, set up VOLTTRON's environment variables and virtual environment, and then launch the Jupyter server:

```
$ export VOLTTRON_HOME=(your volttron home directory, e.g. ~/.volttron)
$ export VOLTTRON_ROOT=(where volttron was installed; e.g. ~/repos/volttron)
$ cd $VOLTTRON_ROOT
$ source env/bin/activate
$ cd examples/JupyterNotebooks
$ jupyter notebook --no-browser
```

- Open a Jupyter client in a web browser:

Look up the host's IP address (e.g., using ifconfig). Open a web browser and navigate to the URL that was displayed when you started jupyter, replacing *localhost* with that IP address. A Jupyter web page should display, listing your notebooks.

1.13 Python for Matlab Users

Matlab is a popular proprietary programming language and tool suite with built in support for matrix operations and graphically plotting computation results. The purpose of this document is to introduce Python to those already familiar Matlab so it will be easier for them to develop tools and agents in VOLTTRON.

1.13.1 A Simple Function

Python and Matlab are similar in many respects, syntactically and semantically. With the addition of the NumPy library in Python, almost all numerical operations in Matlab can be emulated or directly translated. Here are functions in each language that perform the same operation:

```
% Matlab
function [result] = times_two(number)
    result = number * 2;
end
```

```
# Python
def times_two(number):
    result = number * 2
    return result
```

Some notes about the previous functions:

1. Values are explicitly returned with the *return* statement. It is possible to return multiple values, as in Matlab, but doing this without a good reason can lead to overcomplicated functions.
2. Semicolons are not used to end statements in python, and white space is significant. After a block is started (if, for, while, functions, classes) subsequent lines should be indented with four spaces. The block ends when the programmer stops adding the extra level of indentation.

1.13.2 Translating

The following may be helpful if you already have a Matlab file or function that will be translated into Python. Many of the syntax differences between Matlab and Python can be rectified with your text editor's find and replace feature.

Start by copying all of your Matlab code into a new file with a *.py* extension. It is recommended to start by commenting everything out and uncommenting the Matlab code in chunks. This way it is possible to write valid Python and verify it as you translate, instead of waiting till the whole file is "translated". Editors designed to work with Python should be able to highlight syntax errors as well.

1. Comments are created with a %. Find and replace these with #.

```
def test_function():
    # single line Python comment
    """
    Multi-line Python comment
    """
    pass # inline Python comment
```

1. Change *elseif* blocks to *elif* blocks.

```
if thing == 0:
    do_thing1()
elif thing ==1:
    do_thing2()
else:
    do_the_last_thing()
```

1. Python indexes start at zero instead of one. Array slices and range operations don't include the upper bound, so only the lower bound should decrease by one. The following examples are of Python code in the console:

```
>>> test_array = [0, 1, 2, 3, 4]
>>> test_array[0]
0
>>> test_array[1]
1
>>> test_array[0:2]
[0, 1]
>>>>>> test_array[:2]
[0, 1]
>>> test_array[2:]
[2, 3, 4]
>>>
```

1. Semicolons in Matlab are used to suppress output at the end of lines and for organizing array literals. After arranging the arrays into nested lists, all semicolons can be removed.
2. The *end* keyword in Matlab is used both to access the last element in an array and to close blocks. The array use case can be replaced with *-1* and the others can be removed entirely.

```
>>> test_array = [0, 1, 2, 3, 4]
>>> test_array[-1]
4
>>>
```

A More Concrete Example

In the Building Economic Dispatch project, a sibling project to VOLTTRON, a number of components written in Matlab would create a matrix out of some collection of columns and perform least squares regression using the *matrix division* operator. This is straightforward and very similar in both languages assuming that all of the columns are defined and are the same length.

```
% Matlab
XX = [U, xbp, xbp2, xbp3, xbp4, xbp5];
AA = XX \ ybp;
```

```
# Python
import numpy as np

XX = np.column_stack((U, xbp, xbp2, xbp3, xbp4, xbp5))
AA, resid, rank, s = np.linalg.lstsq(XX, ybp)
```

This pattern also included the creation of the *U* column, a column of ones used as the bias term in the linear equation . In order to make the Python version more readable and more robust, the pattern was removed from each component and replaced with a single function call to *least_squares_regression*.

This function does some validation on the input parameters, automatically creates the bias column, and returns the least squares solution to the system. Now if we want to change how the solution is calculated we only have to change the one function, instead of each instance where the pattern was written originally.

```
def least_squares_regression(inputs=None, output=None):
    if inputs is None:
        raise ValueError("At least one input column is required")
    if output is None:
        raise ValueError("Output column is required")

```

(continues on next page)

(continued from previous page)

```
    if type(inputs) != tuple:
        inputs = (inputs,)

    ones = np.ones(len(inputs[0]))
    x_columns = np.column_stack((ones,) + inputs)

    solution, resid, rank, s = np.linalg.lstsq(x_columns, output)
    return solution
```

1.13.3 Lessons Learned (sometimes the hard way)

Variable Names

Use descriptive function and variable names whenever possible. The most important things to consider here are reader comprehension and searching. Consider a variable called *hdr*. Is it *header* without any vowels, or is it short for *high-dynamic-range*? Spelling out full words in variable names can save someone else a lot of guesswork.

Searching comes in when we're looking for instances of a string or variable. Single letter variable names are impossible to search for. Variables names describing the value being stored in a concise but descriptive manner are preferred.

Matlab load/save

Matlab has built-in functions to automatically save and load variables from your programs to disk. Using these functions can lead to poor program design and should be avoided if possible. It would be best to refactor as you translate if they are being used. Few operations are so expensive that that cannot be redone every time the program is run. For part of the program that saves variables, consider making a function that simply returns them instead.

If your Matlab program is loading csv files then use the Pandas library when working in python. Pandas works well with NumPy and is the go-to library when using csv files that contain numeric data.

1.13.4 More Resources

NumPy for Matlab Users Has a nice list of common operations in Matlab and NumPy.

NumPy Homepage

Pandas Homepage

1.14 Bootstrap Process

The *bootstrap.py* Python script in the root directory of the VOLTTRON repository may be used to create VOLTTRON's Python virtual environment and install or update service agent dependencies.

The first running of *bootstrap.py* will be against the systems *python3* executable. During this initial step a virtual environment is created using the *venv* module. Additionally, all requirements for running a base volttron instance are installed. A user can specify additional arguments to the *bootstrap.py* script allowing a way to quickly install dependencies for service agents (e.g. bootstrap.py –mysql).

```
# boostrap with additional dependency requirements for web enabled agents.
user@machine$ python3 bootstrap.py --web
```

After activating an environment (source env/bin/activate) one can use the *bootstrap.py* script to install more service agent dependencies by executing the same boostrap.py command.

Note: In the following example one can tell the environment is activated based upon the (volttron) prefix to the command prompt

```
# Adding additional database requirement for crate
(volttron) user@machine$ python3 bootstrap.py --crate
```

If a fresh install is necessary one can use the –force argument to rebuild the virtual environment from scratch.

```
# Rebuild the environment from the system's python3
user@machine$ python3 bootstrap.py --force
```

Note: Multiple options can be specified on the command line *python3 bootstrap.py –web –crate* installs dependencies for web enabled agents as well as the Crate database historian.

1.14.1 Bootstrap Options

The *bootstrap.py* script takes several options that allow customization of the environment, installing and update packages, and setting the package locations. The following sections can be reproduced by executing:

```
# Show the help output from bootstrap.py
user@machine$ python3 bootstrap --help
```

The options for customizing the location of the virtual environment are as follows.

```
--envdir VIRTUAL_ENV  alternate location for virtual environment
--force               force installing in non-empty directory
-o, --only-virtenv    create virtual environment and exit (skip install)
--prompt PROMPT       provide alternate prompt in activated environment
                      (default: volttron)
```

Additional options are available for customizing where an environment will retrieve packages and/or upgrade existing packages installed.

```
update options:
  --offline             install from cache without downloading
  -u, --upgrade         upgrade installed packages
  -w, --wheel           build wheels in the pip wheelhouse
```

To help boostrap an environment in the shortest number of steps we have grouped dependency packages under named collections. For example, the –web argument will install six different packages from a single call to boostrap.py –web. The following collections are available to use.

```
...

Extra packaging options:
  --all             All dependency groups.
  --crate           Crate database adapter
  --databases       All of the databases (crate, mysql, postgres, etc).
  --dnp3            Dependencies for the dnp3 agent.
```

(continues on next page)

(continued from previous page)

```
  --documentation   All dependency groups to allow generation of documentation␣
↪without error.
  --drivers         All drivers known to the platform driver.
  --influxdb        Influx database adapter
  --market          Base market agent dependencies
  --mongo           Mongo database adapter
  --mysql           Mysql database adapter
  --pandas          Pandas numerical analysis tool
  --postgres        Postgres database adapter
  --testing         A variety of testing tools for running unit/integration tests.
  --web             Packages facilitating the building of web enabled agents.
  --weather         Packages for the base weather agent

rabbitmq options:
  --rabbitmq [RABBITMQ]
                        install rabbitmq server and its dependencies. optional
                        argument: Install directory that exists and is
                        writeable. RabbitMQ server will be installed in a
                        subdirectory.Defaults to /home/osboxes/rabbitmq_server

...
```

1.15 Platform Configuration

Each instance of the VOLTTRON platform includes a *config* file which is used to configure the platform instance on startup. This file is kept in *VOLTTRON_HOME* and is created using the *volttron-cfg* (*vcfg*) command, or will be created with default values on start up of the platform otherwise.

Following is helpful information about the *config* file and the *vcfg* command.

1.15.1 VOLTTRON Environment

By default, the VOLTTRON projects bases its files out of *VOLTTRON_HOME* which defaults to *~/.volttron*.

- `$VOLTTRON_HOME/agents` contains the agents installed on the platform
- `$VOLTTRON_HOME/certificates` contains the certificates for use with the Licensed VOLTTRON code.
- `$VOLTTRON_HOME/run` contains files create by the platform during execution. The main ones are the 0MQ files created for publish and subcribe.
- `$VOLTTRON_HOME/ssh` keys used by agent mobility in the Licensed VOLTTRON code
- `$VOLTTRON_HOME/config` Default location to place a config file to override any platform settings.
- `$VOLTTRON_HOME/packaged` is where agent packages created with *volttron-pkg* are created

1.15.2 VOLTTRON Config File

The VOLTTRON platform config file can contain any of the command line arguments for starting the platform…

```
-c FILE, --config FILE
                 read configuration from FILE
-l FILE, --log FILE   send log output to FILE instead of stderr
```

(continues on next page)

(continued from previous page)

```
-L FILE, --log-config FILE
                 read logging configuration from FILE
-q, --quiet           decrease logger verboseness; may be used multiple
                 times
-v, --verbose         increase logger verboseness; may be used multiple
                 times
--verboseness LEVEL   set logger verboseness
--help                show this help message and exit
--version             show program's version number and exit
```

agent options:

```
--autostart           automatically start enabled agents and services
--publish-address ZMQADDR
                     ZeroMQ URL for used for agent publishing
--subscribe-address ZMQADDR
                     ZeroMQ URL for used for agent subscriptions
```

control options:

```
--control-socket FILE
                      path to socket used for control messages
--allow-root          allow root to connect to control socket
--allow-users LIST    users allowed to connect to control socket
--allow-groups LIST   user groups allowed to connect to control socket
```

Boolean options, which take no argument, may be inverted by prefixing the option with no '-' (e.g. `--autostart` may be inverted using `--no-autostart`).

1.15.3 VOLTTRON Config

The *volttron-cfg* or *vcfg* command allows for an easy configuration of the VOLTTRON environment. The command includes the ability to set up the platform configuration, an instance of the platform historian, VOLTTRON Central UI, and VOLTTRON Central Platform agent.

Running *vcfg* will create a *config* file in *VOLTTRON_HOME* which will be populated according to the answers to prompts. This process should be repeated for each platform instance, and can be re-run to reconfigure a platform instance.

Note: To create a simple instance of VOLTTRON, leave the default response, or select yes (y) if prompted for a yes or no response [Y/N]. You must choose a username and password for the VOLTTRON Central admin account if selected.

A set of example responses are included here (*username* is `user`, *localhost* is `volttron-pc`):

```
(volttron) user@volttron-pc:~/volttron$ vcfg

Your VOLTTRON_HOME currently set to: /home/user/.volttron

Is this the volttron you are attempting to setup? [Y]:
What type of message bus (rmq/zmq)? [zmq]:
What is the vip address? [tcp://127.0.0.1]:
What is the port for the vip address? [22916]:
```

(continues on next page)

(continued from previous page)

```
Is this instance web enabled? [N]: y
What is the protocol for this instance? [https]:
Web address set to: https://volttron-pc
What is the port for this instance? [8443]:
Would you like to generate a new web certificate? [Y]:
WARNING! CA certificate does not exist.
Create new root CA? [Y]:

Please enter the following details for web server certificate:
    Country: [US]:
    State: WA
    Location: Richland
    Organization: PNNL
    Organization Unit: VOLTTRON
Created CA cert
Creating new web server certificate.
Is this an instance of volttron central? [N]: y
Configuring /home/user/volttron/services/core/VolttronCentral.
Installing volttron central.
['volttron', '-vv', '-l', '/home/user/.volttron/volttron.cfg.log']
Should the agent autostart? [N]: y
VC admin and password are set up using the admin web interface.
After starting VOLTTRON, please go to https://volttron-pc:8443/admin/login.html to␣
↪complete the setup.
Will this instance be controlled by volttron central? [Y]:
Configuring /home/user/volttron/services/core/VolttronCentralPlatform.
What is the name of this instance? [volttron1]:
Volttron central address set to https://volttron-pc:8443
['volttron', '-vv', '-l', '/home/user/.volttron/volttron.cfg.log']
Should the agent autostart? [N]: y
Would you like to install a platform historian? [N]: y
Configuring /home/user/volttron/services/core/SQLHistorian.
['volttron', '-vv', '-l', '/home/user/.volttron/volttron.cfg.log']
Should the agent autostart? [N]: y
Would you like to install a master driver? [N]: y
Configuring /home/user/volttron/services/core/MasterDriverAgent.
['volttron', '-vv', '-l', '/home/user/.volttron/volttron.cfg.log']
Would you like to install a fake device on the master driver? [N]: y
Should the agent autostart? [N]: y
Would you like to install a listener agent? [N]: y
Configuring examples/ListenerAgent.
['volttron', '-vv', '-l', '/home/user/.volttron/volttron.cfg.log']
Should the agent autostart? [N]: y
Finished configuration!

You can now start the volttron instance.

If you need to change the instance configuration you can edit
the config file is at /home/user/.volttron/config
```

Once this is finished, run VOLTTRON and test the new configuration.

1.15.4 Optional Arguments

- **-v, –verbose** - Enables verbose output in standard-output (PIP output, etc.)

- **–vhome VHOME** - Provide a path to set *VOLTTRON_HOME* for this instance
- **–instance-name INSTANCE_NAME** - Provide a name for this instance. Required for running secure agents mode
- **–list-agents** - Display a list of configurable agents (Listener, Master Driver, Platform Historian, VOLTTRON Central, VOLTTRON Central Platform)
- **–agent AGENT [AGENT . . .]** - Configure listed agents
- **–rabbitmq RABBITMQ [RABBITMQ . . .]** - Configure rabbitmq for single instance, federation, or shovel either based on configuration file in yml format or providing details when prompted.

  Usage:

  ```
  vcfg --rabbitmq single|federation|shovel [rabbitmq config file]``
  ```

- **–secure-agent-users** - Require that agents run as their own Unix users (this requires running *scripts/secure_user_permissions.sh* as *sudo*)

1.16 Planning a Deployment

The 3 major installation types for VOLTTRON are doing development, doing research using VOLTTRON, and collecting and managing physical devices.

Development and Research installation tend to be smaller footprint installations. For development, the data is usually synthetic or copied from another source. The existing documentation covers development installs in significant detail.

Other deployments will have a better installation experience if they consider certain kinds of questions while they plan their installation.

1.16.1 Questions

- Do you want to send commands to the machines ?
- Do you want to store the data centrally ?
- How many machines do you expect to collect data from on each "collector" ?
- How often will the machines collect data ?
- Are all the devices visible to the same network ?
- What types of VOLTTRON applications do you want to run ?

Commands

If you wish to send commands to the devices, you will want to install and configure the Volttron Central agent. If you are only using VOLTTRON to securely collect the data, you can turn off the extra agents to reduce the footprint.

Storing Data

VOLTTRON supports multiple historians. MySQL and MongoDB are the most commonly used. As you plan your installation, you should consider how quickly you need access to the data and where. If you are looking at the health and well-being of an entire suite of devices, its likely that you want to do that from a central location. Analytics can be performed at the edge by VOLTTRON applications or can be performed across the data usually from a central data repository. The latency that you can tolerate in your data being available will also determine choices in different agents (ForwardHistorian versus Data Mover)

How Many

The ratio of how many devices-to-collector machine is based on several factors. These include:

- how much memory and network bandwidth the collection machine has. More = More devices
- how fast the local storage is can affect how fast the data cache can be written. Very slow storage devices can fall behind

The second half of the "how many" question is how many collector platforms are writing to a single VOLTTRON platform to store data - and whether that storage is local, remote, big enough, etc.

If you are storing more than moderate amount of data, you will probably benefit from installing your database on a different machine than your concrete historian machine.

Note: This is contra-indicated if you have a slow network connection between you concrete historian and your database machine.

In synthetic testing up to 6 virtual machines hosting 500 devices each (18 points) were easily supported by a single centralized platform writing to a Mongo database - using a high speed network. That central platform experienced very little CPU or memory load when the VOLTTRON Central agent was disabled.

How Often

This question is closely related to the last. A higher sampling frequency will create more data. This will place more work in the storage phase.

Networks

In many cases, there are constraints on how networks can interact with each other. In many cases, these include security considerations. On some sites, the primary network will be protected from less secure networks and may require different installation considerations. For example, if a data collector machine and the database machine are on the same network with sufficient security, you may choose to have the data collector write directly to the database. If the collector is on an isolated building network then you will likely need to use the ForwardHistorian to bridge the two networks.

Other Considerations

Physical location and maintenance of collector machines must be considered in all live deployments. Although the number of data points may imply a heavy load on a data collection box, the physical constraints may limit the practicality of having more than a single box. The other side of that discussion is deploying many collector boxes may be simpler initially, but may create a maintenance challenge if you don't plan ahead on how you apply patches, etc.

Naming conventions should also be considered. The ability to trace data through the system and identify the collector machine and device can be invaluable in debugging and analysis.

1.16.2 Deployment Options

There are several ways to deploy the VOLTTRON platform in a Linux environment. It is up to the user to determine which is right for them. The following assumes that the platform has already been bootstrapped and is ready to run.

Simple Command Line

With the VOLTTRON environment activated the platform can be started simply by running VOLTTRON on the command line.

```
$volttron -vv
```

This will start the platform in the current terminal with very verbose logging turned on. This is most appropriate for testing Agents or testing a deployment for problems before switching to a more long term solution. This will print all log messages to the console in real time.

This should not be used for long term deployment. As soon as an SSH session is terminated for whatever reason the processes attached to that session will be killed. This also will not capture log message to a file.

Running VOLTTRON as a Background Process

A simple, more long term solution, is to run volttron in the background and disown it from the current terminal.

Warning: If you plan on running VOLTTRON in the background and detaching it from the terminal with the `disown` command be sure to redirect stderr and stdout to `/dev/null`. Even if logging to a file is used some libraries which VOLTTRON relies on output directly to stdout and stderr. This will cause problems if those file descriptors are not redirected to `/dev/null`.

```
$volttron -vv -l volttron.log > /dev/null 2>&1&
```

Alternatively:

```
``./start-volttron``
```

Note: If you are not in an activated environment, this script will start the platform running in the background in the correct environment, however the environment will not be activated for you, you must activate it yourself.

If there are other jobs running in your terminal be sure to disown the correct one.

```
$jobs
[1]+  Running                 something else
[2]+  Running                 ./start-volttron

#Disown VOLTTRON
$disown %2
```

This will run the VOLTTRON platform in the background and turn it into a daemon. The log output will be directed to a file called `volttron.log` in the current directory.

To keep the size of the log under control for more longer term deployments us the rotating log configuration file `examples/rotatinglog.py`.

```
$volttron -vv --log-config examples/rotatinglog.py > /dev/null 2>&1&
```

This will start a rotate the log file at midnight and limit the total log data to seven days worth.

The main downside to this approach is that the VOLTTRON platform will not automatically resume if the system is restarted. It will need to be restarted manually after reboot.

Setting up VOLTTRON as a System Service

Systemd

An example service file `scripts/admin/volttron.service` for systemd cas be used as a starting point for setting up VOLTTRON as a service. Note that as this will redirect all the output that would be going to stdout - to the syslog. This can be accessed using *journalctl*. For systems that run all the time or have a high level of debugging turned on, we recommend checking the system's logrotate settings.

```
[Unit]
Description=VOLTTRON Platform Service
After=network.target

[Service]
Type=simple

#Change this to the user that VOLTTRON will run as.
User=volttron
Group=volttron

#Uncomment and change this to specify a different VOLTTRON_HOME
#Environment="VOLTTRON_HOME=/home/volttron/.volttron"

#Change these to settings to reflect the install location of VOLTTRON
WorkingDirectory=/var/lib/volttron
ExecStart=/var/lib/volttron/env/bin/volttron -vv
ExecStop=/var/lib/volttron/env/bin/volttron-ctl shutdown --platform


[Install]
WantedBy=multi-user.target
```

After the file has been modified to reflect the setup of the platform you can install it with the following commands. These need to be run as root or with sudo as appropriate.

```
#Copy the service file into place
cp scripts/admin/volttron.service /etc/systemd/system/

#Set the correct permissions if needed
chmod 644 /etc/systemd/system/volttron.service

#Notify systemd that a new service file exists (this is crucial!)
systemctl daemon-reload

#Start the service
systemctl start volttron.service
```

Init.d

An example init script `scripts/admin/volttron` can be used as a starting point for setting up VOLTTRON as a service on init.d based systems.

Minor changes may be needed for the file to work on the target system. Specifically the `USER`, `VLHOME`, and `VOLTTRON_HOME` variables may need to be changed.

```
...
#Change this to the user VOLTTRON will run as.
USER=volttron
#Change this to the install location of VOLTTRON
VLHOME=/var/lib/volttron

...

#Uncomment and change this to specify a different VOLTTRON_HOME
#export VOLTTRON_HOME=/home/volttron/.volttron
```

The script can be installed with the following commands. These need to be run as root or with *sudo* as appropriate.

```
#Copy the script into place
cp scripts/admin/volttron /etc/init.d/

#Make the file executable
chmod 755 /etc/init.d/volttron

#Change the owner to root
chown root:root /etc/init.d/volttron

#These will set it to startup automatically at boot
update-rc.d volttron defaults

#Start the service
/etc/init.d/volttron start
```

1.17 Single Machine

The purpose of this demonstration is to show the process of setting up a simple VOLTTRON instance for use on a single machine.

Note: The simple deployment example below considers only the ZeroMQ deployment scenario. For RabbitMQ deployments, read and perform the RabbitMQ installation steps from the *platform installation* instructions and configuration steps from *VOLTTRON Config*.

1.17.1 Install and Build VOLTTRON

First, *install* VOLTTRON:

For a quick reference for Ubuntu machines:

```
sudo apt-get update
sudo apt-get install build-essential libffi-dev python3-dev python3-venv openssl␣
↪libssl-dev libevent-dev git
git clone https://github.com/VOLTTRON/volttron/
cd volttron
python3 bootstrap.py --drivers --databases
```

Note: For additional detail and more information on installing in other environments, please see the *platform install* section. See the *bootstrap process* docs for more information on its operation and available options.

Activate the Environment

After the build is complete, activate the VOLTTRON environment.

```
source env/bin/activate
```

Run VOLTTRON Config

The *volttron-cfg* or *vcfg* commands can be used to configure platform communication. For an example single machine deployment, most values can be left at their default values. The following is a simple case example of running *vcfg*:

```
(volttron) user@volttron-pc:~/volttron$ vcfg

 Your VOLTTRON_HOME currently set to: /home/james/.volttron

 Is this the volttron you are attempting to setup? [Y]:
 What type of message bus (rmq/zmq)? [zmq]:
 What is the vip address? [tcp://127.0.0.1]:
 What is the port for the vip address? [22916]:
 Is this instance web enabled? [N]:
 Will this instance be controlled by volttron central? [Y]: N
 Would you like to install a platform historian? [N]:
 Would you like to install a master driver? [N]:
 Would you like to install a listener agent? [N]:
 Finished configuration!

 You can now start the volttron instance.

 If you need to change the instance configuration you can edit
 the config file is at /home/james/.volttron/config
```

To learn more, read the *volttron-config* section of the Platform Features docs.

Note: Steps below highlight manually installing some example agents. To skip manual install, supply *y* or *Y* for the `platform historian`, `master driver` and `listener agent` installation options.

Start VOLTTRON

The most convenient way to start the platform is with the *.start-volttron* command (from the volttron root directory).

```
./start-volttron
```

The output following the platform starting successfully will appear like this:

```
2020-10-27 11:34:33,593 () volttron.platform.agent.utils DEBUG: value from env None
2020-10-27 11:34:33,593 () volttron.platform.agent.utils DEBUG: value from config␣
↪False
2020-10-27 11:34:35,656 () root DEBUG: Creating ZMQ Core config.store
2020-10-27 11:34:35,672 () volttron.platform.store INFO: Initializing configuration␣
↪store service.
2020-10-27 11:34:35,717 () root DEBUG: Creating ZMQ Core platform.auth
2020-10-27 11:34:35,728 () volttron.platform.auth INFO: loading auth file /home/james/
↪.volttron/auth.json
2020-10-27 11:34:35,731 () volttron.platform.auth INFO: auth file /home/james/.
↪volttron/auth.json loaded
2020-10-27 11:34:35,732 () volttron.platform.agent.utils INFO: Adding file watch for /
↪home/james/.volttron/auth.json dirname=/home/james/.volttron, filename=auth.json
2020-10-27 11:34:35,734 () volttron.platform.agent.utils INFO: Added file watch for /
↪home/james/.volttron/auth.json
2020-10-27 11:34:35,734 () volttron.platform.agent.utils INFO: Adding file watch for /
↪home/james/.volttron/protected_topics.json dirname=/home/james/.volttron,␣
↪filename=protected_topics.json
2020-10-27 11:34:35,736 () volttron.platform.agent.utils INFO: Added file watch for /
↪home/james/.volttron/protected_topics.json
2020-10-27 11:34:35,737 () volttron.platform.vip.pubsubservice INFO: protected-topics␣
↪loaded
2020-10-27 11:34:35,739 () volttron.platform.vip.agent.core INFO: Connected to␣
↪platform: router: fc054c9f-aa37-4842-a618-6e70d53530f0 version: 1.0 identity:␣
↪config.store
2020-10-27 11:34:35,743 () volttron.platform.vip.agent.core INFO: Connected to␣
↪platform: router: fc054c9f-aa37-4842-a618-6e70d53530f0 version: 1.0 identity:␣
↪platform.auth
2020-10-27 11:34:35,746 () volttron.platform.vip.pubsubservice INFO: protected-topics␣
↪loaded
2020-10-27 11:34:35,750 () volttron.platform.vip.agent.subsystems.configstore DEBUG:␣
↪Processing callbacks for affected files: {}
2020-10-27 11:34:35,879 () root DEBUG: Creating ZMQ Core control
2020-10-27 11:34:35,908 () root DEBUG: Creating ZMQ Core keydiscovery
2020-10-27 11:34:35,913 () root DEBUG: Creating ZMQ Core pubsub
2020-10-27 11:34:35,924 () volttron.platform.auth INFO: loading auth file /home/james/
↪.volttron/auth.json
2020-10-27 11:34:38,010 () volttron.platform.vip.agent.core INFO: Connected to␣
↪platform: router: fc054c9f-aa37-4842-a618-6e70d53530f0 version: 1.0 identity:␣
↪control
2020-10-27 11:34:38,066 () volttron.platform.vip.agent.core INFO: Connected to␣
↪platform: router: fc054c9f-aa37-4842-a618-6e70d53530f0 version: 1.0 identity: pubsub
2020-10-27 11:34:38,069 () volttron.platform.vip.agent.core INFO: Connected to␣
↪platform: router: fc054c9f-aa37-4842-a618-6e70d53530f0 version: 1.0 identity:␣
↪keydiscovery
2020-10-27 11:34:38,429 () volttron.platform.auth WARNING: Attempt 1 to get peerlist␣
↪failed with exception 0.5 seconds
2020-10-27 11:34:38,430 () volttron.platform.auth WARNING: Get list of peers from␣
↪subsystem directly
2020-10-27 11:34:38,433 () volttron.platform.auth INFO: auth file /home/james/.
↪volttron/auth.json loaded
2020-10-27 11:34:38,434 () volttron.platform.auth INFO: loading auth file /home/james/
↪.volttron/auth.json
```

(continues on next page)

(continued from previous page)

```
2020-10-27 11:34:40,961 () volttron.platform.auth WARNING: Attempt 1 to get peerlist␣
↪failed with exception 0.5 seconds
2020-10-27 11:34:40,961 () volttron.platform.auth WARNING: Get list of peers from␣
↪subsystem directly
2020-10-27 11:34:40,969 () volttron.platform.auth INFO: auth file /home/james/.
↪volttron/auth.json loaded
```

Note: While running the platform with verbose logging enabled, the *volttron.log* file is useful for confirming successful platform operations or debugging. It is commonly recommended to open a new terminal window and run the following command to view the VOLTTRON logs as they are created:

```
tail -f volttron.log
```

1.17.2 Install Agents and Historian

Out of the box, VOLTTRON includes a number of agents which may be useful for single machine deployments:

- historians - Historians automatically record a data from a number of topics published to the bus. For more information on the historian framework or one of the included concrete implementations, view the *docs*
- Listener - This example agent can be useful for debugging drivers or other agents publishing to the bus. *docs*
- Master Driver - The *Master Driver* is responsible for managing device communication on a platform instance.
- weather agents - weather agents can be used to collect weather data from sources like Weather.gov

Note: The *services/core*, *services/ops*, and *examples* directories in the repository contain additional agents to use to fit individual use cases.

For a simple setup example, a Master Driver, SQLite Historian, and Listener are installed using the following steps:

1. Create a configuration file for the Master Driver and SQLite Historian (it is advised to create a *configs* directory in volttron root to keep configs for a deployment). For information on how to create configurations for these agents, view their docs:

   - *Master Driver*
   - *SQLite Historian*
   - *Listener*

   For a simple example, the configurations can be copied as-is to the *configs* directory:

   ```
   cp services/core/MasterDriverAgent/master-driver.agent configs
   cp services/core/SQLHistorian/config.sqlite configs
   cp examples/ListenerAgent/config configs/listener.config
   ```

2. Use the *install-agent.py* script to install the agent on the platform:

```
python scripts/install-agent.py -s services/core/SQLHistorian -c configs/config.
↪sqlite --tag listener
python scripts/install-agent.py -s services/core/MasterDriverAgent -c configs/master-
↪driver.agent --tag master_driver
python scripts/install-agent.py -s examples/ListenerAgent -c configs/listener.config -
↪-tag platform_historian

.. note::

   The `volttron.log` file will contain logging indicating that the agent has␣
↪installed successfully.

   .. code-block:: console

      2020-10-27 11:42:08,882 () volttron.platform.auth INFO: AUTH: After␣
↪authenticate user id: control.connection, b'c61dff8e-f362-4906-964f-63c32b99b6d5'
      2020-10-27 11:42:08,882 () volttron.platform.auth INFO: authentication success:␣
↪userid=b'c61dff8e-f362-4906-964f-63c32b99b6d5' domain='vip', address=
↪'localhost:1000:1000:3249', mechanism='CURVE', credentials=[
↪'ZrDvPG4JNLE26GoPUrTP22rV0PV8uGCnrXThrNFk_Ec'], user='control.connection'
      2020-10-27 11:42:08,898 () volttron.platform.aip DEBUG: Using name template
↪"listeneragent-3.3_{n}" to generate VIP ID
      2020-10-27 11:42:08,899 () volttron.platform.aip INFO: Agent b3e7053c-28e8-414f-
↪b685-8522eb230c7a setup to use VIP ID listeneragent-3.3_1
      2020-10-27 11:42:08,899 () volttron.platform.agent.utils DEBUG: missing file /
↪home/james/.volttron/agents/b3e7053c-28e8-414f-b685-8522eb230c7a/listeneragent-3.3/
↪listeneragent-3.3.dist-info/keystore.json
      2020-10-27 11:42:08,899 () volttron.platform.agent.utils INFO: creating file /
↪home/james/.volttron/agents/b3e7053c-28e8-414f-b685-8522eb230c7a/listeneragent-3.3/
↪listeneragent-3.3.dist-info/keystore.json
      2020-10-27 11:42:08,899 () volttron.platform.keystore DEBUG: calling generate␣
↪from keystore
      2020-10-27 11:42:08,909 () volttron.platform.auth INFO: loading auth file /home/
↪james/.volttron/auth.json
      2020-10-27 11:42:11,415 () volttron.platform.auth WARNING: Attempt 1 to get␣
↪peerlist failed with exception 0.5 seconds
      2020-10-27 11:42:11,415 () volttron.platform.auth WARNING: Get list of peers␣
↪from subsystem directly
      2020-10-27 11:42:11,419 () volttron.platform.auth INFO: auth file /home/james/.
↪volttron/auth.json loaded
```

1. Use the *vctl status* command to ensure that the agents have been successfully installed:

```
vctl status
```

```
(volttron)user@volttron-pc:~/volttron$ vctl status
  AGENT                    IDENTITY             TAG                  STATUS          ␣
↪HEALTH
8 listeneragent-3.2        listeneragent-3.2_1 listener
0 master_driveragent-3.2   platform.driver      master_driver
3 sqlhistorianagent-3.7.0  platform.historian   platform_historian
```

Note: After installation, the *STATUS* and *HEALTH* columns of the *vctl status* command will be vacant, indicating that the agent is not running. The *–start* option can be added to the *install-agent.py* script arguments to automatically start agents after they have been installed.

1.17.3 Install a Fake Driver

The following are the simplest steps for installing a fake driver for example use. For more information on installing concrete drivers such as the BACnet or Modbus drivers, view their respective documentation in the *Driver framework* section.

Note: This section will assume the user has created a *configs* directory in the volttron root directory, activated the Python virtual environment, and started the platform as noted above.

```
cp examples/configurations/drivers/fake.config <VOLTTRON root>/configs
cp examples/configurations/drivers/fake.csv <VOLTTRON root>/configs
vctl config store platform.driver devices/campus/building/fake configs/fake.config
vctl config store platform.driver fake.csv devices/fake.csv
```

Note: For more information on the fake driver, or the configurations used in the above example, view the *docs*

1.17.4 Testing the Deployment

To test that the configuration was successful, start an instance of VOLTTRON in the background:

```
./start-volttron
```

Note: This command must be run from the root VOLTTRON directory.

Having following the examples above, the platform should be ready for demonstrating the example deployment. Start the Listener, SQLite historian and Master Driver.

```
vctl start --tag listener platform_historian master_driver
```

The output should look similar to this:

```
(volttron)user@volttron-pc:~/volttron$ vctl status
  AGENT                        IDENTITY            TAG                STATUS          ␣
↪HEALTH
8 listeneragent-3.2            listeneragent-3.2_1 listener           running [2810]   GOOD
0 master_driveragent-3.2       platform.driver     master_driver      running [2813]   GOOD
3 sqlhistorianagent-3.7.0      platform.historian  platform_historian running [2811]   GOOD
```

Note: The *STATUS* column indicates whether the agent is running. The *HEALTH* column indicates whether the current state of the agent is within intended parameters (if the Master Driver is publishing, the platform historian has not been backlogged, etc.)

You can further verify that the agents are functioning correctly with `tail -f volttron.log`.

ListenerAgent:

```
2020-10-27 11:43:33,997 (listeneragent-3.3 3294) __main__ INFO: Peer: pubsub, Sender:␣
↪listeneragent-3.3_1:, Bus: , Topic: heartbeat/listeneragent-3.3_1, Headers: {
↪'TimeStamp': '2020-10-27T18:43:33.988561+00:00', 'min_compatible_version': '3.0',
↪'max_compatible_version': ''}, Message:
'GOOD'
```

Master Driver with Fake Driver:

```
2020-10-27 11:47:50,037 (listeneragent-3.3 3294) __main__ INFO: Peer: pubsub, Sender:␣
↪platform.driver:, Bus: , Topic: devices/campus/building/fake/all, Headers: {'Date':
↪'2020-10-27T18:47:50.005349+00:00', 'TimeStamp': '2020-10-27T18:47:50.005349+00:00',
↪ 'SynchronizedTimeStamp': '2020-10-27T18:47:50.000000+00:00', 'min_compatible_
↪version': '3.0', 'max_compatible_version': ''}, Message:
 [{'EKG': -0.8660254037844386,
  'EKG_Cos': -0.8660254037844386,
  'EKG_Sin': -0.8660254037844386,
  'Heartbeat': True,
  'OutsideAirTemperature1': 50.0,
  'OutsideAirTemperature2': 50.0,
  'OutsideAirTemperature3': 50.0,
  'PowerState': 0,
  'SampleBool1': True,
  'SampleBool2': True,
  'SampleBool3': True,
  'SampleLong1': 50,
  ...
```

SQLite Historian:

```
2020-10-27 11:50:25,021 (master_driveragent-4.0 3535) master_driver.driver DEBUG:␣
↪finish publishing: devices/campus/building/fake/all
2020-10-27 11:50:25,052 (sqlhistorianagent-3.7.0 3551) volttron.platform.dbutils.
↪sqlitefuncts DEBUG: Managing store - timestamp limit: None  GB size limit: None
```

1.18 Multi-Platform Connection

There are multiple ways to establish connection between external VOLTTRON platforms. Given that VOLTTRON now supports ZeroMq and RabbitMQ type of message bus with each using different type authentication mechanism, the number of different ways that agents can connect to external platforms has significantly increased. Various multi-platform deployment scenarios will be covered in this section.

1. Agents can directly connect to external platforms to send and receive messages. Forward historian, Data Mover agents fall under this category. The deployment steps for forward historian is described in *Forward Historian Deployment* and data mover historian in *DataMover Historian Deployment*
2. The platforms maintain the connection with other platforms and agents can send to and receive messages from external platforms without having to establish connection directly. The deployment steps is described in *Multi Platform Router Deployment*
3. RabbitMQ has ready made plugins such as shovel and federation to connect to external brokers. This feature is leveraged to make connections to external platforms. This is described in *Multi Platform RabbitMQ Deployment*
4. A web based admin interface to authenticate multiple instances (ZeroMq or RabbitMQ) wanting to connect to single central instance is now available. The deployment steps is described in *Multi Platform Multi-Bus Deployment*

5. VOLTTRON Central is a platform management web application that allows platforms to communicate and to be managed from a centralized server. The deployment steps is described in *VOLTTRON Central Demo*

1.18.1 Assumptions

- *Data Collector* is the deployment box that has the drivers and is collecting data from devices which will be forwarded to a *VOLTTRON Central*.
- *Volttron Central (VC)* is the deployment box that has the historian which will save data from all Data Collectors to the central database.
- *VOLTTRON_HOME* is assumed to the default on both boxes (*/home/<user>/.volttron*).

Note: `VOLTTRON_HOME` is the directory used by the platform for managing state and configuration of the platform and agents installed locally on the platform. Auth keys, certificates, the configuration store, etc. are stored in this directory by the platform.

Forward Historian

This guide describes a simple setup where one VOLTTRON instance collects data from a fake devices and sends to another instance . Lets consider the following example.

We are going to create two VOLTTRON instances and send data from one VOLTTRON instance running a fake driver(subscribing values from a fake device) and sending the values to the second VOLTTRON instance.

VOLTTRON instance 1 forwards data to VOLTTRON instance 2

VOLTTRON instance 1

- `vctl shutdown -platform` (if the platform is already working)
- `vcfg` (this helps in configuring the volttron instance http://volttron.readthedocs.io/en/releases-4.1/core_services/control/VOLTTRON-Config.html
  - Specify the IP of the machine: `tcp://130.20.*.*:22916`
  - Specify the port you want to use
  - Specify if you want to run VC(Volttron Central) here or this this instance would be controlled by a VC and the IP and port of the VC
    - Then install agents like Master Driver Agent with a fake driver for the instance.
    - Install a listener agent so see the topics that are coming from the diver agent
    - Then run the volttron instance by using the following command: `./start-volttron`
- Volttron authentication: We need to add the IP of the instance 2 in the *auth.config* file of the VOLTTRON agent. This is done as follows:
  - `vctl auth-add`
  - We specify the IP of the instance 2 and the credentials of the agent (read *Agent Authentication*
  - For specifying authentication for all the agents , we specify `/.*/`
  - This should enable authentication for all the volttron-instance based on the IP you specify here

For this documentation, the topics from the driver agent will be send to the instance 2

- We use the existing agent called the Forward Historian for this purpose which is available in service/core in the VOLTTRON directory.
- In the config file under the Forward Historian directory, we modify the following fields:
  - Destination-vip: the IP of the volttron instance to which we have to forward the data to along with the port number. Example : `tcp://130.20.*.*:22916`
  - Destination-serverkey: The server key of the VOLTTRON instance to which we need to forward the data to. This can be obtained at the VOLTTRON instance by typing `vctl auth serverkey`
- Service_topic_list: specify the topics you want to forward specifically instead of all the values.
- Once the above values are set, your forwarder is all set .
- You can create a script file for the same and execute the agent.

VOLTTRON instance 2

- `vctl shutdown -platform` (if the platform is already working)
- `volttron-cfg` (this helps in configuring the volttron instance) http://volttron.readthedocs.io/en/releases-4.1/core_services/control/VOLTTRON-Config.html
  - Specify the IP of the machine : `tcp://130.20.*.*:22916`
  - Specify the port you want to use.
  - Install the listener agent (this will show the connection from instance 1 if its successful and then show all the topics from instance 1.
- Volttron authentication: We need to add the IP of the instance 1 in the auth.config file of the VOLTTRON agent . This is done as follows:
  - `vctl auth-add`
  - We specify the IP of the instance 1 and the credentials of the agent
  - For specifying authentication for all the agents , we specify `/.*/`
  - This should enable authentication for all the volttron-instance based on the IP you specify here

Listener Agent

Run the listener agent on this instance to see the values being forwarded from instance 1. Once the above setup is done, you should be able to see the values from instance 1 on the listener agent of instance 2.

DataMover Historian

This guide describes how a DataMover historian can be used to transfer data from one VOLTTRON instance to another. The DataMover historian is different from Forward historian in the way it sends the data to the remote instance. It first batches the data and makes a RPC call to a remote historian instead of publishing data on the remote message bus instance. The remote historian then stores the data into it's database.

The walk-through below demonstrates how to setup DataMover historian to send data from one VOLTTRON instance to another.

VOLTTRON instance 1 sends data to platform historian on VOLTTRON instance 2

As an example two VOLTTRON instances will be created and to send data from one VOLTTRON instance running a fake driver (subscribing to publishes from a fake device) and sending the values to a remote historian running on the second VOLTTRON instance.

VOLTTRON instance 1

- `vctl shutdown -platform` (if the platform is already working)
- `volttron-cfg` (this helps in configuring the volttron instance http://volttron.readthedocs.io/en/releases-4.1/core_services/control/VOLTTRON-Config.html
  - Specify the VIP address of the instance: `tcp://127.0.0.1:22916`
  - Install Master Driver Agent with a fake driver for the instance.
  - Install a listener agent so see the topics that are coming from the diver agent
- Then run the volttron instance by using the following command: `./start-volttron`

VOLTTRON instance 2

- `vctl shutdown -platform` (if the platform is already working)
- `volttron-cfg` (this helps in configuring the volttron instance) http://volttron.readthedocs.io/en/releases-4.1/core_services/control/VOLTTRON-Config.html
  - Specify the VIP address of the instance : `tcp://127.0.0.2:22916`
  - Install a platform historian. `volttron-cfg` installs a default SQL historian.
- Start the VOLTTRON instance by using following command: `./start-volttron`

DataMover Configuration

An example config file is available in `services/core/DataMover/config`. We need to update the *destination-vip*, *destination-serverkey*, and *destination-historian-identity* entries as per our setup.

Note: Here the topics from the driver on VOLTTRON instance 1 will be sent to instance 2.

- **destination-vip**: The VIP address of the volttron instance to which we need to send data. Example : `tcp://127.0.0.2:22916`
- **destination-serverkey**: The server key of remote VOLTTRON instance - Get the server key of VOLTTRON instance 2 and set *destination-serverkey* property with the server key

  ```
  vctl auth serverkey
  ```

- destination-historian-identity: Identity of remote platform historian. Default is “platform.historian”

Running DataMover Historian

- Install the DataMover historian on the VOLTTRON instance 1

```
python scripts/install-agent.py -s services/core/DataMover -c services/core/DataMover/
↪config -i datamover --start
```

- Add the public key of the DataMover historian on VOLTTRON instance 2 to enable authentication of the DataMover on VOLTTRON instance 2.
  - Get the public key of the DataMover. Run the below command on instance 1 terminal.

  ```
  vctl auth publickey --name datamoveragent-0.1
  ```

  - Add the credentials of the DataMover historian in VOLTTRON instance 2

  ```
  vctl auth add --credentials <public key of data mover>
  ```

Check data in SQLite database

To check if data is transferred and stored in the database of remote platform historian, we need to check the entries in the database. The default location of SQL database (if not explicitly specified in the config file) will be in the *data* directory inside the platform historian's installed directory within it's *$VOLTTRON_HOME*.

- Get the uuid of the platform historian. This can be found by running the `vctl status` on the terminal of instance 2. The first column of the data mover historian entry in the status table gives the first alphabet/number of the uuid.
- Go the *data* directory of platform historian's install directory. For example, */home/ubuntu/.platform2/agents/6292302c-32cf-4744-bd13-27e78e96184f/sqlhistorianagent-3.7.0/data*
- **Run the SQL command to see the data**

  ```
  sqlite3 platform.historian.sqlite
  select * from data;
  ```

- You will see similar entries

  ```
  2020-10-27T15:07:55.006549+00:00|14|true
  2020-10-27T15:07:55.006549+00:00|15|10.0
  2020-10-27T15:07:55.006549+00:00|16|20
  2020-10-27T15:07:55.006549+00:00|17|true
  2020-10-27T15:07:55.006549+00:00|18|10.0
  2020-10-27T15:07:55.006549+00:00|19|20
  2020-10-27T15:07:55.006549+00:00|20|true
  2020-10-27T15:07:55.006549+00:00|21|0
  2020-10-27T15:07:55.006549+00:00|22|0
  ```

Multi-Platform Between Routers

Multi-Platform between routers alleviates the need for an agent in one platform to connect to another platform directly in order for it to send/receive messages from the other platform. Instead with this new type of connection, connections to external platforms will be maintained by the platforms itself and agents do not have the burden to manage the connections directly. This guide will show how to connect three VOLTTRON instances with a fake driver running on

VOLTTRON instance 1 publishing to topic with prefix=”devices” and listener agents running on other 2 VOLTTRON instances subscribed to topic “devices”.

Getting Started

Modify the subscribe annotate method parameters in the listener agent (examples/ListenerAgent/listener/agent.py in the VOLTTRON root directory) to include `all_platforms=True` parameter to receive messages from external platforms.

```
@PubSub.subscribe('pubsub', '')
```

to

```
@PubSub.subscribe('pubsub', 'devices', all_platforms=True)
```

or add below line in the *onstart* method

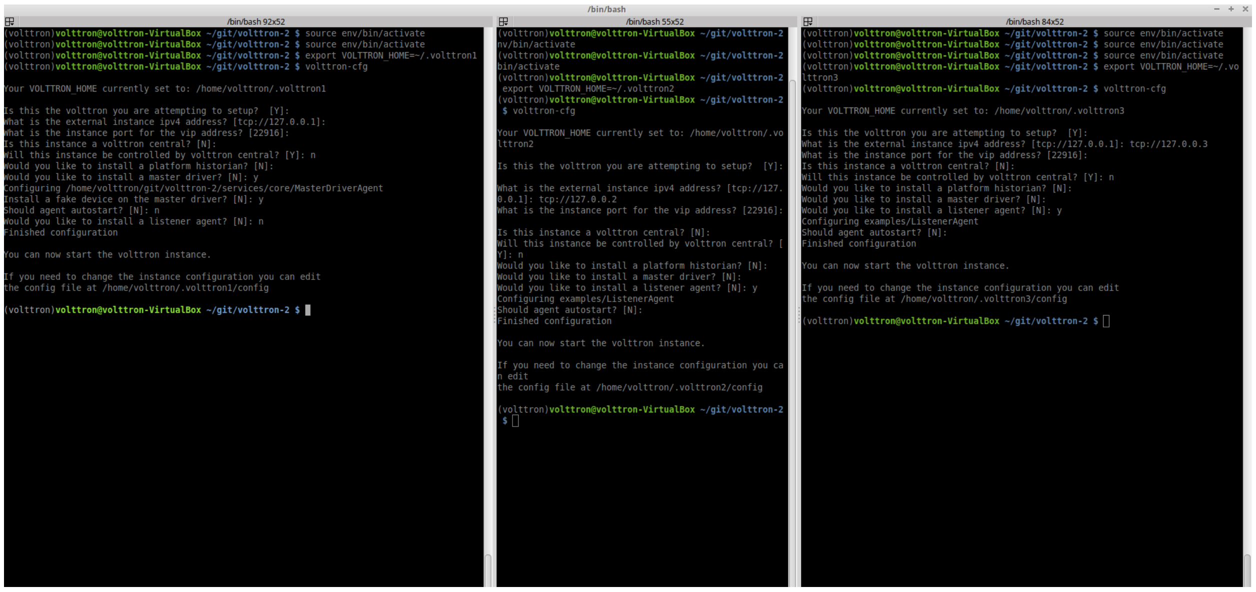
```
self.vip.pubsub.subscribe('pubsub', 'devices', self.on_match, all_platforms=True)
```

Note: If using the onstart method remove the @PubSub.subscribe(‘pubsub’, ‘’) from the top of the method.

After *installing VOLTTRON*, open three shells with the current directory the root of the VOLTTRON repository. Then activate the VOLTTRON environment and export the *VOLTTRON_HOME* variable. The home variable needs to be different for each instance.

```
$ source env/bin/activate
$ export VOLTTRON_HOME=~/.volttron1
```

Run *vcfg* in all the three shells. This command will ask how the instance should be set up. Many of the options have defaults and that will be sufficient. Enter a different VIP address for each platform. Configure fake master driver in the first shell and listener agent in second and third shell.



```
(volttron)volttron@volttron-VirtualBox ~/git/volttron-2 $ source env/bin/activate
(volttron)volttron@volttron-VirtualBox ~/git/volttron-2 $ source env/bin/activate
(volttron)volttron@volttron-VirtualBox ~/git/volttron-2 $ export VOLTTRON_HOME=~/.volttron1
(volttron)volttron@volttron-VirtualBox ~/git/volttron-2 $ volttron-cfg

Your VOLTTRON_HOME currently set to: /home/volttron/.volttron1

Is this the volttron you are attempting to setup?  [Y]:
What is the external instance ipv4 address? [tcp://127.0.0.1]:
What is the instance port for the vip address? [22916]:
Is this instance a volttron central? [N]:
Will this instance be controlled by volttron central? [Y]: n
Would you like to install a platform historian? [N]:
Would you like to install a master driver? [N]: y
Configuring /home/volttron/git/volttron-2/services/core/MasterDriverAgent
Install a fake device on the master driver? [N]: y
Should agent autostart? [N]: n
Would you like to install a listener agent? [N]: n
Finished configuration

You can now start the volttron instance.

If you need to change the instance configuration you can edit
the config file at /home/volttron/.volttron1/config

(volttron)volttron@volttron-VirtualBox ~/git/volttron-2 $
```

```
(volttron)volttron@volttron-VirtualBox ~/git/volttron-2
nv/bin/activate
(volttron)volttron@volttron-VirtualBox ~/git/volttron-2
bin/activate
(volttron)volttron@volttron-VirtualBox ~/git/volttron-2
 export VOLTTRON_HOME=~/.volttron2
(volttron)volttron@volttron-VirtualBox ~/git/volttron-2
 $ volttron-cfg

Your VOLTTRON_HOME currently set to: /home/volttron/.vo
lttron2

Is this the volttron you are attempting to setup?  [Y]:

What is the external instance ipv4 address? [tcp://127.
0.0.1]: tcp://127.0.0.2
What is the instance port for the vip address? [22916]:

Is this instance a volttron central? [N]:
Will this instance be controlled by volttron central? [
Y]: n
Would you like to install a platform historian? [N]:
Would you like to install a master driver? [N]:
Would you like to install a listener agent? [N]: y
Configuring examples/ListenerAgent
Should agent autostart? [N]:
Finished configuration

You can now start the volttron instance.

If you need to change the instance configuration you ca
n edit
the config file at /home/volttron/.volttron2/config

(volttron)volttron@volttron-VirtualBox ~/git/volttron-2
 $
```

```
(volttron)volttron@volttron-VirtualBox ~/git/volttron-2 $ source env/bin/activate
(volttron)volttron@volttron-VirtualBox ~/git/volttron-2 $ source env/bin/activate
(volttron)volttron@volttron-VirtualBox ~/git/volttron-2 $ source env/bin/activate
(volttron)volttron@volttron-VirtualBox ~/git/volttron-2 $ export VOLTTRON_HOME=~/.vo
lttron3
(volttron)volttron@volttron-VirtualBox ~/git/volttron-2 $ volttron-cfg

Your VOLTTRON_HOME currently set to: /home/volttron/.volttron3

Is this the volttron you are attempting to setup?  [Y]:
What is the external instance ipv4 address? [tcp://127.0.0.1]: tcp://127.0.0.3
What is the instance port for the vip address? [22916]:
Is this instance a volttron central? [N]:
Will this instance be controlled by volttron central? [Y]: n
Would you like to install a platform historian? [N]:
Would you like to install a master driver? [N]:
Would you like to install a listener agent? [N]: y
Configuring examples/ListenerAgent
Should agent autostart? [N]:
Finished configuration

You can now start the volttron instance.

If you need to change the instance configuration you can edit
the config file at /home/volttron/.volttron3/config

(volttron)volttron@volttron-VirtualBox ~/git/volttron-2 $
```

Multi-Platform Configuration

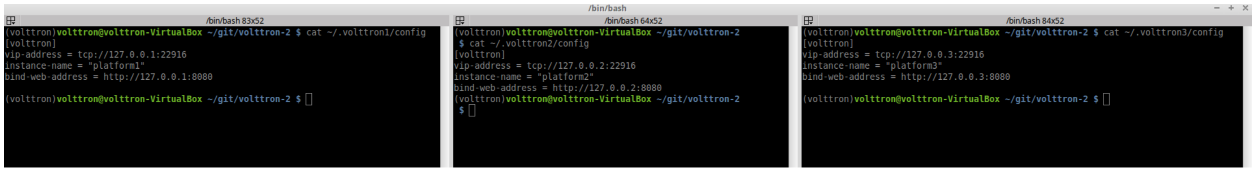
For each instance, specify the instance name in platform config file under it's `VOLTTRON_HOME` directory. If the platform supports web server, add the `bind-web-address` as well.

Here is an example,

Path of the config: *$VOLTTRON_HOME/config*

```
[volttron]
vip-address = tcp://127.0.0.1:22916
instance-name = "platform1"
bind-web-address = http://127.0.0.1:8080
```

Instance name and bind web address entries added into each VOLTTRON platform's config file is shown below.



```
(volttron)volttron@volttron-VirtualBox ~/git/volttron-2 $ cat ~/.volttron1/config
[volttron]
vip-address = tcp://127.0.0.1:22916
instance-name = "platform1"
bind-web-address = http://127.0.0.1:8080

(volttron)volttron@volttron-VirtualBox ~/git/volttron-2 $
```

```
(volttron)volttron@volttron-VirtualBox ~/git/volttron-2
 $ cat ~/.volttron2/config
[volttron]
vip-address = tcp://127.0.0.2:22916
instance-name = "platform2"
bind-web-address = http://127.0.0.2:8080
(volttron)volttron@volttron-VirtualBox ~/git/volttron-2
 $
```

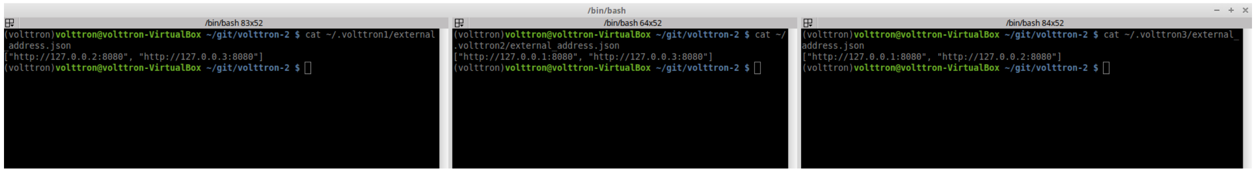
```
(volttron)volttron@volttron-VirtualBox ~/git/volttron-2 $ cat ~/.volttron3/config
[volttron]
vip-address = tcp://127.0.0.3:22916
instance-name = "platform3"
bind-web-address = http://127.0.0.3:8080

(volttron)volttron@volttron-VirtualBox ~/git/volttron-2 $
```

Next, each instance needs to know the VIP address, platform name and server keys of the remote platforms that it is connecting to. In addition, each platform has to authenticate or accept the connecting instances' public keys. We can do this step either by running VOLTTRON in setup mode or configure the information manually.

Configuration and Authentication in Setup Mode

Note: It is necessary for **each** platform to have a web server if running in setup mode

Add list of web addresses of remote platforms in `$VOLTTRON_HOME/external_address.json`



```
(volttron)volttron@volttron-VirtualBox ~/git/volttron-2 $ cat ~/.volttron1/external
_address.json
["http://127.0.0.2:8080", "http://127.0.0.3:8080"]
(volttron)volttron@volttron-VirtualBox ~/git/volttron-2 $
```

```
(volttron)volttron@volttron-VirtualBox ~/git/volttron-2 $ cat ~/
.volttron2/external_address.json
["http://127.0.0.1:8080", "http://127.0.0.3:8080"]
(volttron)volttron@volttron-VirtualBox ~/git/volttron-2 $
```

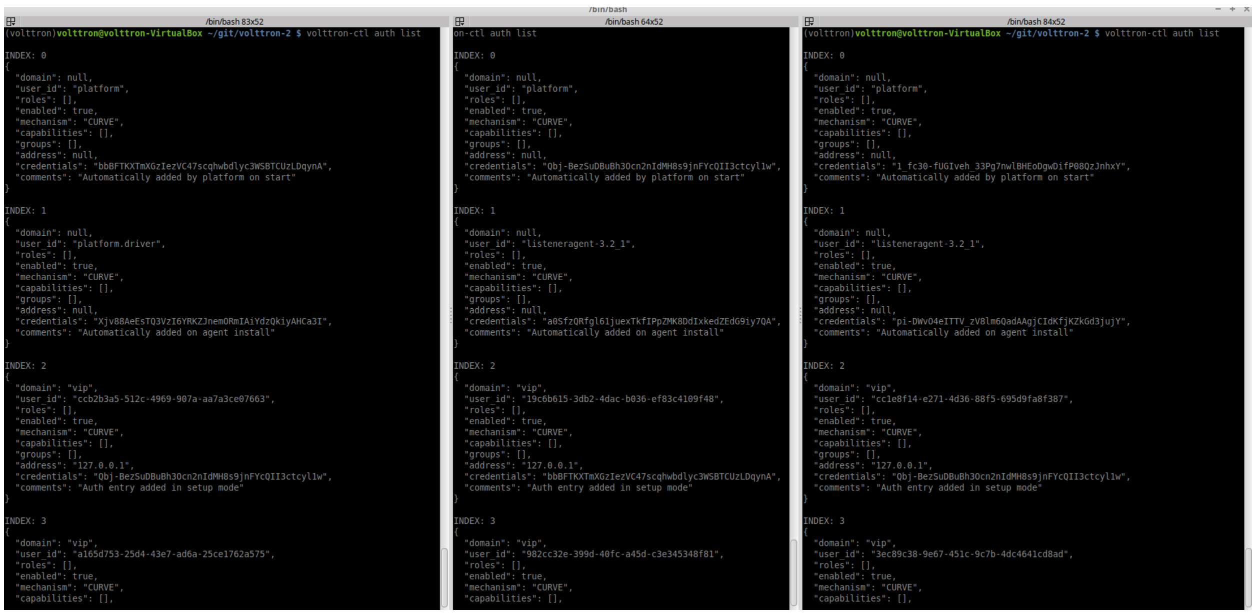
```
(volttron)volttron@volttron-VirtualBox ~/git/volttron-2 $ cat ~/.volttron3/external_
address.json
["http://127.0.0.1:8080", "http://127.0.0.2:8080"]
(volttron)volttron@volttron-VirtualBox ~/git/volttron-2 $
```

Start VOLTTRON instances in setup mode in the three terminal windows. The "-l" option in the following command tells VOLTTRON to log to a file. The file name should be different for each instance.

```
$ ./start-volttron --setup-mode
```

A new auth entry is added for each new platform connection. This can be checked with below command in each terminal window.

```
$ vctl auth list
```



```
(volttron)volttron@volttron-VirtualBox ~/git/volttron-2 $ volttron-ctl auth list

INDEX: 0
{
  "domain": null,
  "user_id": "platform",
  "roles": [],
  "enabled": true,
  "mechanism": "CURVE",
  "capabilities": [],
  "groups": [],
  "address": null,
  "credentials": "bbBFTKXTmXGzIezVC47scqhwbdlyc3WSBTCUzLDqynA",
  "comments": "Automatically added by platform on start"
}

INDEX: 1
{
  "domain": null,
  "user_id": "platform.driver",
  "roles": [],
  "enabled": true,
  "mechanism": "CURVE",
  "capabilities": [],
  "groups": [],
  "address": null,
  "credentials": "Xjv88AeEsTQ3VzI6YRKZJnemORmIAiYdzQkiyAHCa3I",
  "comments": "Automatically added on agent install"
}

INDEX: 2
{
  "domain": "vip",
  "user_id": "ccb2b3a5-512c-4969-907a-aa7a3ce07663",
  "roles": [],
  "enabled": true,
  "mechanism": "CURVE",
  "capabilities": [],
  "groups": [],
  "address": "127.0.0.1",
  "credentials": "Qbj-BezSuDBuBh3Ocn2nIdMH8s9jnFYcQII3ctcyl1w",
  "comments": "Auth entry added in setup mode"
}

INDEX: 3
{
  "domain": "vip",
  "user_id": "a165d753-25d4-43e7-ad6a-25ce1762a575",
  "roles": [],
  "enabled": true,
  "mechanism": "CURVE",
  "capabilities": [],
```

```
on-ctl auth list

INDEX: 0
{
  "domain": null,
  "user_id": "platform",
  "roles": [],
  "enabled": true,
  "mechanism": "CURVE",
  "capabilities": [],
  "groups": [],
  "address": null,
  "credentials": "Qbj-BezSuDBuBh3Ocn2nIdMH8s9jnFYcQII3ctcyl1w",
  "comments": "Automatically added by platform on start"
}

INDEX: 1
{
  "domain": null,
  "user_id": "listeneragent-3.2_1",
  "roles": [],
  "enabled": true,
  "mechanism": "CURVE",
  "capabilities": [],
  "groups": [],
  "address": null,
  "credentials": "a0SfzQRfgl61juexTkfIPpZMK8DdIxkedZEdG9iy7QA",
  "comments": "Automatically added on agent install"
}

INDEX: 2
{
  "domain": "vip",
  "user_id": "19c6b615-3db2-4dac-b036-ef83c4109f48",
  "roles": [],
  "enabled": true,
  "mechanism": "CURVE",
  "capabilities": [],
  "groups": [],
  "address": "127.0.0.1",
  "credentials": "bbBFTKXTmXGzIezVC47scqhwbdlyc3WSBTCUzLDqynA",
  "comments": "Auth entry added in setup mode"
}

INDEX: 3
{
  "domain": "vip",
  "user_id": "982cc32e-399d-40fc-a45d-c3e345348f81",
  "roles": [],
  "enabled": true,
  "mechanism": "CURVE",
  "capabilities": [],
```

```
(volttron)volttron@volttron-VirtualBox ~/git/volttron-2 $ volttron-ctl auth list

INDEX: 0
{
  "domain": null,
  "user_id": "platform",
  "roles": [],
  "enabled": true,
  "mechanism": "CURVE",
  "capabilities": [],
  "groups": [],
  "address": null,
  "credentials": "1_fc30-fUGIveh_33Pg7nwlBHEoDgwDifP08QzJnhxY",
  "comments": "Automatically added by platform on start"
}

INDEX: 1
{
  "domain": null,
  "user_id": "listeneragent-3.2_1",
  "roles": [],
  "enabled": true,
  "mechanism": "CURVE",
  "capabilities": [],
  "groups": [],
  "address": null,
  "credentials": "pi-DWvO4eITTV_zV8lm6QadAAgjCIdKfjKZkGd3jujY",
  "comments": "Automatically added on agent install"
}

INDEX: 2
{
  "domain": "vip",
  "user_id": "cc1e8f14-e271-4d36-88f5-695d9fa8f387",
  "roles": [],
  "enabled": true,
  "mechanism": "CURVE",
  "capabilities": [],
  "groups": [],
  "address": "127.0.0.1",
  "credentials": "Qbj-BezSuDBuBh3Ocn2nIdMH8s9jnFYcQII3ctcyl1w",
  "comments": "Auth entry added in setup mode"
}

INDEX: 3
{
  "domain": "vip",
  "user_id": "3ec89c38-9e67-451c-9c7b-4dc4641cd8ad",
  "roles": [],
  "enabled": true,
  "mechanism": "CURVE",
  "capabilities": [],
```

After all the connections are authenticated, we can start the instances in normal mode.

```
$ ./stop-volttron
$ ./start-volttron
```

Setup Configuration and Authentication Manually

If you do not need web servers in your setup, then you will need to build the platform discovery config file manually. The config file should contain an entry containing VIP address, instance name and serverkey of each remote platform connection.

Name of the file: *external_platform_discovery.json*

Directory path: Each platform's VOLTTRON_HOME directory.

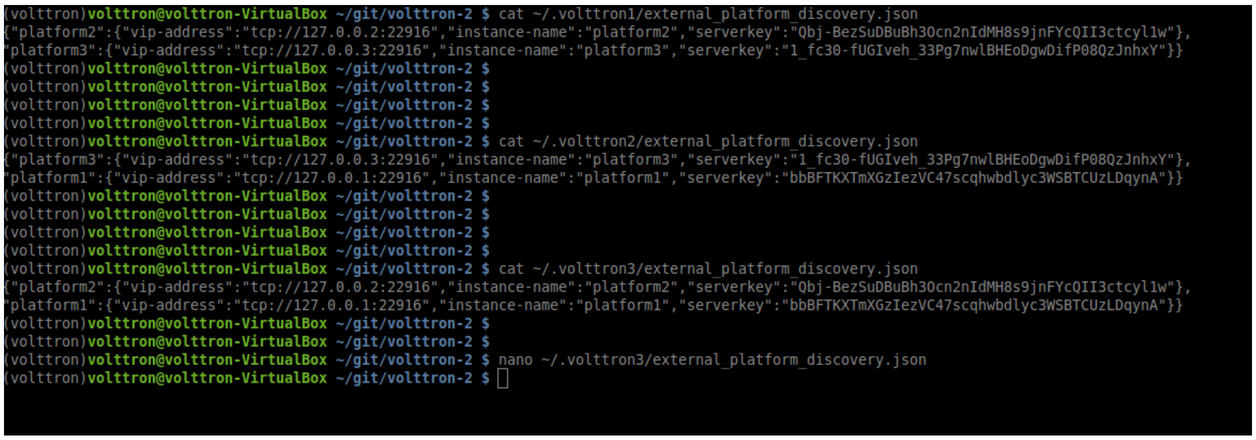
For example, since VOLTTRON instance 1 is connecting to VOLTTRON instance 2 and 3, contents of `external_platform_discovery.json` will be

```
{
    "platform2": {"vip-address":"tcp://127.0.0.2:22916",
                  "instance-name":"platform2",
                  "serverkey":"YFyIgXy2H7gIKC1x6uPMdDOB_i9lzfAPB1IgbxfXLGc"},
    "platform3": {"vip-address":"tcp://127.0.0.3:22916",
                  "instance-name":"platform3",
                  "serverkey":"hzU2bnlacAhZSaI0rI8a6XK_bqLSpA0JRK4jq8ttZxw"}
}
```

We can obtain the serverkey of each platform using below command in each terminal window:

```
$ vctl auth serverkey
```

Contents of `external_platform_discovery.json` of VOLTTRON instance 1, 2, 3 is shown below.



```
(volttron)volttron@volttron-VirtualBox ~/git/volttron-2 $ cat ~/.volttron1/external_platform_discovery.json
{"platform2":{"vip-address":"tcp://127.0.0.2:22916","instance-name":"platform2","serverkey":"Qbj-BezSuDBuBh3Ocn2nIdMH8s9jnFYcQII3ctcyl1w"},
"platform3":{"vip-address":"tcp://127.0.0.3:22916","instance-name":"platform3","serverkey":"1_fc30-fUGIveh_33Pg7nwlBHEoDgwDifP08QzJnhxY"}}
(volttron)volttron@volttron-VirtualBox ~/git/volttron-2 $
(volttron)volttron@volttron-VirtualBox ~/git/volttron-2 $
(volttron)volttron@volttron-VirtualBox ~/git/volttron-2 $
(volttron)volttron@volttron-VirtualBox ~/git/volttron-2 $
(volttron)volttron@volttron-VirtualBox ~/git/volttron-2 $ cat ~/.volttron2/external_platform_discovery.json
{"platform3":{"vip-address":"tcp://127.0.0.3:22916","instance-name":"platform3","serverkey":"1_fc30-fUGIveh_33Pg7nwlBHEoDgwDifP08QzJnhxY"},
"platform1":{"vip-address":"tcp://127.0.0.1:22916","instance-name":"platform1","serverkey":"bbBFTKXTmXGzIezVC47scqhwbdlyc3WSBTCUzLDqynA"}}
(volttron)volttron@volttron-VirtualBox ~/git/volttron-2 $
(volttron)volttron@volttron-VirtualBox ~/git/volttron-2 $
(volttron)volttron@volttron-VirtualBox ~/git/volttron-2 $
(volttron)volttron@volttron-VirtualBox ~/git/volttron-2 $
(volttron)volttron@volttron-VirtualBox ~/git/volttron-2 $ cat ~/.volttron3/external_platform_discovery.json
{"platform2":{"vip-address":"tcp://127.0.0.2:22916","instance-name":"platform2","serverkey":"Qbj-BezSuDBuBh3Ocn2nIdMH8s9jnFYcQII3ctcyl1w"},
"platform1":{"vip-address":"tcp://127.0.0.1:22916","instance-name":"platform1","serverkey":"bbBFTKXTmXGzIezVC47scqhwbdlyc3WSBTCUzLDqynA"}}
(volttron)volttron@volttron-VirtualBox ~/git/volttron-2 $
(volttron)volttron@volttron-VirtualBox ~/git/volttron-2 $
(volttron)volttron@volttron-VirtualBox ~/git/volttron-2 $ nano ~/.volttron3/external_platform_discovery.json
(volttron)volttron@volttron-VirtualBox ~/git/volttron-2 $
```

After this, you will need to add the server keys of the connecting platforms using the `vctl` utility. Type **vctl auth add** command on the command prompt and simply hit Enter to select defaults on all fields except **credentials**. Here, we can either add serverkey of connecting platform or type */.*/* to allow ALL connections.

Warning: */.*/* allows ALL agent and platform connections without authentication.

```
$ vctl auth add
domain []:
address []:
user_id []:
capabilities (delimit multiple entries with comma) []:
roles (delimit multiple entries with comma) []:
groups (delimit multiple entries with comma) []:
mechanism [CURVE]:
credentials []: /.*/
comments []:
enabled [True]:
added entry domain=None, address=None, mechanism='CURVE', credentials=u'/.*/', user_
↪id=None
```

For more information on authentication see *authentication*.

Once the initial configuration are setup, you can start all the VOLTTRON instances in normal mode.

```
$ ./start-volttron
```

Next step is to start agents in each platform to observe the multi-platform PubSub communication behavior.

Start Master driver on VOLTTRON instance 1

If master driver is not configured to auto start when the instance starts up, we can start it explicitly with this command.

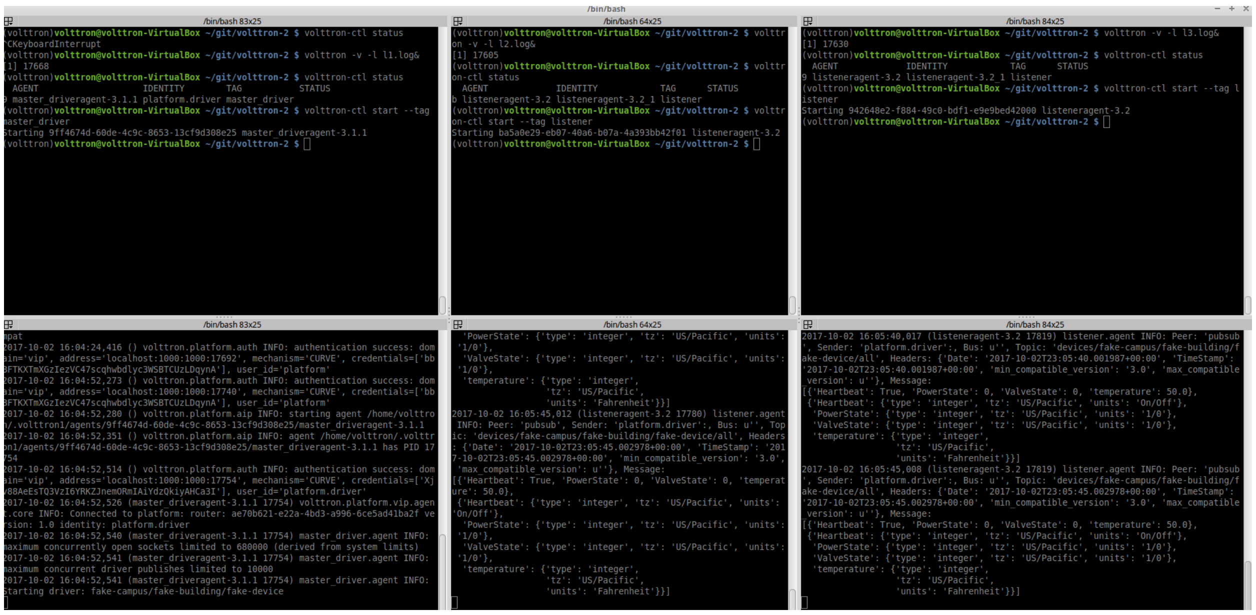
```
$ vctl start --tag master_driver
```

Start Listener agents on VOLTTRON instance 2 and 3

If the listener agent is not configured to auto start when the instance starts up, we can start it explicitly with this command.

```
$ vctl start --tag listener
```

We should start seeing messages with prefix="devices" in the logs of VOLTTRON instances 2 and 3.



```
(volttron)volttron@volttron-VirtualBox ~/git/volttron-2 $ volttron-ctl status
^CKeyboardInterrupt
(volttron)volttron@volttron-VirtualBox ~/git/volttron-2 $ volttron -v -l l1.log&
[1] 17668
(volttron)volttron@volttron-VirtualBox ~/git/volttron-2 $ volttron-ctl status
  AGENT                   IDENTITY        TAG           STATUS
9 master_driveragent-3.1.1 platform.driver master_driver
(volttron)volttron@volttron-VirtualBox ~/git/volttron-2 $ volttron-ctl start --tag
master_driver
Starting 9ff4674d-60de-4c9c-8653-13cf9d308e25 master_driveragent-3.1.1
(volttron)volttron@volttron-VirtualBox ~/git/volttron-2 $ 
```

```
(volttron)volttron@volttron-VirtualBox ~/git/volttron-2 $ volttr
on -v -l l2.log&
[1] 17605
(volttron)volttron@volttron-VirtualBox ~/git/volttron-2 $ volttr
on-ctl status
  AGENT             IDENTITY             TAG      STATUS
b listeneragent-3.2 listeneragent-3.2_1 listener
(volttron)volttron@volttron-VirtualBox ~/git/volttron-2 $ volttr
on-ctl start --tag listener
Starting ba5a0e29-eb07-40a6-b07a-4a393bb42f01 listeneragent-3.2
(volttron)volttron@volttron-VirtualBox ~/git/volttron-2 $ 
```

```
(volttron)volttron@volttron-VirtualBox ~/git/volttron-2 $ volttron -v -l l3.log&
[1] 17630
(volttron)volttron@volttron-VirtualBox ~/git/volttron-2 $ volttron-ctl status
  AGENT             IDENTITY             TAG      STATUS
9 listeneragent-3.2 listeneragent-3.2_1 listener
(volttron)volttron@volttron-VirtualBox ~/git/volttron-2 $ volttron-ctl start --tag l
istener
Starting 942648e2-f884-49c0-bdf1-e9e9bed42000 listeneragent-3.2
(volttron)volttron@volttron-VirtualBox ~/git/volttron-2 $ 
```

```
mpat
2017-10-02 16:04:24,416 () volttron.platform.auth INFO: authentication success: dom
ain='vip', address='localhost:1000:1000:17692', mechanism='CURVE', credentials=['bb
3FTKXTmXGzIezVC47scqhwbdlyc3WSBTCUzLDqynA'], user_id='platform'
2017-10-02 16:04:52,273 () volttron.platform.auth INFO: authentication success: dom
ain='vip', address='localhost:1000:1000:17740', mechanism='CURVE', credentials=['bb
3FTKXTmXGzIezVC47scqhwbdlyc3WSBTCUzLDqynA'], user_id='platform'
2017-10-02 16:04:52,280 () volttron.platform.aip INFO: starting agent /home/volttro
n/.volttron1/agents/9ff4674d-60de-4c9c-8653-13cf9d308e25/master_driveragent-3.1.1
2017-10-02 16:04:52,351 () volttron.platform.aip INFO: agent /home/volttron/.voltt
ron1/agents/9ff4674d-60de-4c9c-8653-13cf9d308e25/master_driveragent-3.1.1 has PID 17
754
2017-10-02 16:04:52,514 () volttron.platform.auth INFO: authentication success: dom
ain='vip', address='localhost:1000:1000:17754', mechanism='CURVE', credentials=['Xj
v88AeEsTQ3VzI6YRKZJnemORmIAiYdzQkiyAHCa3I'], user_id='platform.driver'
2017-10-02 16:04:52,526 (master_driveragent-3.1.1 17754) volttron.platform.vip.agen
t.core INFO: Connected to platform: router: ae70b621-e22a-4bd3-a996-6ce5ad41ba2f ve
rsion: 1.0 identity: platform.driver
2017-10-02 16:04:52,540 (master_driveragent-3.1.1 17754) master_driver.agent INFO:
maximum concurrently open sockets limited to 680000 (derived from system limits)
2017-10-02 16:04:52,541 (master_driveragent-3.1.1 17754) master_driver.agent INFO:
maximum concurrent driver publishes limited to 10000
2017-10-02 16:04:52,541 (master_driveragent-3.1.1 17754) master_driver.agent INFO:
Starting driver: fake-campus/fake-building/fake-device
```

```
  'PowerState': {'type': 'integer', 'tz': 'US/Pacific', 'units':
 '1/0'},
  'ValveState': {'type': 'integer', 'tz': 'US/Pacific', 'units':
 '1/0'},
  'temperature': {'type': 'integer',
                  'tz': 'US/Pacific',
                  'units': 'Fahrenheit'}}]
2017-10-02 16:05:45,012 (listeneragent-3.2 17780) listener.agent
 INFO: Peer: 'pubsub', Sender: 'platform.driver':, Bus: u'', Top
ic: 'devices/fake-campus/fake-building/fake-device/all', Headers
: {'Date': '2017-10-02T23:05:45.002978+00:00', 'TimeStamp': '201
7-10-02T23:05:45.002978+00:00', 'min_compatible_version': '3.0',
 'max_compatible_version': u''}, Message:
[{'Heartbeat': True, 'PowerState': 0, 'ValveState': 0, 'temperat
ure': 50.0},
 {'Heartbeat': {'type': 'integer', 'tz': 'US/Pacific', 'units':
'On/Off'},
  'PowerState': {'type': 'integer', 'tz': 'US/Pacific', 'units':
 '1/0'},
  'ValveState': {'type': 'integer', 'tz': 'US/Pacific', 'units':
 '1/0'},
  'temperature': {'type': 'integer',
                  'tz': 'US/Pacific',
                  'units': 'Fahrenheit'}}]
```

```
2017-10-02 16:05:40,017 (listeneragent-3.2 17819) listener.agent INFO: Peer: 'pubsub
', Sender: 'platform.driver':, Bus: u'', Topic: 'devices/fake-campus/fake-building/f
ake-device/all', Headers: {'Date': '2017-10-02T23:05:40.001987+00:00', 'TimeStamp':
'2017-10-02T23:05:40.001987+00:00', 'min_compatible_version': '3.0', 'max_compatible
_version': u''}, Message:
[{'Heartbeat': True, 'PowerState': 0, 'ValveState': 0, 'temperature': 50.0},
 {'Heartbeat': {'type': 'integer', 'tz': 'US/Pacific', 'units': 'On/Off'},
  'PowerState': {'type': 'integer', 'tz': 'US/Pacific', 'units': '1/0'},
  'ValveState': {'type': 'integer', 'tz': 'US/Pacific', 'units': '1/0'},
  'temperature': {'type': 'integer',
                  'tz': 'US/Pacific',
                  'units': 'Fahrenheit'}}]
2017-10-02 16:05:45,008 (listeneragent-3.2 17819) listener.agent INFO: Peer: 'pubsub
', Sender: 'platform.driver':, Bus: u'', Topic: 'devices/fake-campus/fake-building/f
ake-device/all', Headers: {'Date': '2017-10-02T23:05:45.002978+00:00', 'TimeStamp':
'2017-10-02T23:05:45.002978+00:00', 'min_compatible_version': '3.0', 'max_compatible
_version': u''}, Message:
[{'Heartbeat': True, 'PowerState': 0, 'ValveState': 0, 'temperature': 50.0},
 {'Heartbeat': {'type': 'integer', 'tz': 'US/Pacific', 'units': 'On/Off'},
  'PowerState': {'type': 'integer', 'tz': 'US/Pacific', 'units': '1/0'},
  'ValveState': {'type': 'integer', 'tz': 'US/Pacific', 'units': '1/0'},
  'temperature': {'type': 'integer',
                  'tz': 'US/Pacific',
                  'units': 'Fahrenheit'}}]
```

Stopping All the Platforms

We can stop all the VOLTTRON instances by executing below command in each terminal window.

```
$ vctl shutdown --platform
```

Platform External Address Configuration

In the configuration file located in *$VOLTTRON_HOME/config* add `vip-address=tcp://ip:port` for each address you want to listen on:

```
Example
vip-address=tcp://127.0.0.102:8182
vip-address=tcp://127.0.0.103:8083
vip-address=tcp://127.0.0.103:8183
```

Note: The config file is generated after running the *vcfg* command. The VIP-address is for the local platform, NOT the remote platform.

Multi-platform RabbitMQ Deployment

With ZeroMQ based VOLTTRON, multi-platform communication was accomplished in three different ways:

1. Direct connection to remote instance - Write an agent that would connect to a remote instance directly.
2. Special agents - Use special agents such as forward historian/data puller agents that would forward/receive messages to/from remote instances. In RabbitMQ-VOLTTRON, we make use of the *shovel* plugin to achieve this behavior. Please refer to *Shovel Plugin* to get an overview of shovels.
3. Multi-Platform RPC and PubSub - Configure VIP address of all remote instances that an instance has to connect to in it's *$VOLTTRON_HOME/external_discovery.json* and let the router module in each instance manage the connection and take care of the message routing for us. In RabbitMQ-VOLTTRON, we make use of the *federation* plugin to achieve this behavior. Please refer to *Federation Plugin* get an overview of federation.

Using the Federation Plugin

We can connect multiple VOLTTRON instances using the federation plugin. Before setting up federation links, we need to first identify the upstream server and downstream server. The upstream server is the node that is publishing some message of interest and downstream server is the node that wants to receive messages from the upstream server. A federation link needs to be established from the downstream VOLTTRON instance to the upstream VOLTTRON instance. To setup a federation link, we will need to add upstream server information in a RabbitMQ federation configuration file:

Path: *$VOLTTRON_HOME/rabbitmq_federation_config.yml*

```
# Mandatory parameters for federation setup
federation-upstream:
  rabbit-4:
    port: '5671'
    virtual-host: volttron4
  rabbit-5:
    port: '5671'
    virtual-host: volttron5
```

To configure the VOLTTRON instance to setup federation, run the following command:

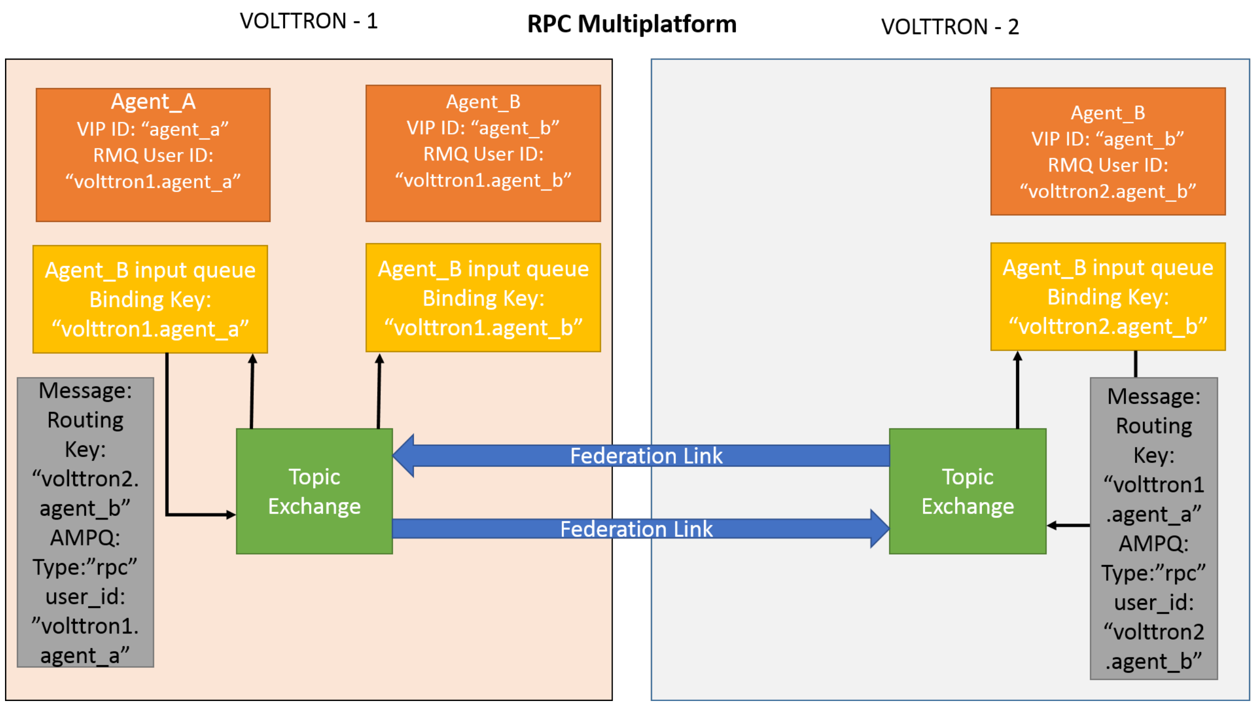
```
vcfg --rabbitmq federation [optional path to rabbitmq_federation_config.yml]
```

This will setup federation links to upstream servers and sets policy to make the VOLTTRON exchange *federated*. Once a federation link is established to remote instance, the messages published on the remote instance become available to local instance as if it were published on the local instance.

For detailed instructions to setup federation, please refer to the *platform installation docs*.

Multi-Platform RPC With Federation

For multi-platform RPC communication, federation links need to be established on both the VOLTTRON nodes. Once the federation links are established, RPC communication becomes fairly simple.


VOLTTRON - 1
RPC Multiplatform
VOLTTRON - 2
Agent_A VIP ID: "agent_a" RMQ User ID: "volttron1.agent_a"
Agent_B VIP ID: "agent_b" RMQ User ID: "volttron1.agent_b"
Agent_B VIP ID: "agent_b" RMQ User ID: "volttron2.agent_b"
Agent_B input queue Binding Key: "volttron1.agent_a"
Agent_B input queue Binding Key: "volttron1.agent_b"
Agent_B input queue Binding Key: "volttron2.agent_b"
Message: Routing Key: "volttron2.agent_b" AMPQ: Type:"rpc" user_id: "volttron1.agent_a"
Topic Exchange
Federation Link
Federation Link
Topic Exchange
Message: Routing Key: "volttron1.agent_a" AMPQ: Type:"rpc" user_id: "volttron2.agent_b"

Consider Agent A on VOLTTRON instance "volttron1" on host "host_A" wants to make RPC call to Agent B on VOLTTRON instance "volttron2" on host "host_B".

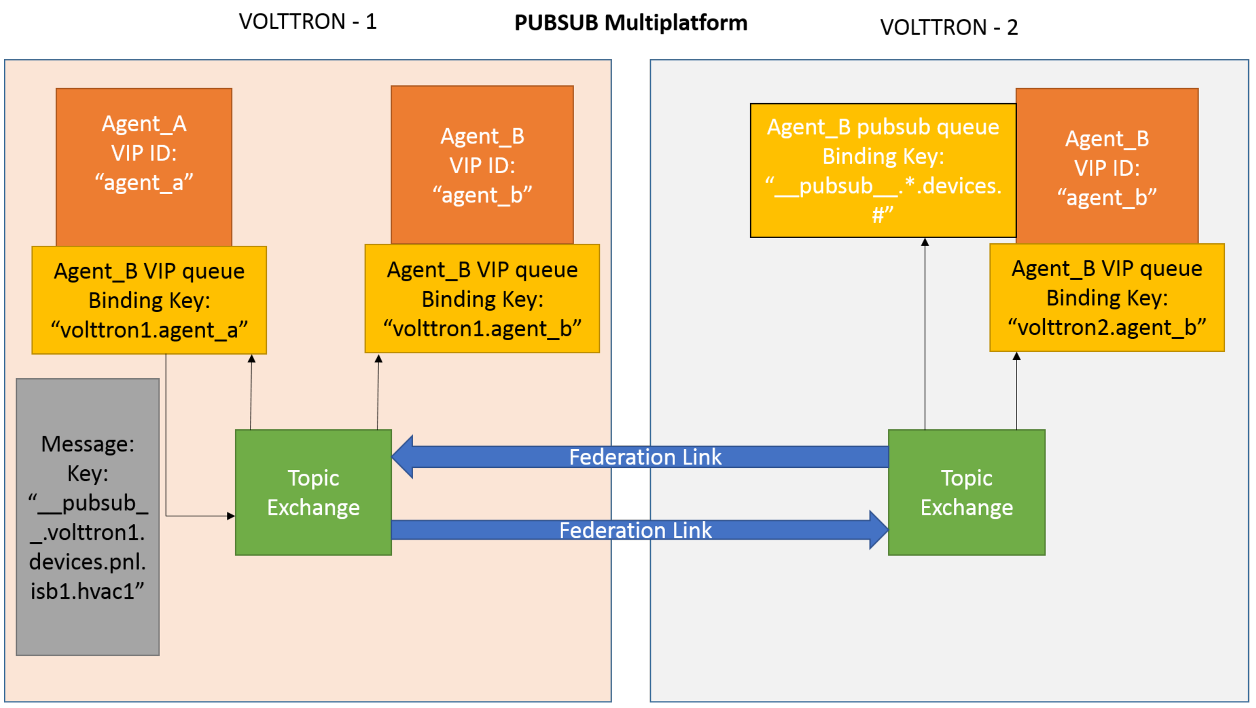
1. Agent A makes RPC call.

```
kwargs = {"external_platform": self.destination_instance_name}
agent_a.vip.rpc.call("agent_b", set_point, "point_name", 2.5, \**kwargs)
```

2. The message is transferred over federation link to VOLTTRON instance "volttron2" as both the exchanges are made *federated*.
3. The RPC subsystem of Agent B calls the actual RPC method and gets the result. It encapsulates the message result into a VIP message object and sends it back to Agent A on VOLTTRON instance "volttron1".
4. The RPC subsystem on Agent A receives the message result and gives it to the Agent A application.

Multi-Platform PubSub With Federation

For multi-platform PubSub communication, it is sufficient to have federation link from the downstream server to the upstream server. In case of bi-directional data flow, links have to established in both the directions.


VOLTTRON - 1
PUBSUB Multiplatform
VOLTTRON - 2
Agent_A VIP ID: "agent_a"
Agent_B VIP ID: "agent_b"
Agent_B pubsub queue Binding Key: "__pubsub__.*.devices.#"
Agent_B VIP ID: "agent_b"
Agent_B VIP queue Binding Key: "volttron1.agent_a"
Agent_B VIP queue Binding Key: "volttron1.agent_b"
Agent_B VIP queue Binding Key: "volttron2.agent_b"
Message: Key: "__pubsub__.volttron1.devices.pnl.isb1.hvac1"
Topic Exchange
Federation Link
Federation Link
Topic Exchange

Consider Agent B on VOLTTRON instance "volttron2" on host "host_B" which wants to subscribe to messages from VOLTTRON instance "volttron2" on host "host_B". First, a federation link needs to be established from "volttron2" to "volttron1".

1. Agent B makes a subscribe call:

```
agent_b.vip.subscribe.call("pubsub", prefix="devices", all_platforms=True)
```

2. The PubSub subsystem converts the prefix to `__pubsub__.*.devices.#`. Here, "*" indicates that agent is subscribing to the "devices" topic from all VOLTTRON platforms.
3. A new queue is created and bound to VOLTTRON exchange with the above binding key. Since the VOLTTRON exchange is a *federated exchange*, any subscribed message on the upstream server becomes available on the federated exchange and Agent B will be able to receive it.
4. Agent A publishes message to topic *devices/pnnl/isb1/hvac1*
5. The PubSub subsystem publishes this message on it's VOLTTRON exchange.
6. Due to the federation link, message is received by the Pubsub subsystem of Agent A.

Using the Shovel Plugin

Shovels act as well written client applications which move messages from a source to a destination broker. The below configuration shows how to setup a shovel to forward PubSub messages or perform multi-platform RPC communication from local to a remote instance. It expects *hostname*, *port* and *virtual host* configuration values for the remote instance.

Path: *$VOLTTRON_HOME/rabbitmq_shovel_config.yml*

```
# Mandatory parameters for shovel setup
shovel:
  rabbit-2:
```

(continues on next page)

(continued from previous page)

```
    port: '5671'
    virtual-host: volttron
    # Configuration to forward pubsub topics
    pubsub:
      # Identity of agent that is publishing the topic
      platform.driver:
        - devices
    # Configuration to make remote RPC calls
    rpc:
      # Remote instance name
      volttron2:
        # List of pair of agent identities (local caller, remote callee)
        - [scheduler, platform.actuator]
```

To forward PubSub messages, the topic and agent identity of the publisher agent is needed. To perform RPC, the instance name of the remote instance and agent identities of the local agent and remote agent are needed.

To configure the VOLTTRON instance to setup shovel, run the following command.

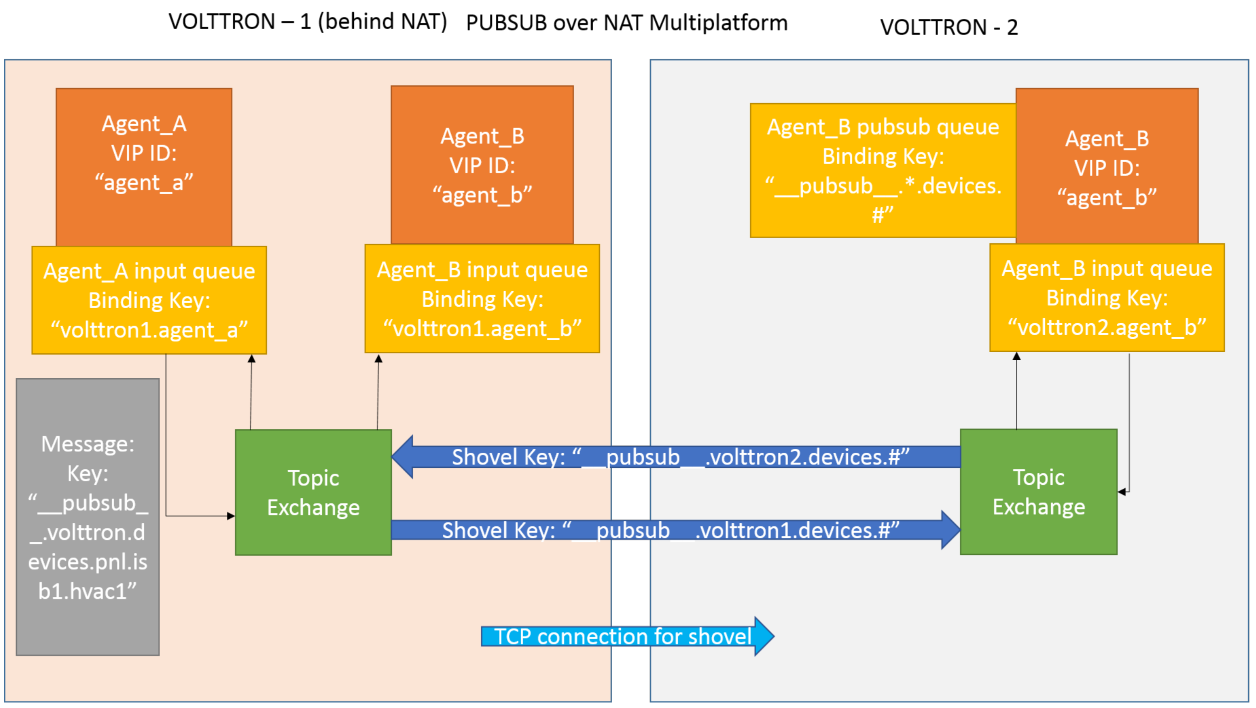
```
vcfg --rabbitmq shovel [optional path to rabbitmq_shovel_config.yml]
```

This setups up a shovel that forwards messages (either PubSub or RPC) from local exchange to remote exchange.

Multi-Platform PubSub With Shovel

After the shovel link is established for Pubsub, the below figure shows how the communication happens.

Note: For bi-directional pubsub communication, shovel links need to be created on both the nodes. The “blue” arrows show the shovel binding key. The pubsub topic configuration in *$VOLTTRON_HOME/rabbitmq_shovel_config.yml* gets internally converted to the shovel binding key: *“__pubsub__.<local instance name>.<actual topic>”*.


VOLTTRON – 1 (behind NAT)
PUBSUB over NAT Multiplatform
VOLTTRON - 2
Agent_A VIP ID: "agent_a"
Agent_B VIP ID: "agent_b"
Agent_A input queue Binding Key: "volttron1.agent_a"
Agent_B input queue Binding Key: "volttron1.agent_b"
Message: Key: "__pubsub__.volttron.devices.pnl.isb1.hvac1"
Topic Exchange
Shovel Key: "__pubsub__.volttron2.devices.#"
Shovel Key: "__pubsub__.volttron1.devices.#"
TCP connection for shovel
Agent_B pubsub queue Binding Key: "__pubsub__.*.devices.#"
Agent_B VIP ID: "agent_b"
Agent_B input queue Binding Key: "volttron2.agent_b"
Topic Exchange

Now consider a case where shovels are setup in both the directions for forwarding "devices" topic.

1. Agent B makes a subscribe call to receive messages with topic "devices" from all connected platforms.

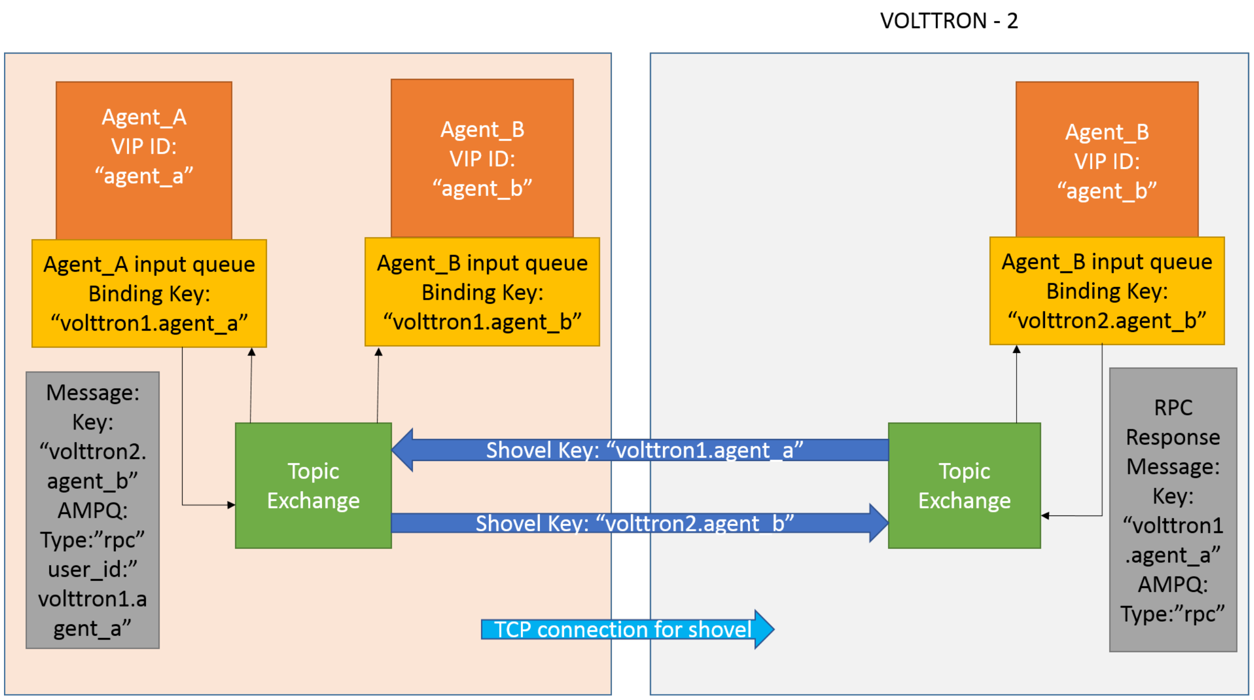
```
agent_b.vip.subscribe.call("pubsub", prefix="devices", all_platforms=True)
```

2. The PubSub subsystem converts the prefix to `__pubsub__.*.devices.#` "*" indicates that agent is subscribing to the "devices" topic from all the VOLTTRON platforms.
3. A new queue is created and bound to VOLTTRON exchange with above binding key.
4. Agent A publishes message to topic *devices/pnnl/isb1/hvac1*
5. PubSub subsystem publishes this message on it's VOLTTRON exchange.
6. Due to a shovel link from VOLTTRON instance "volttron1" to "volttron2", the message is forwarded from VOLTTRON exchange "volttron1" to "volttron2" and is picked up by Agent A on "volttron2".

Multi-Platform RPC With Shovel

After the shovel link is established for multi-platform RPC, the below figure shows how the RPC communication happens.

Note: It is mandatory to have shovel links on both directions as it is request-response type of communication. We will need to set the agent identities for caller and callee in the *$VOLTTRON_HOME/rabbitmq_shovel_config.yml*. The "blue" arrows show the resulting the shovel binding key.


VOLTTRON - 2
Agent_A VIP ID: "agent_a"
Agent_B VIP ID: "agent_b"
Agent_B VIP ID: "agent_b"
Agent_A input queue Binding Key: "volttron1.agent_a"
Agent_B input queue Binding Key: "volttron1.agent_b"
Agent_B input queue Binding Key: "volttron2.agent_b"
Message: Key: "volttron2.agent_b" AMPQ: Type:"rpc" user_id:"volttron1.agent_a"
Topic Exchange
Shovel Key: "volttron1.agent_a"
Shovel Key: "volttron2.agent_b"
Topic Exchange
RPC Response Message: Key: "volttron1.agent_a" AMPQ: Type:"rpc"
TCP connection for shovel

Consider Agent A on VOLTTRON instance "volttron1" on host "host_A" wants to make RPC call on Agent B on VOLTTRON instance "volttron2" on host "host_B".

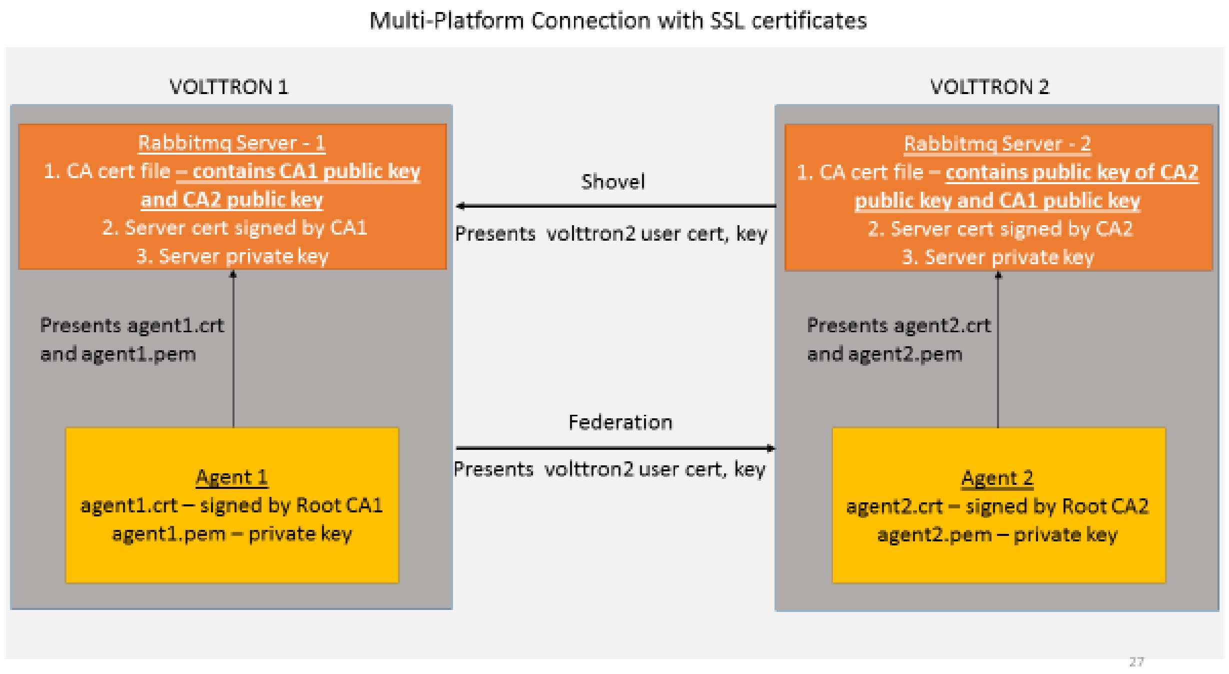
1. Agent A makes RPC call:

```
kwargs = {"external_platform": self.destination_instance_name}
agent_a.vip.rpc.call("agent_b", set_point, "point_name", 2.5, \**kwargs)
```

2. The message is transferred over shovel link to VOLTTRON instance "volttron2".
3. The RPC subsystem of Agent B calls the actual RPC method and gets the result. It encapsulates the message result into a VIP message object and sends it back to Agent A on VOLTTRON instance "volttron1".
4. The RPC subsystem on Agent A receives the message result and gives it to Agent A's application.

Multi-Platform Communication With RabbitMQ SSL

For multi-platform communication over federation and shovel, we need the connecting instances to trust each other.


Multi-Platform Connection with SSL certificates
VOLTTRON 1
VOLTTRON 2
Rabbitmq Server - 1
1. CA cert file – contains CA1 public key and CA2 public key
2. Server cert signed by CA1
3. Server private key
Rabbitmq Server - 2
1. CA cert file – contains public key of CA2 public key and CA1 public key
2. Server cert signed by CA2
3. Server private key
Shovel
Presents volttron2 user cert, key
Presents agent1.crt and agent1.pem
Presents agent2.crt and agent2.pem
Federation
Presents volttron2 user cert, key
Agent 1
agent1.crt – signed by Root CA1
agent1.pem – private key
Agent 2
agent2.crt – signed by Root CA2
agent2.pem – private key
27

Suppose there are two VMs (VOLTTRON1 and VOLTTRON2) running single instances of RabbitMQ, and VOLTTRON1 and VOLTTRON2 want to talk to each other via either the federation or shovel plugins. In order for VOLTTRON1 to talk to VOLTTRON2, VOLTTRON1's root certificate must be appended to VOLTTRON's trusted CA certificate, so that when VOLTTRON1 presents it's root certificate during connection, VOLTTRON2's RabbitMQ server can trust the connection. VOLTTRON2's root CA must be appended to VOLTTRON1's root CA and it must in turn present its root certificate during connection, so that VOLTTRON1 will know it is safe to talk to VOLTTRON2.

Agents trying to connect to remote instance directly need to have a public certificate signed by the remote instance for authenticated SSL based connection. To facilitate this process, the VOLTTRON platform exposes a web based server API for requesting, listing, approving and denying certificate requests. For more detailed description, refer to *Agent communication to Remote RabbitMQ instance*

Multi-Platform Multi-Bus

This guide describes the setup process for a multi-platform connection that has a combination of ZeroMQ and RabbitMQ instances. For this example, we want to use the Forwarder to pass device data from two VOLTTRON instance to a single "central" instance for storage. It will also have a Volttron Central agent running on the "central" instance and Volttron Central Platform agents on all 3 instances and connected to "central" instance to provide operational status of it's instance to the "central" instance. For this document "node" will be used interchangeably with VOLTTRON instance.

Node Setup

For this example we will have two types of nodes; a data collector and a central node. Each of the data collectors will have different message buses (VOLTTRON supports both RabbitMQ and ZeroMQ). The nodes will be configured as in the following table.

Table 2: Node Configuration

	Central	Node-ZMQ	Node-RMQ
Node Type	Central	Data Collector	Data Collector
Master Driver		yes	yes
Forwarder		yes	yes
SQL Historian	yes		
Volttron Central	yes		
Volttron Central Platform	yes	yes	yes
Exposes RMQ Port	yes		
Exposes ZMQ Port	yes		
Exposes HTTPS Port	yes		

The goal of this is to be able to see the data from Node-ZMQ and Node-RMQ in the Central SQL Historian and on the trending charts of Volttron Central.

Virtual Machine Setup

The first step in creating a VOLTTRON instance is to make sure the machine is ready for VOLTTRON. Each machine should have its hostname setup. For this walk-through, the hostnames "central", "node-zmq" and "node-rmq" will be used.

For Central and Node-RMQ follow the instructions *platform installation steps for RMQ*. For Node-ZMQ use *Platform Installation steps for ZeroMQ*.

Instance Setup

The following conventions/assumptions are made for the rest of this document:

- Commands should be run from the VOLTTRON root
- Default values are used for VOLTTRON_HOME($HOME/.volttron), VIP port (22916), HTTPS port (8443), rabbitmq ports (5671 for AMQPs and 15671 for RabbitMQ management interface). If using different *VOLTTRON_HOME* or ports, please replace accordingly.
- Replace central, node-zmq and node-rmq with your own hostnames.
- user will represent your current user.

The following will use *vcfg* (volttron-cfg) to configure the individual platforms.

Central Instance Setup

Note: This instance must have been bootstrapped using `--rabbitmq` see *RabbitMq installation instructions*.

Next step would be to configure the instance to have a web interface to accept/deny incoming certificate signing requests from other instances. Additionally, we will need to install a Volttron Central agent, Volttron Central Platform agent, SQL historian agent and a Listener agent. The following shows an example command output for this setup.

```
(volttron)user@central:~/volttron$ vcfg

Your VOLTTRON_HOME currently set to: /home/user/.volttron

Is this the volttron you are attempting to setup? [Y]:
What type of message bus (rmq/zmq)? [zmq]: rmq
Name of this volttron instance: [volttron1]: central
RabbitMQ server home: [/home/user/rabbitmq_server/rabbitmq_server-3.7.7]:
Fully qualified domain name of the system: [central]:
Would you like to create a new self signed root CAcertificate for this instance: [Y]:

Please enter the following details for root CA certificate
    Country: [US]:
    State: WA
    Location: Richland
    Organization: PNNL
    Organization Unit: volttron
Do you want to use default values for RabbitMQ home, ports, and virtual host: [Y]:
2020-04-13 13:29:36,347 rmq_setup.py INFO: Starting RabbitMQ server
2020-04-13 13:29:46,528 rmq_setup.py INFO: Rmq server at /home/user/rabbitmq_server/
↪rabbitmq_server-3.7.7 is running at
2020-04-13 13:29:46,554 volttron.utils.rmq_mgmt DEBUG: Creating new VIRTUAL HOST:␣
↪volttron
2020-04-13 13:29:46,582 volttron.utils.rmq_mgmt DEBUG: Create READ, WRITE and␣
↪CONFIGURE permissions for the user: central-admin
Create new exchange: volttron, {'durable': True, 'type': 'topic', 'arguments': {
↪'alternate-exchange': 'undeliverable'}}
Create new exchange: undeliverable, {'durable': True, 'type': 'fanout'}
2020-04-13 13:29:46,600 rmq_setup.py INFO:
Checking for CA certificate

2020-04-13 13:29:46,601 rmq_setup.py INFO:
 Creating root ca for volttron instance: /home/user/.volttron/certificates/certs/
↪central-root-ca.crt
2020-04-13 13:29:46,601 rmq_setup.py INFO: Creating root ca with the following info: {
↪'C': 'US', 'ST': 'WA', 'L': 'Richland', 'O': 'PNNL', 'OU': 'VOLTTRON', 'CN':
↪'central-root-ca'}
Created CA cert
2020-04-13 13:29:49,668 rmq_setup.py INFO: **Stopped rmq server
2020-04-13 13:30:00,556 rmq_setup.py INFO: Rmq server at /home/user/rabbitmq_server/
↪rabbitmq_server-3.7.7 is running at
2020-04-13 13:30:00,557 rmq_setup.py INFO:

######################

Setup complete for volttron home /home/user/.volttron with instance name=central
Notes:
 - On production environments, restrict write access to /home/user/.volttron/
↪certificates/certs/central-root-ca.crt to only admin user. For example: sudo chown␣
↪root /home/user/.volttron/certificates/certs/central-root-ca.crt and /home/user/.
↪volttron/certificates/certs/central-trusted-cas.crt
 - A new admin user was created with user name: central-admin and password=default_
↪passwd.
   You could change this user's password by logging into https://central:15671/␣
↪Please update /home/user/.volttron/rabbitmq_config.yml if you change password

######################
```

(continues on next page)

(continued from previous page)

```
The rmq message bus has a backward compatibility
layer with current zmq instances. What is the
zmq bus's vip address? [tcp://127.0.0.1]: tcp://192.168.56.101
What is the port for the vip address? [22916]:
Is this instance web enabled? [N]: y
Web address set to: https://central
What is the port for this instance? [8443]:
Is this an instance of volttron central? [N]: y
Configuring /home/user/volttron/services/core/VolttronCentral.
Installing volttron central.
['volttron', '-vv', '-l', '/home/user/.volttron/volttron.cfg.log']
Should the agent autostart? [N]: y
VC admin and password are set up using the admin web interface.
After starting VOLTTRON, please go to https://central:8443/admin/login.html to␣
↪complete the setup.
Will this instance be controlled by volttron central? [Y]:
Configuring /home/user/volttron/services/core/VolttronCentralPlatform.
What is the name of this instance? [central]:
Volttron central address set to https://central:8443
['volttron', '-vv', '-l', '/home/user/.volttron/volttron.cfg.log']
Should the agent autostart? [N]:
Would you like to install a platform historian? [N]: y
Configuring /home/user/volttron/services/core/SQLHistorian.
['volttron', '-vv', '-l', '/home/user/.volttron/volttron.cfg.log']
Should the agent autostart? [N]: y
Would you like to install a master driver? [N]:
Would you like to install a listener agent? [N]: y
Configuring examples/ListenerAgent.
['volttron', '-vv', '-l', '/home/user/.volttron/volttron.cfg.log']
Should the agent autostart? [N]: y
Finished configuration!

You can now start the volttron instance.

If you need to change the instance configuration you can edit
the config file is at /home/user/.volttron/config
```

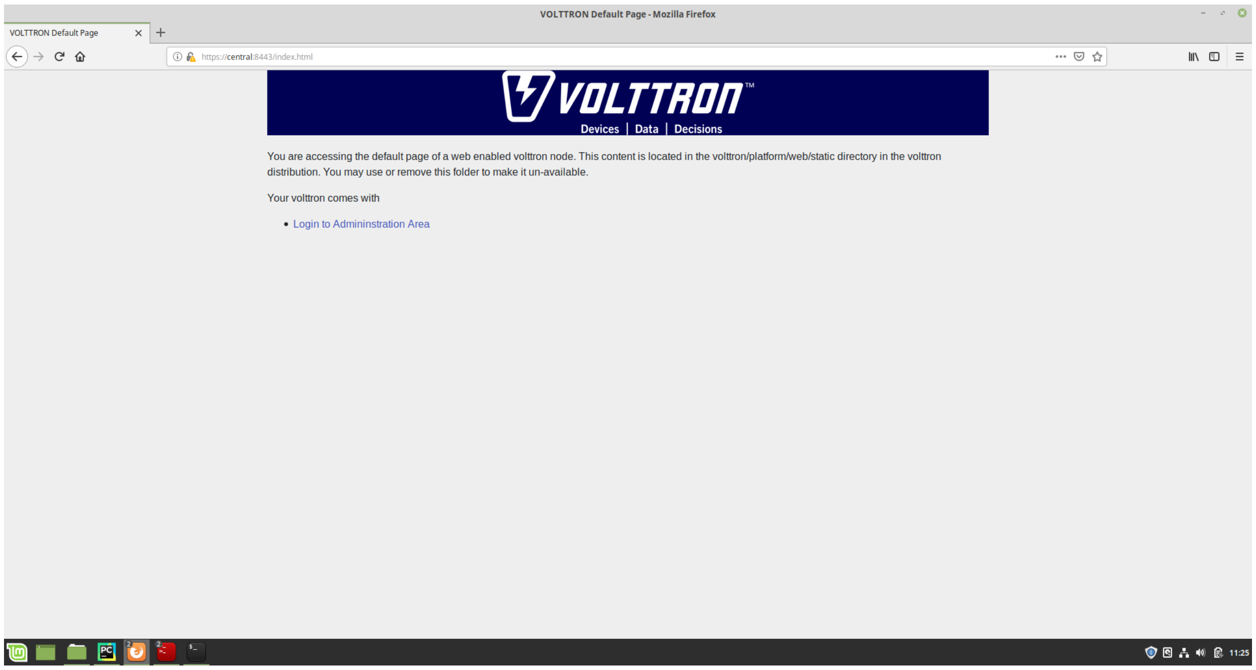
Start VOLTTRON instance and check if the agents are installed.

```
./start-volttron
vctl status
```

Open browser and go to master admin authentication page *https://central:8443/index.html* to accept/reject incoming certificate signing request (CSR) from other platforms.

Note: Replace "central" with the proper hostname of VC instance in the admin page URL. If opening the admin page from a different system, then please make that the hostname is resolvable in that machine.

Click on "Login To Admistration Area".

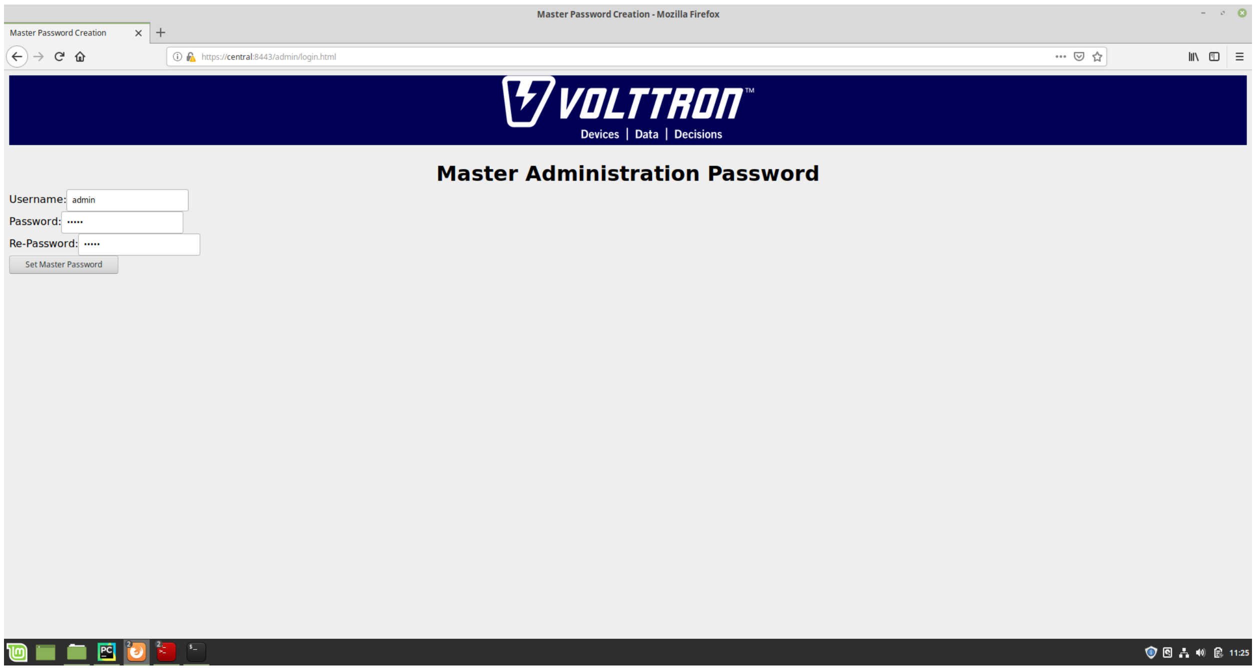

VOLTTRON Default Page - Mozilla Firefox
VOLTTRON Default Page
https://central:8443/index.html
VOLTTRON
Devices | Data | Decisions

You are accessing the default page of a web enabled volttron node. This content is located in the volttron/platform/web/static directory in the volttron distribution. You may use or remove this folder to make it un-available.

Your volttron comes with

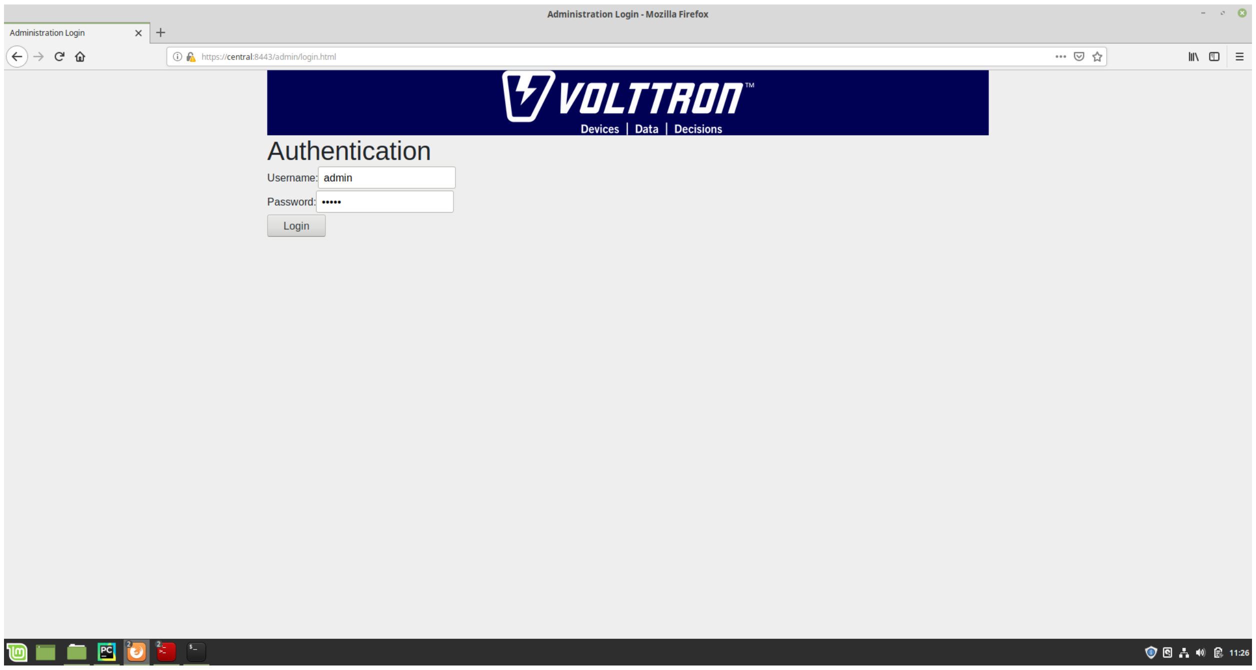
- Login to Admininstration Area

Set the master admin username and password. This can be later used to login into master admin authentication page. This username and password will also be used to log in to Volttron Central.


Master Password Creation - Mozilla Firefox
Master Password Creation
https://central:8443/admin/login.html
VOLTTRON
Devices | Data | Decisions
Master Administration Password

Username: admin
Password: •••••
Re-Password: •••••
Set Master Password

Login into the Master Admin page.

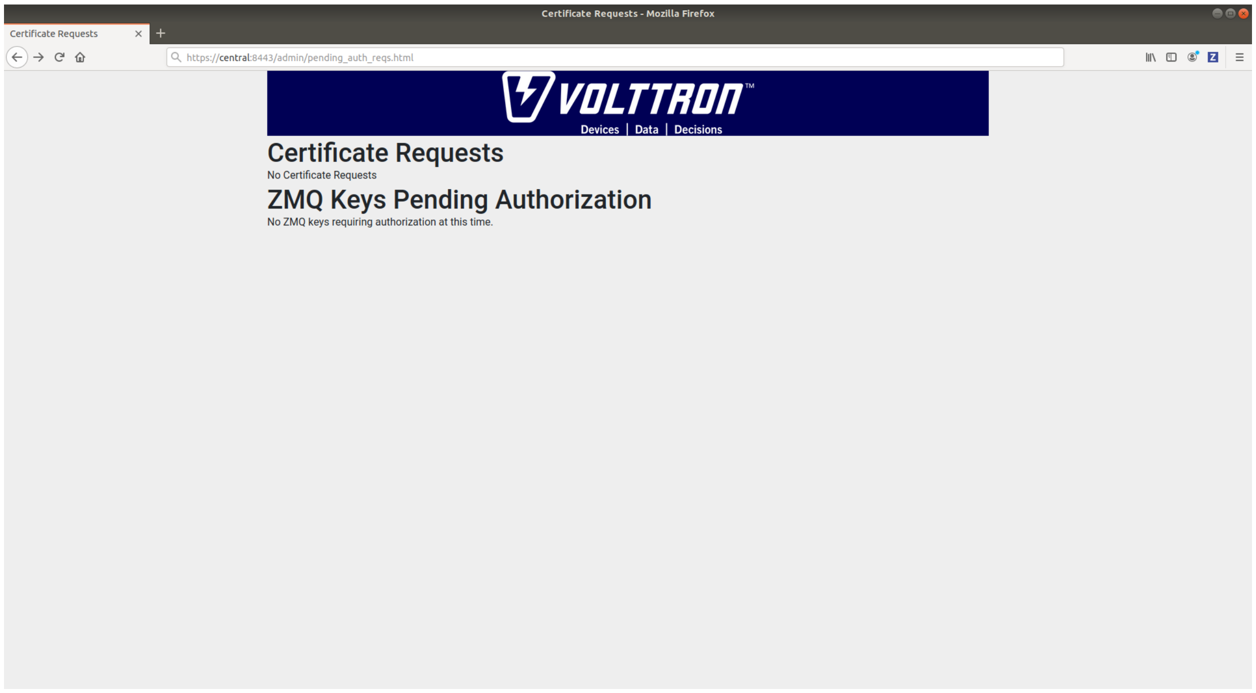

Administration Login - Mozilla Firefox
Administration Login
https://central:8443/admin/login.html
VOLTTRON
Devices | Data | Decisions
Login

Authentication

Username: admin

Password: •••••

After logging in, you will see no CSR requests initially.


Certificate Requests - Mozilla Firefox
Certificate Requests
https://central:8443/admin/pending_auth_reqs.html
VOLTTRON
Devices | Data | Decisions

Certificate Requests

No Certificate Requests

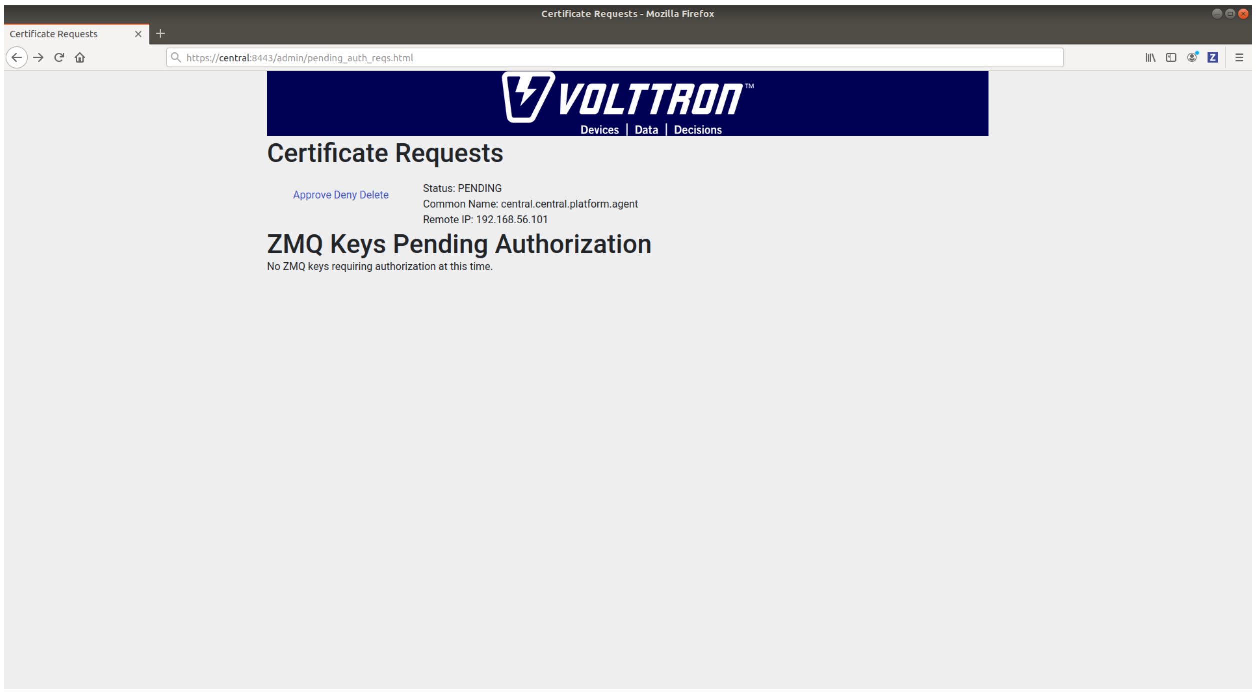
ZMQ Keys Pending Authorization

No ZMQ keys requiring authorization at this time.

Go back to the terminal and start Volttron Central Platform agent on the "central" instance. The agent will send a CSR request to the web interface.

```
vctl start --tag vcp
```

Now go to master admin page to check if there is a new pending CSR request. You will see a "PENDING" request from "central.central.platform.agent"


Certificate Requests - Mozilla Firefox
Certificate Requests
https://central:8443/admin/pending_auth_reqs.html
VOLTTRON
Devices | Data | Decisions

Certificate Requests

Approve Deny Delete

Status: PENDING
Common Name: central.central.platform.agent
Remote IP: 192.168.56.101

ZMQ Keys Pending Authorization

No ZMQ keys requiring authorization at this time.

Approve the CSR request to allow authenticated SSL based connection to the “central” instance.

Go back to the terminal and check the status of Voltron Central Platform agent. It should be set to “GOOD”.

Node-ZMQ Instance Setup

On the “node-zmq” VM, setup a ZeroMQ based VOLTTRON instance. Using “vcfg” command, install Volttron Central Platform agent, a master driver agent with a fake driver.

Note: This instance will use old ZeroMQ based authentication mechanism using CURVE keys.

```
(volttron)user@node-zmq:~/volttron$ vcfg

Your VOLTTRON_HOME currently set to: /home/user/.volttron

Is this the volttron you are attempting to setup? [Y]:
What type of message bus (rmq/zmq)? [zmq]:
What is the vip address? [tcp://127.0.0.1]:
What is the port for the vip address? [22916]:
Is this instance web enabled? [N]:
Will this instance be controlled by volttron central? [Y]:
Configuring /home/user/volttron/services/core/VolttronCentralPlatform.
What is the name of this instance? [volttron1]: collector1
What is the hostname for volttron central? [http://node-zmq]: https://central
What is the port for volttron central? [8080]: 8443
['volttron', '-vv', '-l', '/home/user/.volttron/volttron.cfg.log']
Should the agent autostart? [N]:
Would you like to install a platform historian? [N]:
Would you like to install a master driver? [N]: y
Configuring /home/user/volttron/services/core/MasterDriverAgent.
['volttron', '-vv', '-l', '/home/user/.volttron/volttron.cfg.log']
```

(continues on next page)

(continued from previous page)

```
Would you like to install a fake device on the master driver? [N]: y
Should the agent autostart? [N]: y
Would you like to install a listener agent? [N]:
Finished configuration!

You can now start the volttron instance.

If you need to change the instance configuration you can edit
the config file is at /home/user/.volttron/config
```

Please note the Volttron Central web-address should point to that of the "central" instance.

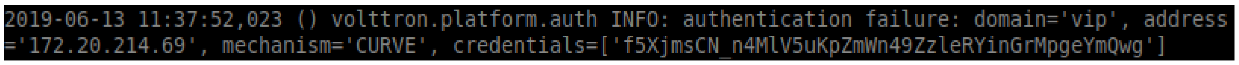
Start VOLTTRON instance and check if the agents are installed.

```
./start-volttron
vctl status
```

Start Volttron Central Platform on this platform manually.

```
vctl start --tag vcp
```

Check the VOLTTRON log in the "central" instance, you will see "authentication failure" entry from the incoming connection. You will need to add the public key of VCP agent on the "central" instance.

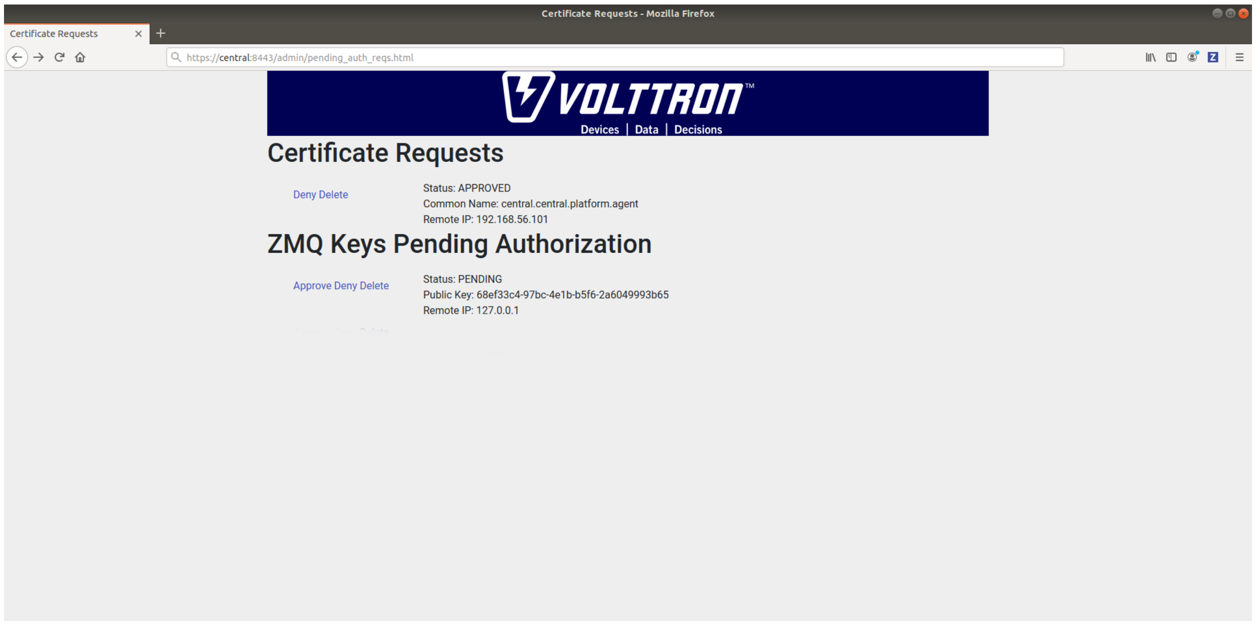


```
2019-06-13 11:37:52,023 () volttron.platform.auth INFO: authentication failure: domain='vip', address
='172.20.214.69', mechanism='CURVE', credentials=['f5XjmsCN_n4MlV5uKpZmWn49ZzleRYinGrMpgeYmQwg']
```

At this point, you can either accept the connection through the admin page or the command line.

Using the admin page:

Navigate back to the master admin authentication page. You should see a pending request under the ZMQ Keys Pending Authorization header.



Certificate Requests - Mozilla Firefox
Certificate Requests
https://central:8443/admin/pending_auth_reqs.html
VOLTTRON™
Devices | Data | Decisions

Certificate Requests

Deny Delete

Status: APPROVED
Common Name: central.central.platform.agent
Remote IP: 192.168.56.101

ZMQ Keys Pending Authorization

Approve Deny Delete

Status: PENDING
Public Key: 68ef33c4-97bc-4e1b-b5f6-2a6049993b65
Remote IP: 127.0.0.1

Accept the credential in the same method as a CSR.

Using the command line:

On the "node-zmq" box execute this command and grab the public key of the VCP agent.

```
vctl auth publickey
```

Add auth entry corresponding to VCP agent on "central" instance using the below command. Replace the user id value and credentials value appropriately before running

```
vctl auth add --user_id <any unique user id. for example zmq_node_vcp> --credentials
↪<public key of vcp on zmq node>
```

Complete similar steps to start a forwarder agent that connects to "central" instance. Modify the configuration in *services/core/ForwardHistorian/rmq_config.yml* to have a destination VIP address pointing to VIP address of the "central" instance and server key of the "central" instance.

```
---
destination-vip: tcp://<ip>:22916
destination-serverkey: <serverkey>
```

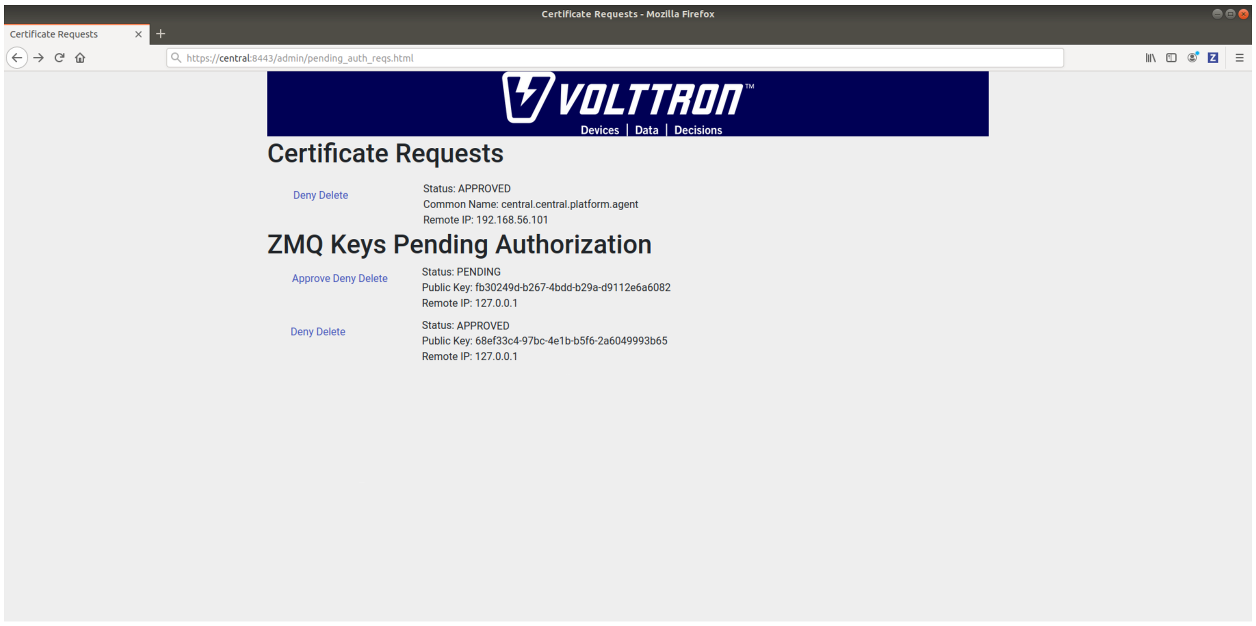
Note: Replace <ip> with public facing IP-address of "central" instance and <serverkey> with serverkey of "central" instance. Use the command **vctl auth serverkey** on the "central" instance to get the server key of the instance

Install and start forwarder agent.

```
python scripts/install-agent.py -s services/core/ForwardHistorian -c services/core/
↪ForwardHistorian/rmq_config.yml --start
```

To accept the credential using the admin page:

Navigate back to the master admin authentication page. You should see another pending request under the ZMQ Keys Pending Authorization header.



Certificate Requests - Mozilla Firefox

https://central:8443/admin/pending_auth_reqs.html

Certificate Requests

Deny Delete

Status: APPROVED
Common Name: central.central.platform.agent
Remote IP: 192.168.56.101

ZMQ Keys Pending Authorization

Approve Deny Delete

Status: PENDING
Public Key: fb30249d-b267-4bdd-b29a-d9112e6a6082
Remote IP: 127.0.0.1

Deny Delete

Status: APPROVED
Public Key: 68ef33c4-97bc-4e1b-b5f6-2a6049993b65
Remote IP: 127.0.0.1

Accept this credential in the same method as before.

To accept the credential using the command line:

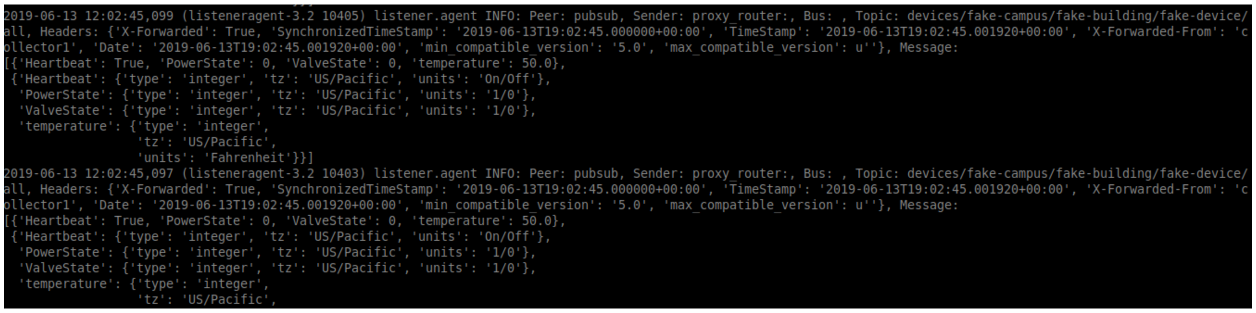
Grab the public key of the forwarder agent.

```
vctl auth publickey
```

Add auth entry corresponding to VCP agent on **central** instance.

```
vctl auth add --user_id <any unique user id. for example zmq_node_forwarder> --
↪credentials <public key of forwarder on zmq node>
```

In either case, you should start seeing messages from "collector1" instance on the "central" instance's VOLTTRON log now.



```
2019-06-13 12:02:45,099 (listeneragent-3.2 10405) listener.agent INFO: Peer: pubsub, Sender: proxy_router:, Bus: , Topic: devices/fake-campus/fake-building/fake-device/
all, Headers: {'X-Forwarded': True, 'SynchronizedTimeStamp': '2019-06-13T19:02:45.000000+00:00', 'TimeStamp': '2019-06-13T19:02:45.001920+00:00', 'X-Forwarded-From': 'c
ollector1', 'Date': '2019-06-13T19:02:45.001920+00:00', 'min_compatible_version': '5.0', 'max_compatible_version': u''}, Message:
[{'Heartbeat': True, 'PowerState': 0, 'ValveState': 0, 'temperature': 50.0},
 {'Heartbeat': {'type': 'integer', 'tz': 'US/Pacific', 'units': 'On/Off'},
  'PowerState': {'type': 'integer', 'tz': 'US/Pacific', 'units': '1/0'},
  'ValveState': {'type': 'integer', 'tz': 'US/Pacific', 'units': '1/0'},
  'temperature': {'type': 'integer',
                  'tz': 'US/Pacific',
                  'units': 'Fahrenheit'}}]
2019-06-13 12:02:45,097 (listeneragent-3.2 10403) listener.agent INFO: Peer: pubsub, Sender: proxy_router:, Bus: , Topic: devices/fake-campus/fake-building/fake-device/
all, Headers: {'X-Forwarded': True, 'SynchronizedTimeStamp': '2019-06-13T19:02:45.000000+00:00', 'TimeStamp': '2019-06-13T19:02:45.001920+00:00', 'X-Forwarded-From': 'c
ollector1', 'Date': '2019-06-13T19:02:45.001920+00:00', 'min_compatible_version': '5.0', 'max_compatible_version': u''}, Message:
[{'Heartbeat': True, 'PowerState': 0, 'ValveState': 0, 'temperature': 50.0},
 {'Heartbeat': {'type': 'integer', 'tz': 'US/Pacific', 'units': 'On/Off'},
  'PowerState': {'type': 'integer', 'tz': 'US/Pacific', 'units': '1/0'},
  'ValveState': {'type': 'integer', 'tz': 'US/Pacific', 'units': '1/0'},
  'temperature': {'type': 'integer',
                  'tz': 'US/Pacific',
```

Node-RMQ Instance Setup

Note: This instance must have been bootstrapped using –rabbitmq see *RabbitMq installation instructions*.

Using "vcfg" command, install Volttron Central Platform agent, a master driver agent with fake driver. The instance name is set to "collector2".

```
(volttron)user@node-rmq:~/volttron$ vcfg

Your VOLTTRON_HOME currently set to: /home/user/.volttron

Is this the volttron you are attempting to setup? [Y]:
What type of message bus (rmq/zmq)? [zmq]: rmq
Name of this volttron instance: [volttron1]: collector2
RabbitMQ server home: [/home/user/rabbitmq_server/rabbitmq_server-3.7.7]:
Fully qualified domain name of the system: [node-rmq]:
Would you like to create a new self signed root CA certificate for this instance: [Y]:

Please enter the following details for root CA certificate
    Country: [US]:
    State: WA
    Location: Richland
    Organization: PNNL
    Organization Unit: volttron
Do you want to use default values for RabbitMQ home, ports, and virtual host: [Y]:
2020-04-13 13:29:36,347 rmq_setup.py INFO: Starting RabbitMQ server
2020-04-13 13:29:46,528 rmq_setup.py INFO: Rmq server at /home/user/rabbitmq_server/
↪rabbitmq_server-3.7.7 is running at
2020-04-13 13:29:46,554 volttron.utils.rmq_mgmt DEBUG: Creating new VIRTUAL HOST:␣
↪volttron
2020-04-13 13:29:46,582 volttron.utils.rmq_mgmt DEBUG: Create READ, WRITE and␣
↪CONFIGURE permissions for the user: collector2-admin
Create new exchange: volttron, {'durable': True, 'type': 'topic', 'arguments': {
↪'alternate-exchange': 'undeliverable'}}
```

(continues on next page)

(continued from previous page)

```
Create new exchange: undeliverable, {'durable': True, 'type': 'fanout'}
2020-04-13 13:29:46,600 rmq_setup.py INFO:
Checking for CA certificate

2020-04-13 13:29:46,601 rmq_setup.py INFO:
 Creating root ca for volttron instance: /home/user/.volttron/certificates/certs/
↪collector2-root-ca.crt
2020-04-13 13:29:46,601 rmq_setup.py INFO: Creating root ca with the following info: {
↪'C': 'US', 'ST': 'WA', 'L': 'Richland', 'O': 'PNNL', 'OU': 'VOLTTRON', 'CN':
↪'collector2-root-ca'}
Created CA cert
2020-04-13 13:29:49,668 rmq_setup.py INFO: **Stopped rmq server
2020-04-13 13:30:00,556 rmq_setup.py INFO: Rmq server at /home/user/rabbitmq_server/
↪rabbitmq_server-3.7.7 is running at
2020-04-13 13:30:00,557 rmq_setup.py INFO:

#######################

Setup complete for volttron home /home/user/.volttron with instance name=collector2
Notes:
 - On production environments, restrict write access to /home/user/.volttron/
↪certificates/certs/collector2-root-ca.crt to only admin user. For example: sudo␣
↪chown root /home/user/.volttron/certificates/certs/collector2-root-ca.crt and /home/
↪user/.volttron/certificates/certs/collector2-trusted-cas.crt
 - A new admin user was created with user name: collector2-admin and password=default_
↪passwd.
   You could change this user's password by logging into https://node-rmq:15671/␣
↪Please update /home/user/.volttron/rabbitmq_config.yml if you change password

#######################

The rmq message bus has a backward compatibility
layer with current zmq instances. What is the
zmq bus's vip address? [tcp://127.0.0.1]:
What is the port for the vip address? [22916]:
Is this instance web enabled? [N]:
Will this instance be controlled by volttron central? [Y]:
Configuring /home/user/volttron/services/core/VolttronCentralPlatform.
What is the name of this instance? [collector2]:
What is the hostname for volttron central? [http://node-rmq]: https://central
What is the port for volttron central? [8443]:
['volttron', '-vv', '-l', '/home/user/.volttron/volttron.cfg.log']
Should the agent autostart? [N]:
Would you like to install a platform historian? [N]:
Would you like to install a master driver? [N]: y
Configuring /home/user/volttron/services/core/MasterDriverAgent.
['volttron', '-vv', '-l', '/home/user/.volttron/volttron.cfg.log']
Would you like to install a fake device on the master driver? [N]: y
Should the agent autostart? [N]: y
Would you like to install a listener agent? [N]:
Finished configuration!

You can now start the volttron instance.

If you need to change the instance configuration you can edit
the config file is at /home/user/.volttron/config
```

Note: The Volttron Central web-address should point to that of the "central" instance.

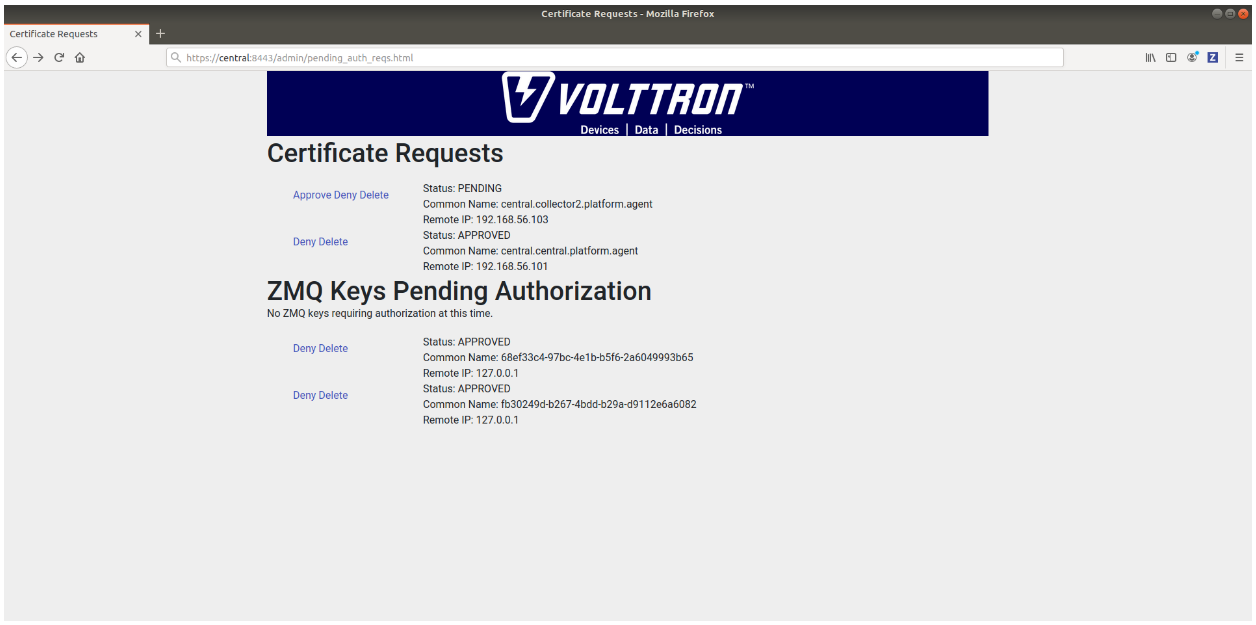
Start VOLTTRON instance and check if the agents are installed.

```
./start-volttron
vctl status
```

Start Volttron Central Platform on this platform manually.

```
vctl start --tag vcp
```

Go the master admin authentication page and check if there is a new pending CSR request from VCP agent of "collector2" instance.


Certificate Requests - Mozilla Firefox
Certificate Requests
https://central:8443/admin/pending_auth_reqs.html
VOLTTRON
Devices | Data | Decisions

Certificate Requests

Approve Deny Delete
Status: PENDING
Common Name: central.collector2.platform.agent
Remote IP: 192.168.56.103

Deny Delete
Status: APPROVED
Common Name: central.central.platform.agent
Remote IP: 192.168.56.101

ZMQ Keys Pending Authorization

No ZMQ keys requiring authorization at this time.

Deny Delete
Status: APPROVED
Common Name: 68ef33c4-97bc-4e1b-b5f6-2a6049993b65
Remote IP: 127.0.0.1

Deny Delete
Status: APPROVED
Common Name: fb30249d-b267-4bdd-b29a-d9112e6a6082
Remote IP: 127.0.0.1

Approve the CSR request to allow authenticated SSL based connection to the "central" instance.

Now go back to the terminal and check the status of Volttron Central Platform agent. It should be set to "GOOD".

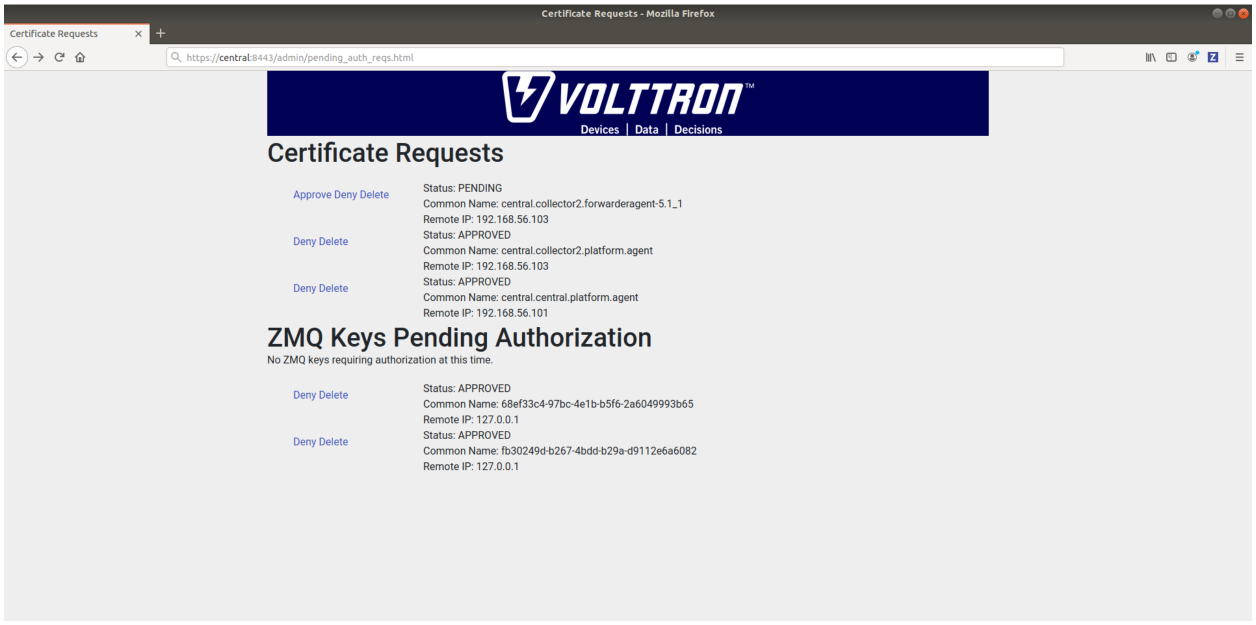
Let's now install a forwarder agent on this instance to forward local messages matching "devices" topic to external "central" instance. Modify the configuration in *services/core/ForwardHistorian/rmq_config.yml* to have a destination address pointing to web address of the "central" instance.

```
---
destination-address: https://central:8443
```

Start forwarder agent.

```
python scripts/install-agent.py -s services/core/ForwardHistorian -c services/core/
↪ForwardHistorian/rmq_config.yml --start
```

Go the master admin authentication page and check if there is a new pending CSR request from forwarder agent of "collector2" instance.


Certificate Requests - Mozilla Firefox
Certificate Requests
https://central:8443/admin/pending_auth_reqs.html
VOLTTRON
Devices | Data | Decisions

Certificate Requests

Approve Deny Delete — Status: PENDING
Common Name: central.collector2.forwarderagent-5.1_1
Remote IP: 192.168.56.103

Deny Delete — Status: APPROVED
Common Name: central.collector2.platform.agent
Remote IP: 192.168.56.103

Deny Delete — Status: APPROVED
Common Name: central.central.platform.agent
Remote IP: 192.168.56.101

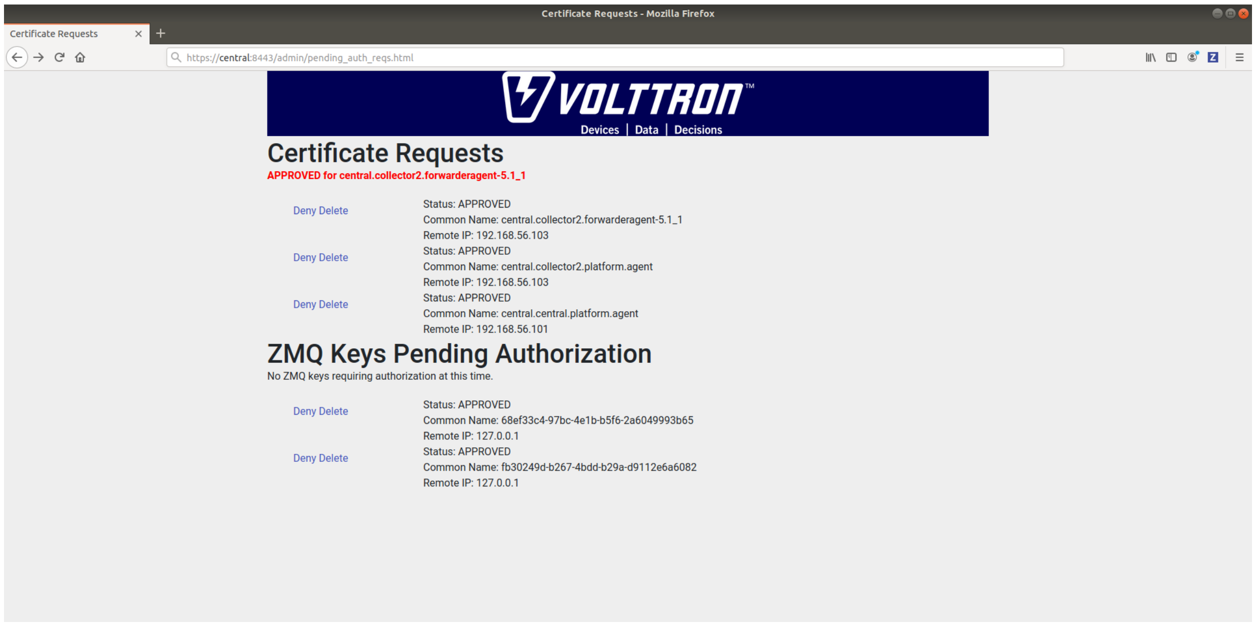
ZMQ Keys Pending Authorization

No ZMQ keys requiring authorization at this time.

Deny Delete — Status: APPROVED
Common Name: 68ef33c4-97bc-4e1b-b5f6-2a6049993b65
Remote IP: 127.0.0.1

Deny Delete — Status: APPROVED
Common Name: fb30249d-b267-4bdd-b29a-d9112e6a6082
Remote IP: 127.0.0.1

Approve the CSR request to allow authenticated SSL based connection to the "central" instance.


Certificate Requests - Mozilla Firefox
Certificate Requests
https://central:8443/admin/pending_auth_reqs.html
VOLTTRON
Devices | Data | Decisions

Certificate Requests

APPROVED for central.collector2.forwarderagent-5.1_1

Deny Delete — Status: APPROVED
Common Name: central.collector2.forwarderagent-5.1_1
Remote IP: 192.168.56.103

Deny Delete — Status: APPROVED
Common Name: central.collector2.platform.agent
Remote IP: 192.168.56.103

Deny Delete — Status: APPROVED
Common Name: central.central.platform.agent
Remote IP: 192.168.56.101

ZMQ Keys Pending Authorization

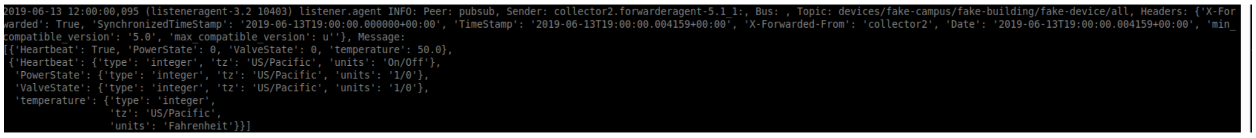
No ZMQ keys requiring authorization at this time.

Deny Delete — Status: APPROVED
Common Name: 68ef33c4-97bc-4e1b-b5f6-2a6049993b65
Remote IP: 127.0.0.1

Deny Delete — Status: APPROVED
Common Name: fb30249d-b267-4bdd-b29a-d9112e6a6082
Remote IP: 127.0.0.1

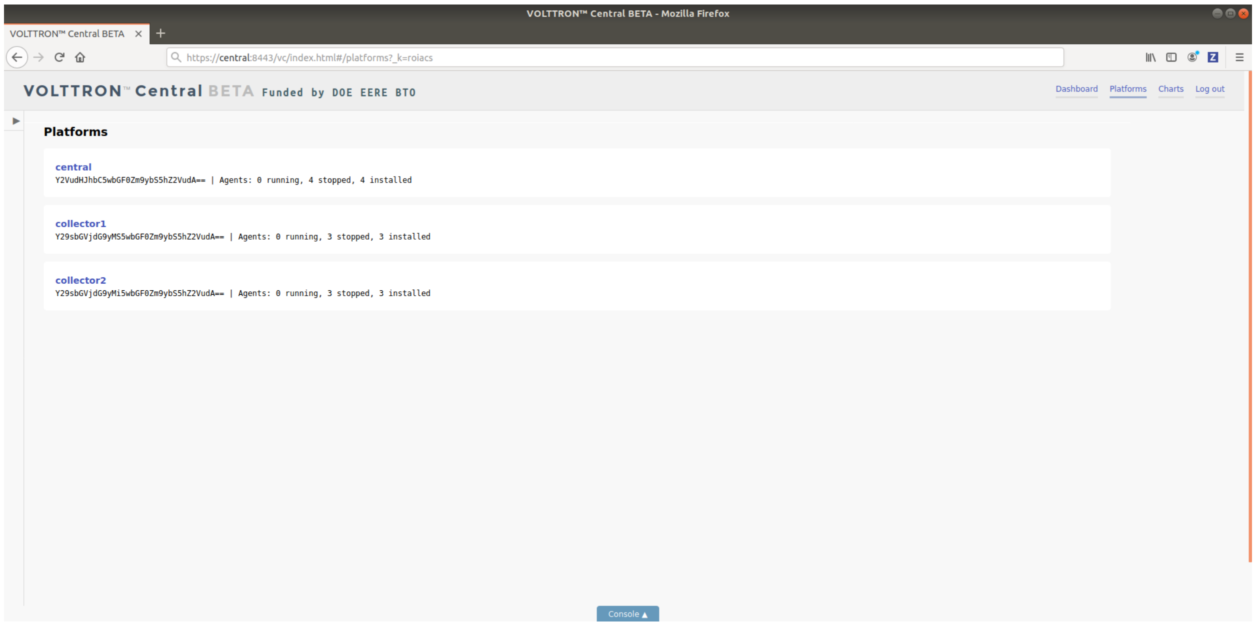
Now go back to the terminal and check the status of forwarder agent. It should be set to "GOOD".

Check the VOLTTRON log of "central" instance. You should see messages with "devices" topic coming from "collector2" instance.



```
2019-06-13 12:00:00,095 (listeneragent-3.2 10403) listener.agent INFO: Peer: pubsub, Sender: collector2.forwarderagent-5.1_1:, Bus: , Topic: devices/fake-campus/fake-building/fake-device/all, Headers: {'X-For
warded': True, 'SynchronizedTimeStamp': '2019-06-13T19:00:00.000000+00:00', 'TimeStamp': '2019-06-13T19:00:00.004159+00:00', 'X-Forwarded-From': 'collector2', 'Date': '2019-06-13T19:00:00.004159+00:00', 'min_
compatible_version': '5.0', 'max_compatible_version': u''}, Message:
[{'Heartbeat': True, 'PowerState': 0, 'ValveState': 0, 'temperature': 50.0},
 {'Heartbeat': {'type': 'integer', 'tz': 'US/Pacific', 'units': 'On/Off'},
  'PowerState': {'type': 'integer', 'tz': 'US/Pacific', 'units': '1/0'},
  'ValveState': {'type': 'integer', 'tz': 'US/Pacific', 'units': '1/0'},
  'temperature': {'type': 'integer',
                  'tz': 'US/Pacific',
                  'units': 'Fahrenheit'}}]
```

To confirm that VolttronCentral is monitoring the status of all the 3 platforms, open a browser and type this URL *https://central:8443/vc/index.html*. Login using credentials (username and password) earlier set during the VC configuration step (using vcfg command in "central" instance). Click on "platforms" tab in the far right corner. You should see all three platforms listed in that page. Click on each of the platforms and check the status of the agents.


VOLTTRON™ Central BETA - Mozilla Firefox
VOLTTRON™ Central BETA
https://central:8443/vc/index.html#/platforms?_k=roiacs
VOLTTRON™ Central BETA Funded by DOE EERE BTO
Dashboard
Platforms
Charts
Log out
Console ▲

Platforms

central
Y2VudHJhbC5wbGF0Zm9ybS5hZ2VudA== | Agents: 0 running, 4 stopped, 4 installed

collector1
Y29sbGVjdG9yMS5wbGF0Zm9ybS5hZ2VudA== | Agents: 0 running, 3 stopped, 3 installed

collector2
Y29sbGVjdG9yMi5wbGF0Zm9ybS5hZ2VudA== | Agents: 0 running, 3 stopped, 3 installed

VOLTTRON Central Deployment

VOLTTRON Central is a platform management web application that allows platforms to communicate and to be managed from a centralized server. This agent alleviates the need to ssh into independent nodes in order to manage them. The demo will start up three different instances of VOLTTRON with three historians and different agents on each host. The following entries will help to navigate around the VOLTTRON Central interface.

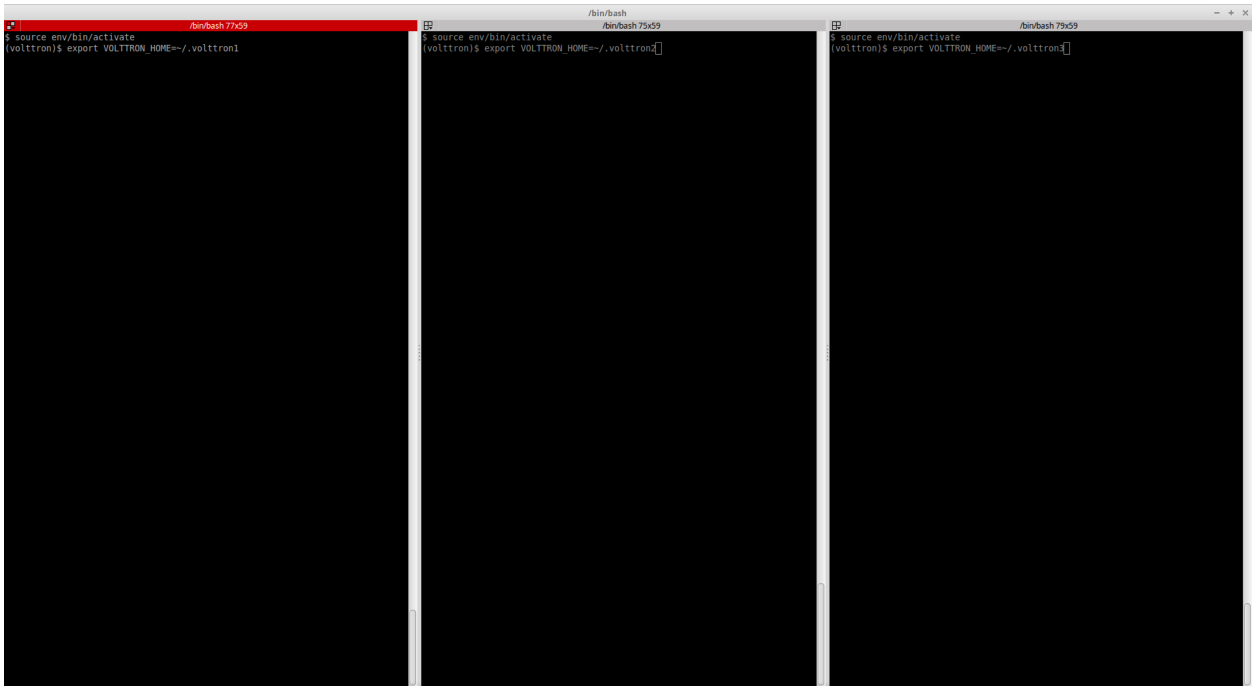
- *Getting Started*
- *Remote Platform Configuration*
- *Starting the Demo*
- *Stopping the Demo*
- *Log In*
- *Log Out*
- *Platforms Tree*
- *Loading the Tree*
- *Health Status*
- *Filter the Tree*
- *Platforms Screen*
- *Register New Platform*
- *Deregister Platform*
- *Platform View*
- *Add Charts*
- *Dashboard Charts*
- *Remove Charts*

Getting Started

After *installing VOLTTRON*, open three shells with the current directory the root of the VOLTTRON repository. Then activate the VOLTTRON environment and export the *VOLTTRON_HOME* variable. The home variable needs to be different for each instance.

If you are using Terminator you can right click and select "Split Vertically". This helps us keep from losing terminal windows or duplicating work.

```
$ source env/bin/activate
$ export VOLTTRON_HOME=~/.volttron1
```



```
$ source env/bin/activate
(volttron)$ export VOLTTRON_HOME=~/.volttron1
```

```
$ source env/bin/activate
(volttron)$ export VOLTTRON_HOME=~/.volttron2
```

```
$ source env/bin/activate
(volttron)$ export VOLTTRON_HOME=~/.volttron3
```

One of our instances will have a VOLTTRON Central agent. We will install a platform agent and a historian on all three platforms. Please note, for this demo all the instances run on ZeroMQ messages. For multi-platform, multi-bus deployment setup please follow the steps described in *Multi Platform Multi-Bus Deployment*.

Run *vcfg* in the first shell. This command will ask how the instance should be set up. Many of the options have defaults that will be sufficient. When asked if this instance is a VOLTTRON Central enter *y*. Read through the options and use the enter key to accept default options. There are no default credentials for VOLTTRON Central. You can have it install the agents at this time. Below is an example configuration. In this case, username is user and localhost is volttron-pc.

```
(volttron)user@volttron-pc:~/volttron$ vcfg

Your VOLTTRON_HOME currently set to: /home/user/.volttron1

Is this the volttron you are attempting to setup? [Y]:
What type of message bus (rmq/zmq)? [zmq]:
What is the vip address? [tcp://127.0.0.1]:
What is the port for the vip address? [22916]:
Is this instance web enabled? [N]: y
What is the protocol for this instance? [https]:
Web address set to: https://volttron-pc
```

(continues on next page)

(continued from previous page)

```
What is the port for this instance? [8443]:
Would you like to generate a new web certificate? [Y]:
WARNING! CA certificate does not exist.
Create new root CA? [Y]:

Please enter the following details for web server certificate:
        Country: [US]:
        State: WA
        Location: Richland
        Organization: PNNL
        Organization Unit: VOLTTRON
Created CA cert
Creating new web server certificate.
Is this an instance of volttron central? [N]: y
Configuring /home/user/volttron/services/core/VolttronCentral.
Installing volttron central.
Should the agent autostart? [N]: y
VC admin and password are set up using the admin web interface.
After starting VOLTTRON, please go to https://volttron-pc:8443/admin/login.
↪html to complete the setup.
Will this instance be controlled by volttron central? [Y]: y
Configuring /home/user/volttron/services/core/VolttronCentralPlatform.
What is the name of this instance? [volttron1]:
Volttron central address set to https://volttron-pc:8443
Should the agent autostart? [N]: y
Would you like to install a platform historian? [N]: y
Configuring /home/user/volttron/services/core/SQLHistorian.
Should the agent autostart? [N]: y
Would you like to install a master driver? [N]: y
Configuring /home/user/volttron/services/core/MasterDriverAgent.
Would you like to install a fake device on the master driver? [N]: y
Should the agent autostart? [N]: y
Would you like to install a listener agent? [N]: y
Configuring examples/ListenerAgent.
Should the agent autostart? [N]: y
Finished configuration!

You can now start the volttron instance.

If you need to change the instance configuration you can edit
the config file is at /home/user/.volttron1/config

(volttron)user@volttron-pc:~/volttron$
```

VOLTTRON Central needs to accept the connecting instances' public keys. For this example we'll allow any CURVE credentials to be accepted. After *starting*, the command **vctl auth add** will prompt the user for information about how the credentials should be used. We can simply hit Enter to select defaults on all fields except **credentials**, where we will type */.*/*

```
$ vctl auth add --credentials "/.*/"
added entry domain=None, address=None, mechanism='CURVE', credentials=u'/.*/', user_
↪id='63b126a7-2941-4ebe-8588-711d1e6c70d1'
```

For more information on authorization see *authentication*.

Remote Platform Configuration

The next step is to configure the instances that will connect to VOLTTRON Central. In the second and third terminal windows run *vcfg*. Like the VOLTTRON_HOME variable, these instances need to have unique VIP addresses and unique instance names.

Install a platform agent and a historian as before. Since we used the default options when configuring VOLTTRON Central, we can use the default options when configuring these platform agents as well. The configuration will be a little different. The example below is for the second volttron instance. Note the unique VIP address and instance name. Please ensure the web-address of the volttron central is configured correctly.

```
(volttron)user@volttron-pc:~/volttron$ vcfg

Your VOLTTRON_HOME currently set to: /home/user/.volttron2

Is this the volttron you are attempting to setup? [Y]:
What type of message bus (rmq/zmq)? [zmq]:
What is the vip address? [tcp://127.0.0.1]: tcp://127.0.0.2
What is the port for the vip address? [22916]:
Is this instance web enabled? [N]:
Will this instance be controlled by volttron central? [Y]:
Configuring /home/user/volttron/services/core/VolttronCentralPlatform.
What is the name of this instance? [volttron1]: volttron2
What is the hostname for volttron central? [https://volttron-pc]:
What is the port for volttron central? [8443]:
Should the agent autostart? [N]: y
Would you like to install a platform historian? [N]: y
Configuring /home/user/volttron/services/core/SQLHistorian.
Should the agent autostart? [N]: y
Would you like to install a master driver? [N]:
Would you like to install a listener agent? [N]:
Finished configuration!

You can now start the volttron instance.

If you need to change the instance configuration you can edit
the config file is at /home/user/.volttron2/config

(volttron)user@volttron-pc:~/volttron$
```

Starting the Demo

Start each Volttron instance after configuration. You have two options.

Option 1: The following command starts the volttron process in the background. The "-l" option tells volttron to log to a file. The file name should be different for each instance.

```
$ volttron -vv -l volttron.log&
```

Option 2: Use the utility script start-volttron.

```
$ ./start-volttron
```

Note: If you choose to not start your agents with their platforms they will need to be started by hand.

List the installed agents with

```
$ vctl status
```

A portion of each agent's uuid makes up the leftmost column of the status output. This is all that is needed to start or stop the agent. If any installed agents share a common prefix then more of the uuid will be needed to identify it.

```
$ vctl start uuid
```

or

```
$ vctl start --tag tag
```

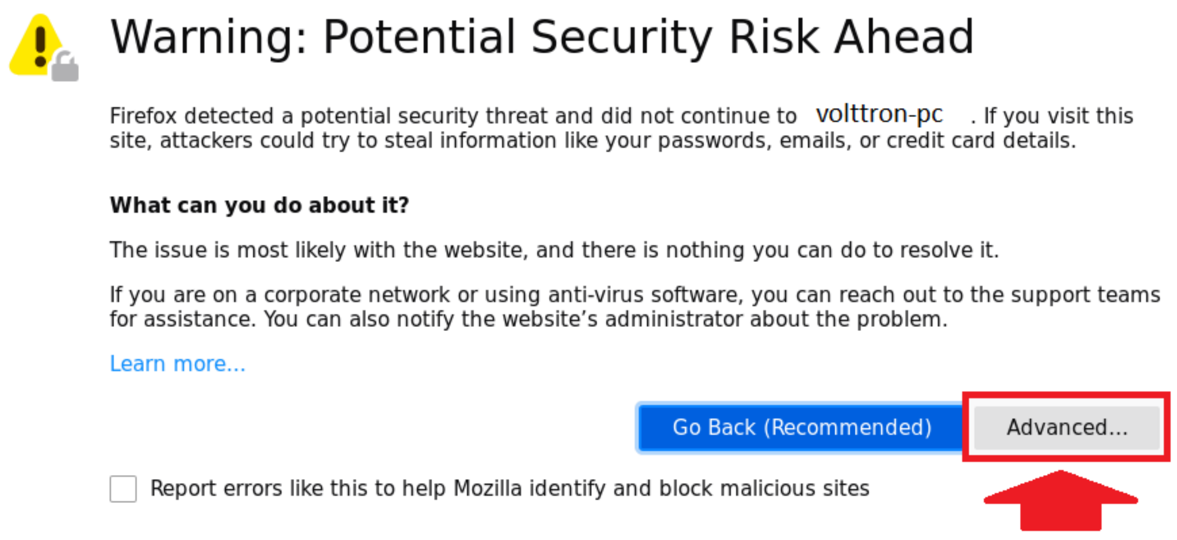
Note: In each of the above examples one could use * suffix to match more than one agent.

VOLTTRON Admin

The admin page is used to set the master username and password for both admin page and VOLTTRON Central page. Admin page can then be used to manage RMQ and ZMQ certificates and credentials.

Open a web browser and navigate to https://volttron-pc:8443/admin/login.html

There may be a message warning about a potential security risk. Check to see if the certificate that was created in vcfg is being used. The process below is for firefox.



Warning: Potential Security Risk Ahead

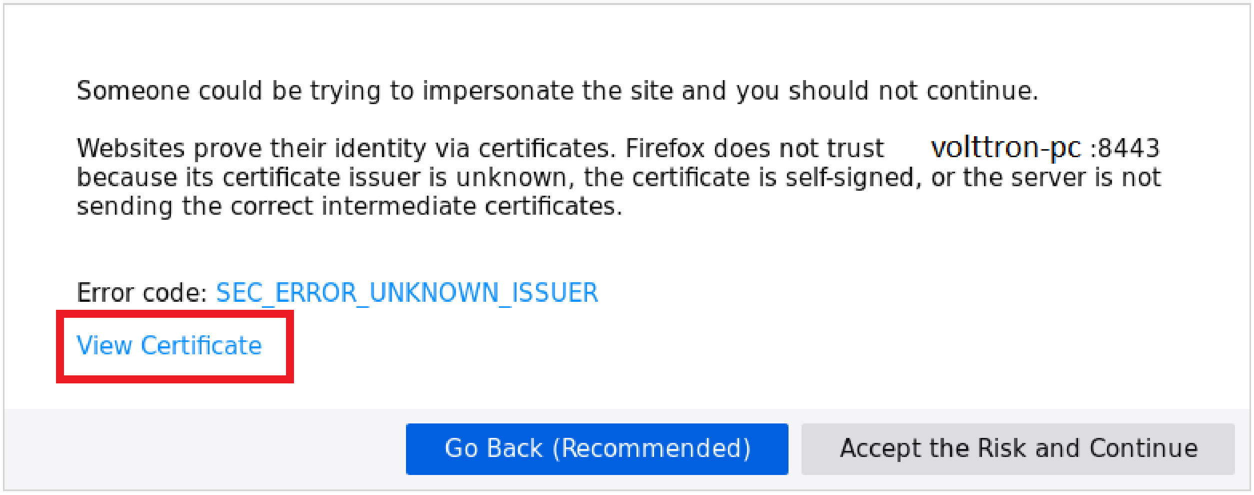
Firefox detected a potential security threat and did not continue to volttron-pc . If you visit this site, attackers could try to steal information like your passwords, emails, or credit card details.

What can you do about it?

The issue is most likely with the website, and there is nothing you can do to resolve it.

If you are on a corporate network or using anti-virus software, you can reach out to the support teams for assistance. You can also notify the website's administrator about the problem.

Learn more...
Go Back (Recommended)
Advanced...
Report errors like this to help Mozilla identify and block malicious sites

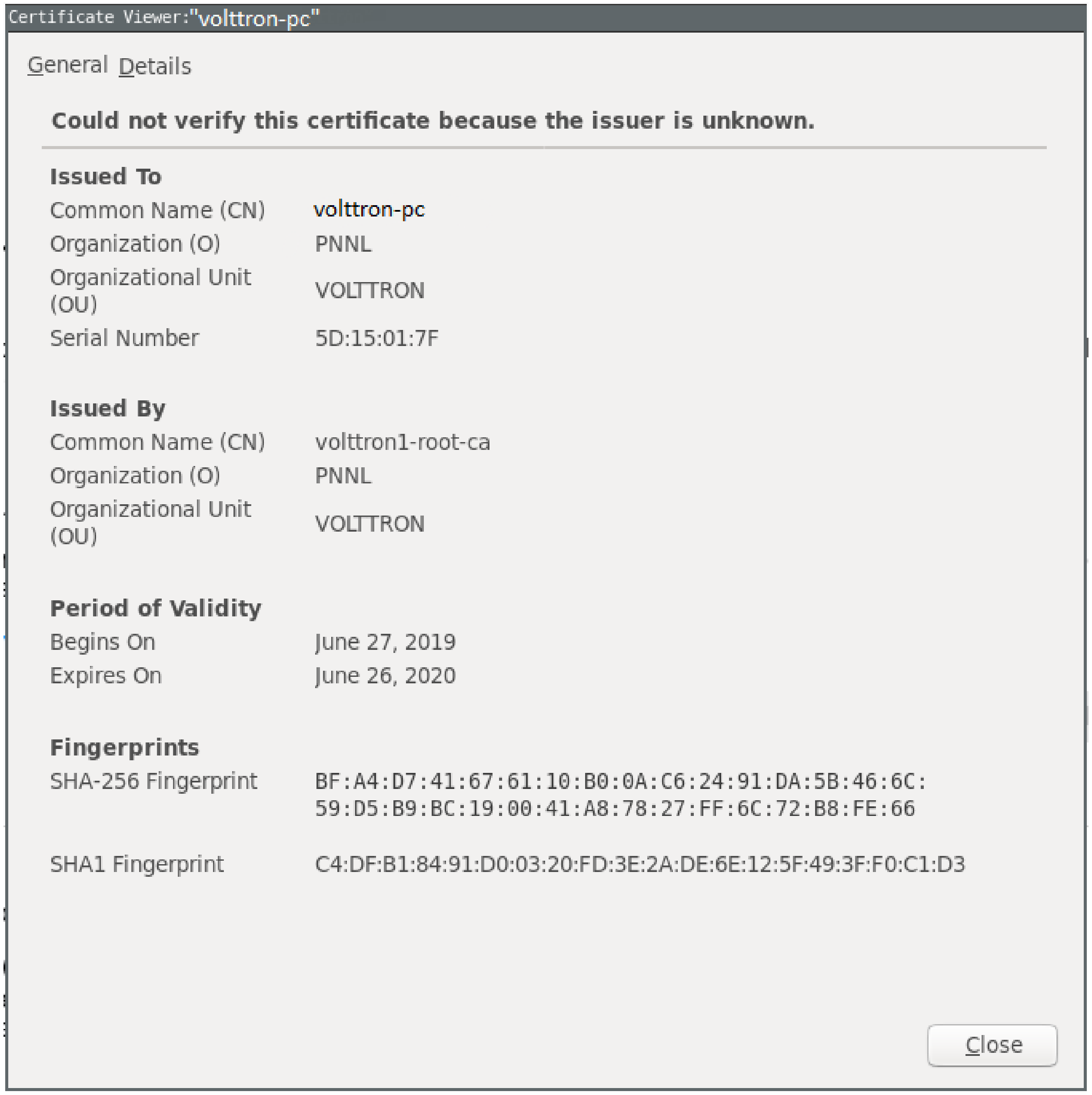


Someone could be trying to impersonate the site and you should not continue.

Websites prove their identity via certificates. Firefox does not trust volttron-pc :8443 because its certificate issuer is unknown, the certificate is self-signed, or the server is not sending the correct intermediate certificates.

Error code: SEC_ERROR_UNKNOWN_ISSUER

View Certificate


Certificate Viewer:"volttron-pc"

General Details

Could not verify this certificate because the issuer is unknown.

Issued To

Common Name (CN)	volttron-pc
Organization (O)	PNNL
Organizational Unit (OU)	VOLTTRON
Serial Number	5D:15:01:7F

Issued By

Common Name (CN)	volttron1-root-ca
Organization (O)	PNNL
Organizational Unit (OU)	VOLTTRON

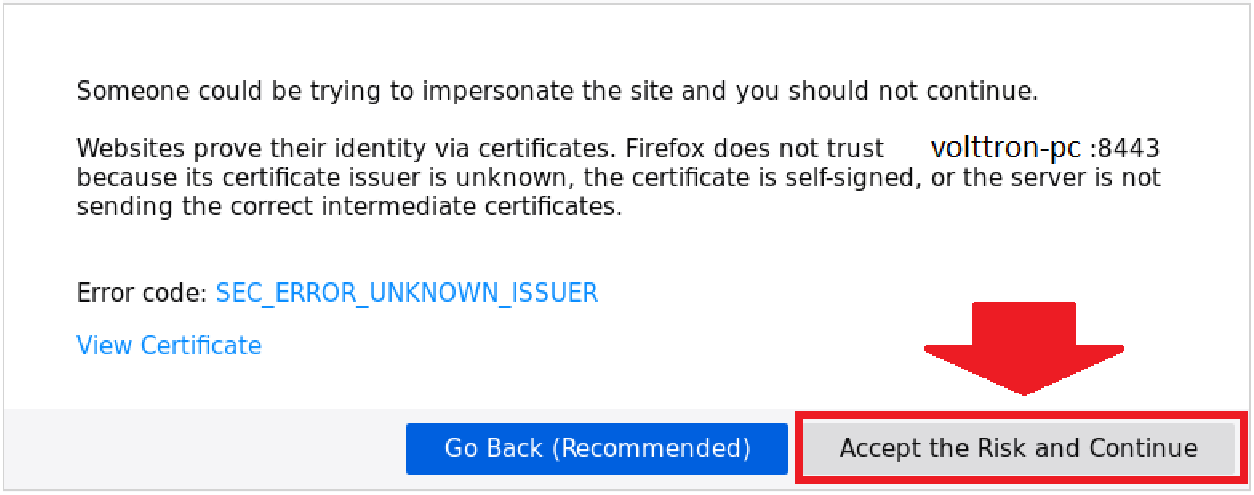
Period of Validity

Begins On	June 27, 2019
Expires On	June 26, 2020

Fingerprints

SHA-256 Fingerprint	BF:A4:D7:41:67:61:10:B0:0A:C6:24:91:DA:5B:46:6C:59:D5:B9:BC:19:00:41:A8:78:27:FF:6C:72:B8:FE:66
SHA1 Fingerprint	C4:DF:B1:84:91:D0:03:20:FD:3E:2A:DE:6E:12:5F:49:3F:F0:C1:D3

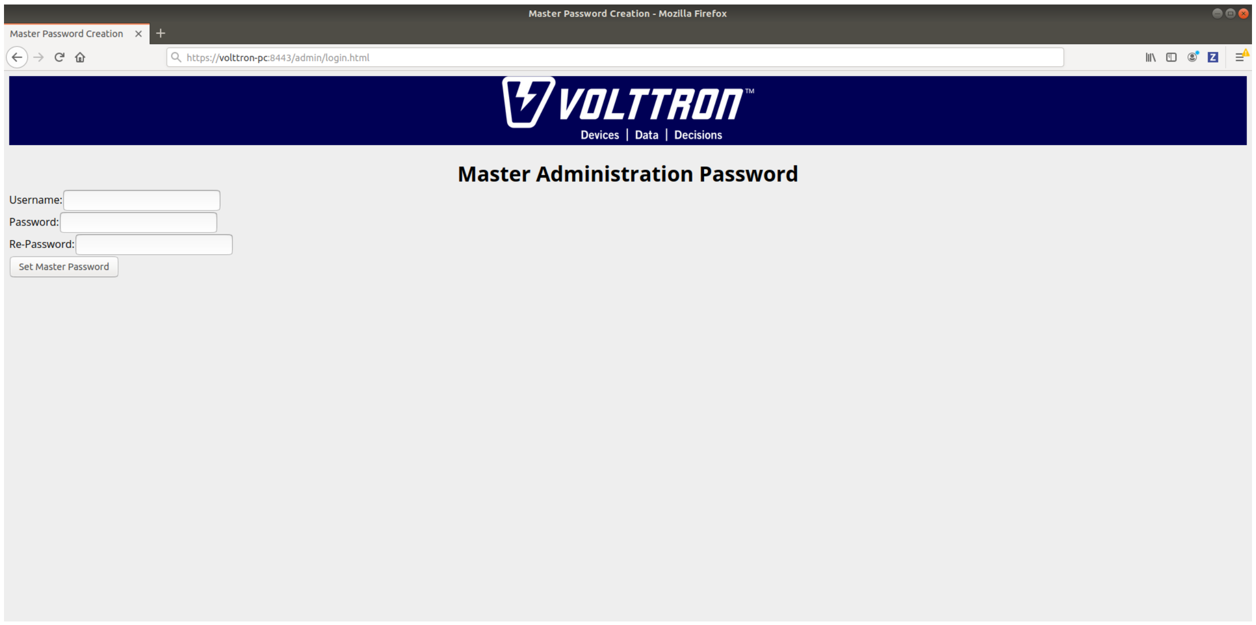
Close


Someone could be trying to impersonate the site and you should not continue.

Websites prove their identity via certificates. Firefox does not trust volttron-pc :8443 because its certificate issuer is unknown, the certificate is self-signed, or the server is not sending the correct intermediate certificates.

Error code: SEC_ERROR_UNKNOWN_ISSUER

When the admin page is accessed for the first time, the user will be prompted to set up a master username and password.


Master Password Creation - Mozilla Firefox
Master Password Creation
https://volttron-pc:8443/admin/login.html
VOLTTRON
Devices | Data | Decisions

Master Administration Password

Username:
Password:
Re-Password:

Open your browser to the web address that you specified for the VOLTTRON Central agent that you configured for the first instance. In the above examples, the configuration file would be located at *~/.volttron1/config* and the VOLTTRON Central address would be defined in the "volttron-central-address" field. The VOLTTRON Central address takes the pattern: *https://<localhost>:8443/vc/index.html*, where localhost is the hostname of your machine. In the above examples, our hostname is *volttron-pc*; thus our VC interface would be *https://volttron-pc:8443/vc/index.html*.

You will need to provide the username and password set earlier through admin web page.

Stopping the Demo

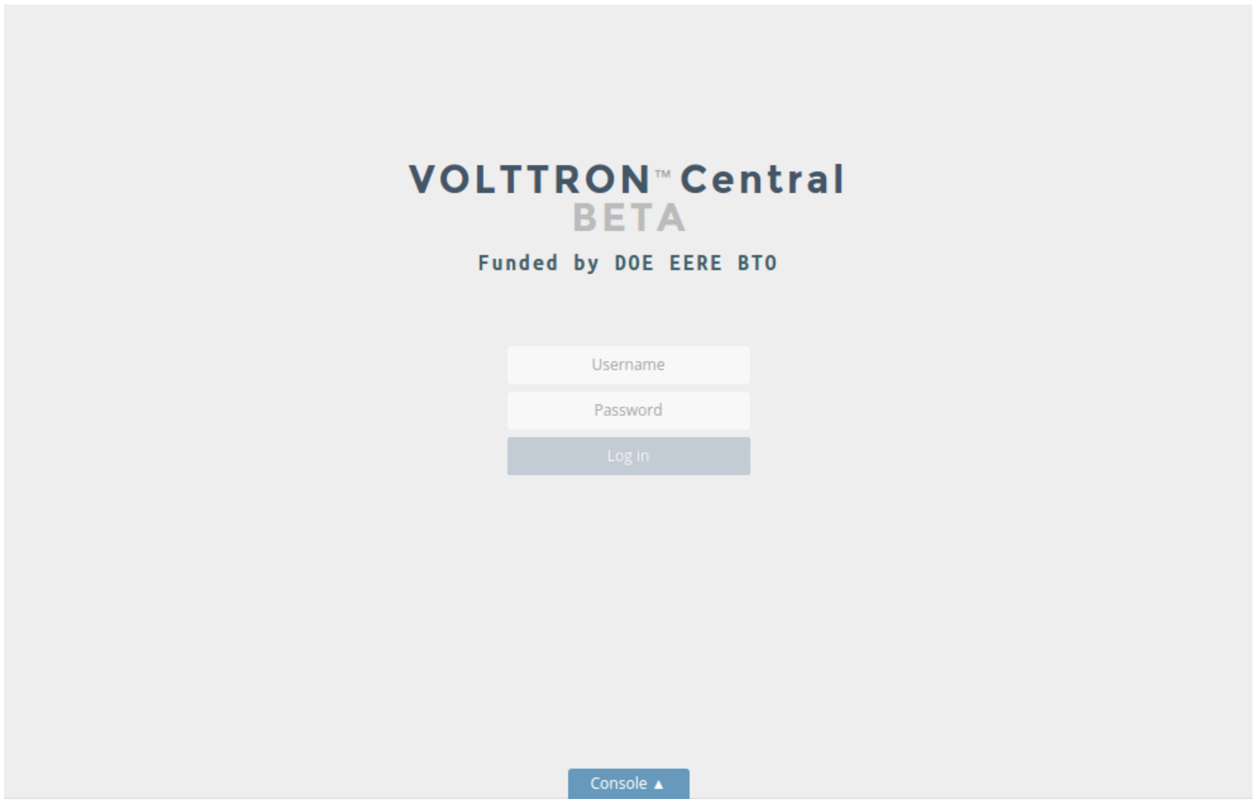
Once you have completed your walk through of the different elements of the VOLTTRON Central demo you can stop the demos by executing the following command in each terminal window.

```
$ ./stop-volttron
```

Once the demo is complete you may wish to see the *VOLTTRON Central Management Agent* page for more details on how to configure the agent for your specific use case.

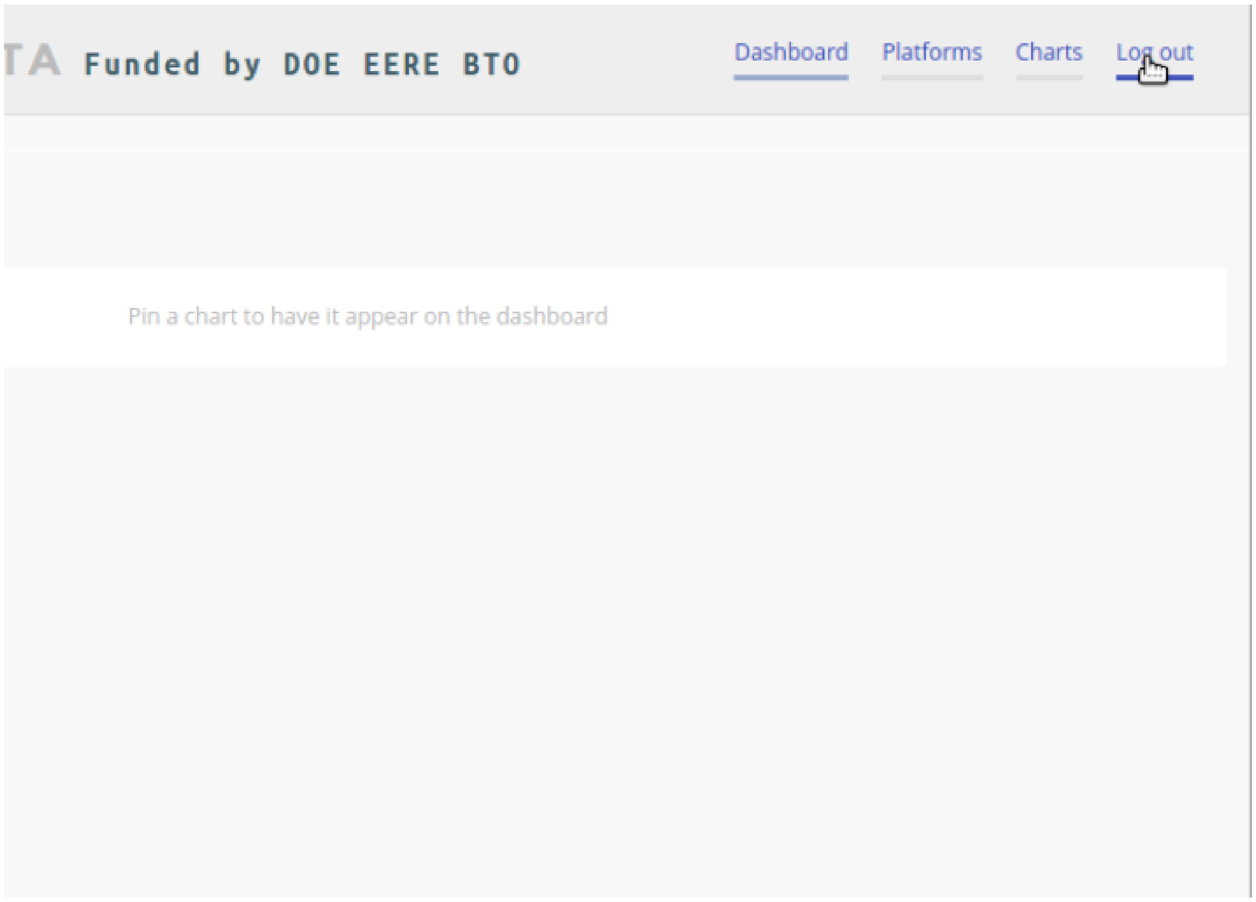
Log In

To log in to VOLTTRON Central, open a browser and login to the Volttron web interface, which takes the form *https://localhost:8443/vc/index.html* where localhost is the hostname of your machine. In the above example, we open the following URL in which our localhost is "volttron-pc": https://volttron-pc:8443/vc/index.html and enter the user name and password on the login screen.



Log Out

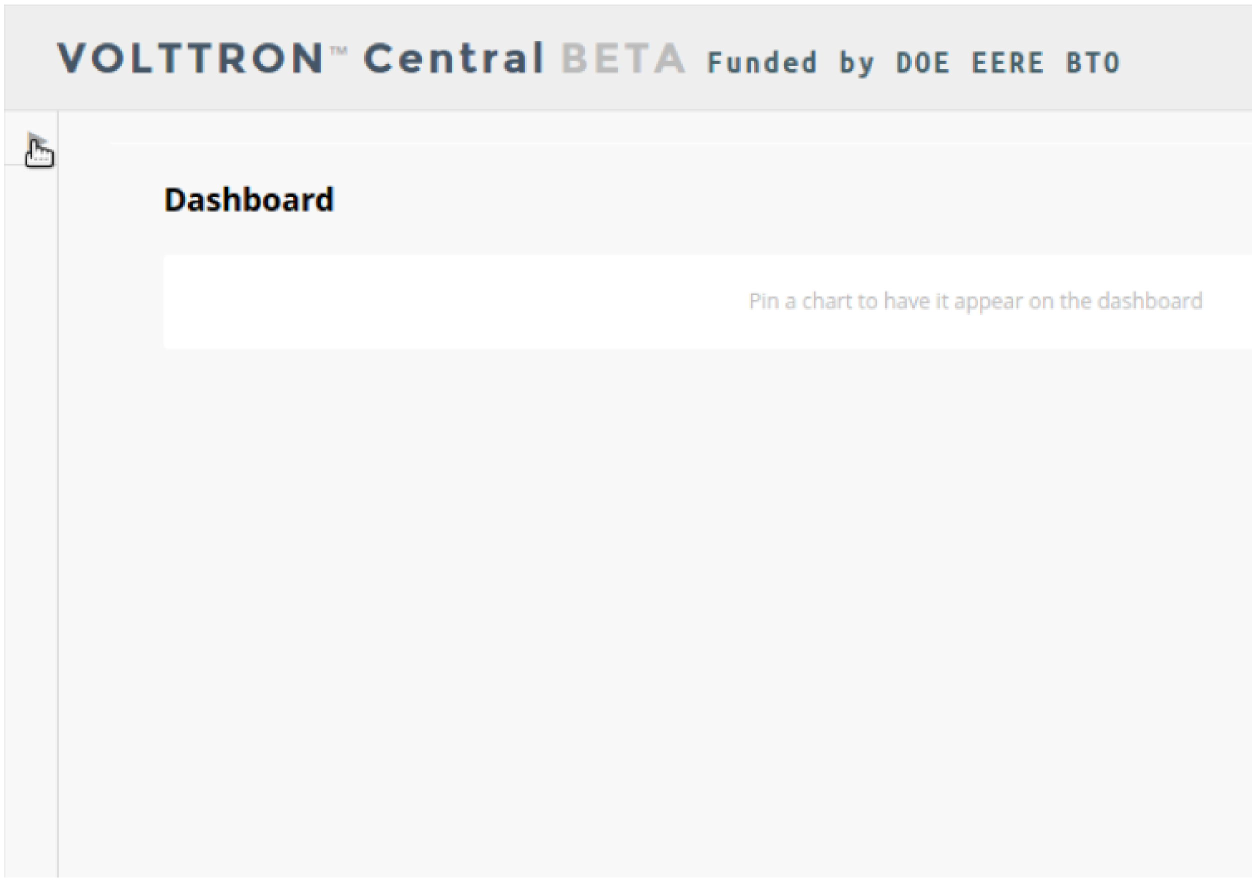
To log out of VOLTTRON Central, click the link at the top right of the screen.



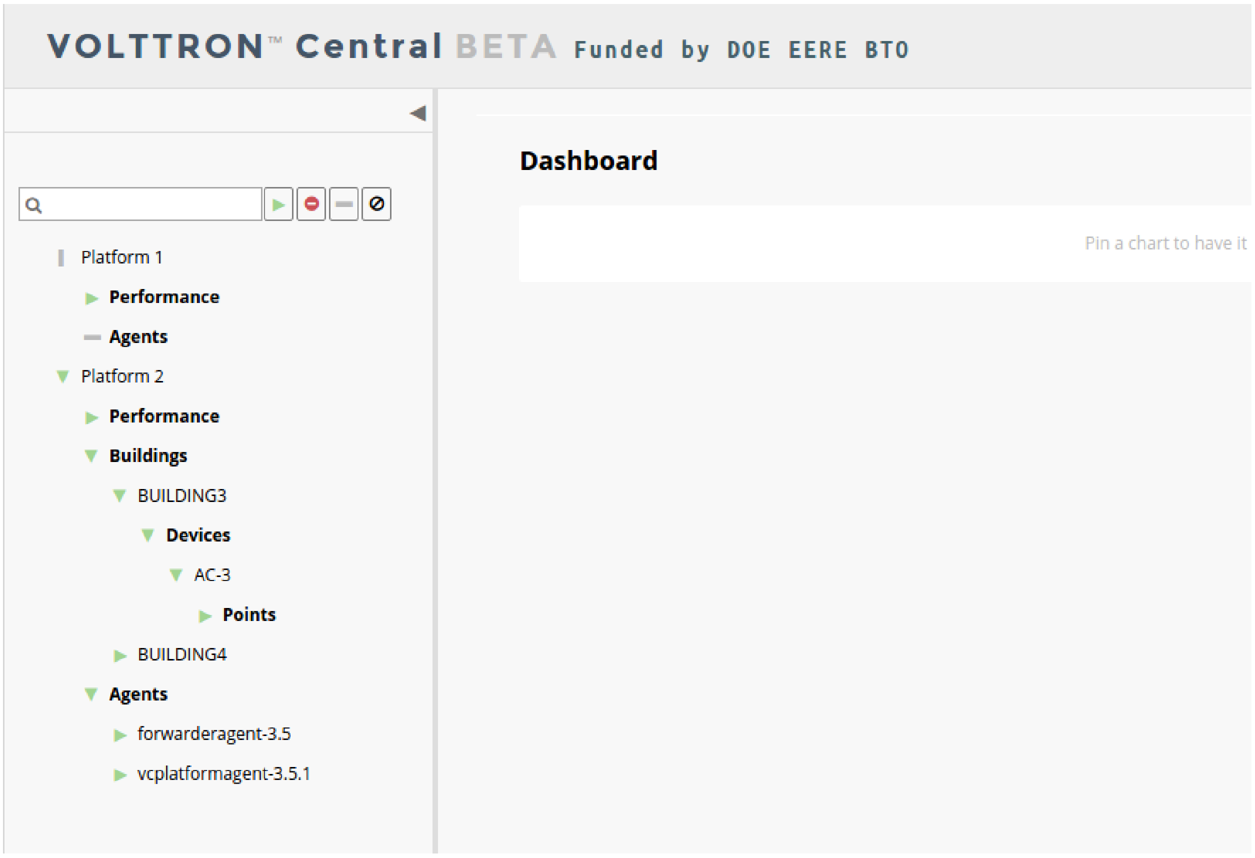
Pin a chart to have it appear on the dashboard

Platforms Tree

The side panel on the left of the screen can be extended to reveal the tree view of registered platforms.



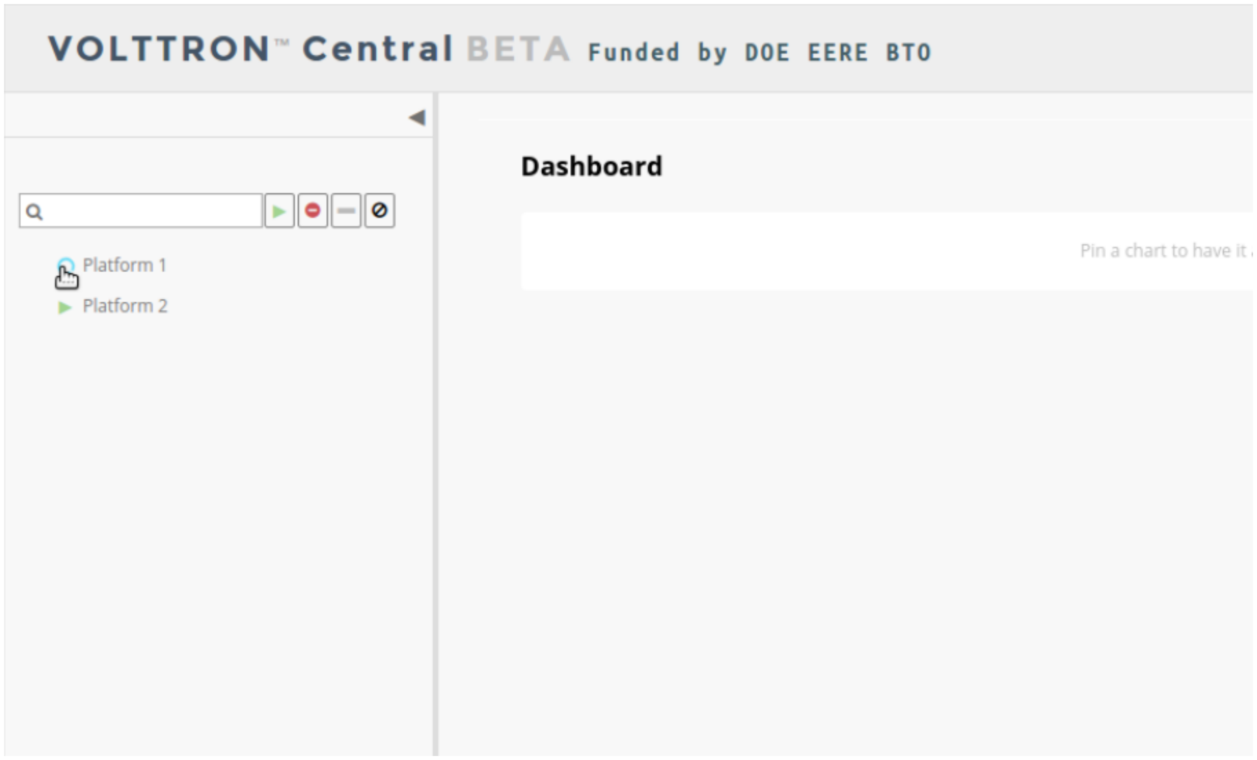
Pin a chart to have it appear on the dashboard



Top-level nodes in the tree are platforms. Platforms can be expanded in the tree to reveal installed agents, devices on buildings, and performance statistics about the platform instances.

Loading the Tree

The initial state of the tree is not loaded. The first time a top-level node is expanded is when the items for that platform are loaded.

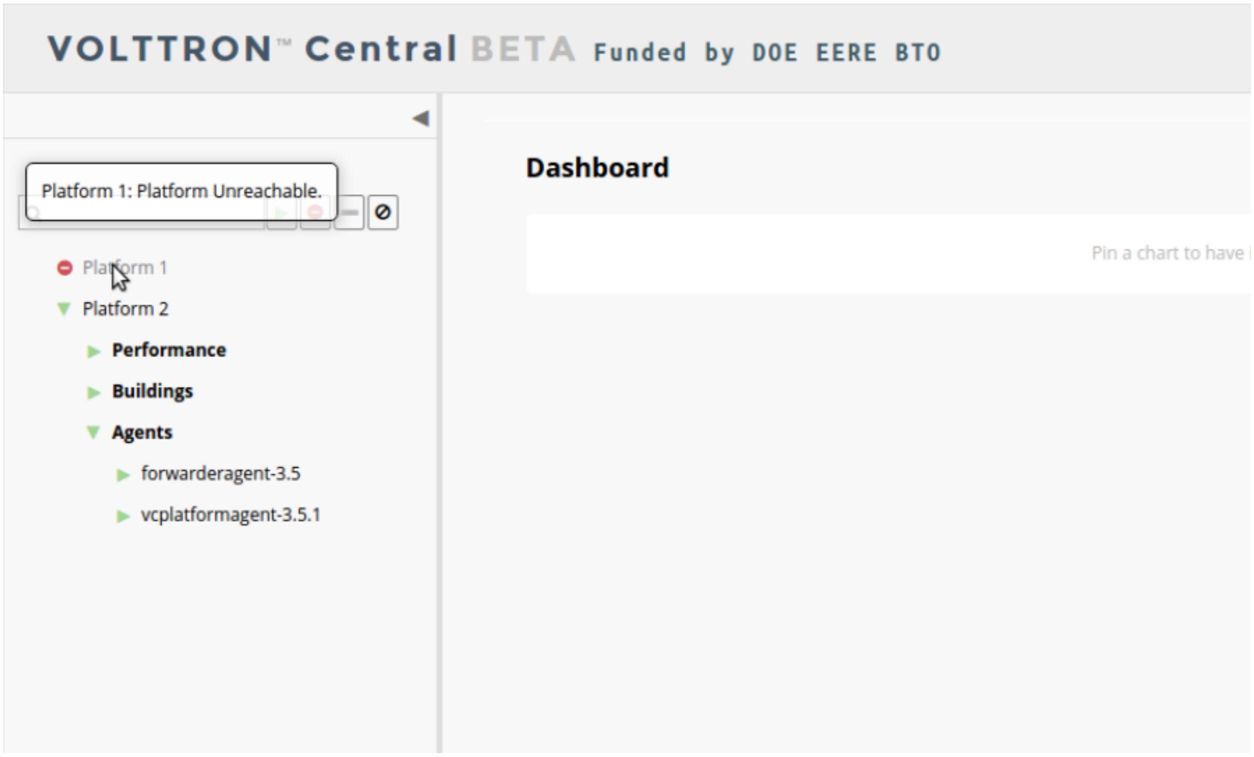


After a platform has been loaded in the tree, all the items under a node can be quickly expanded by double-clicking on the node.

Health Status

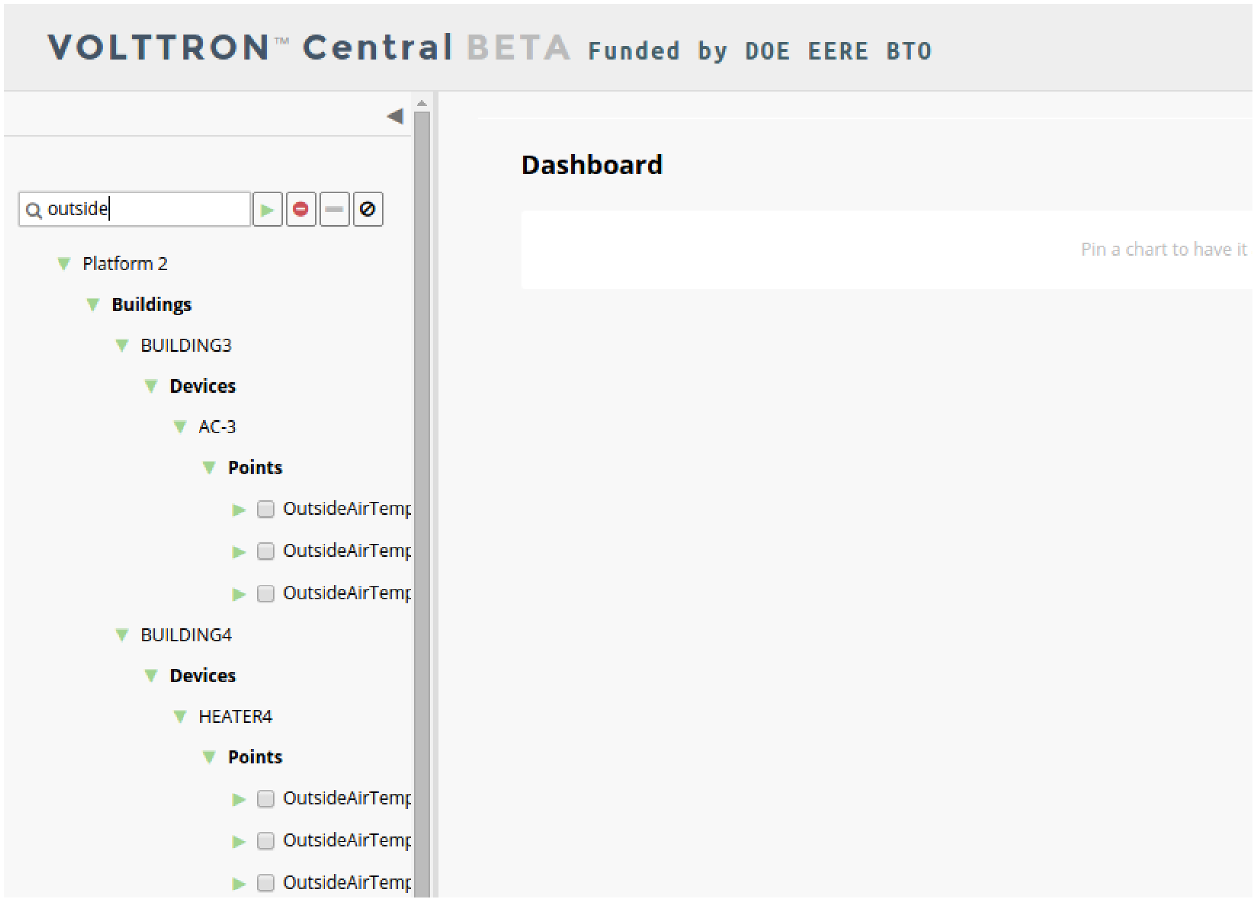
The health status of an item in the tree is indicated by the color and shape next to it. A green triangle means healthy, a red circle means there's a problem, and a gray rectangle means the status can't be determined.

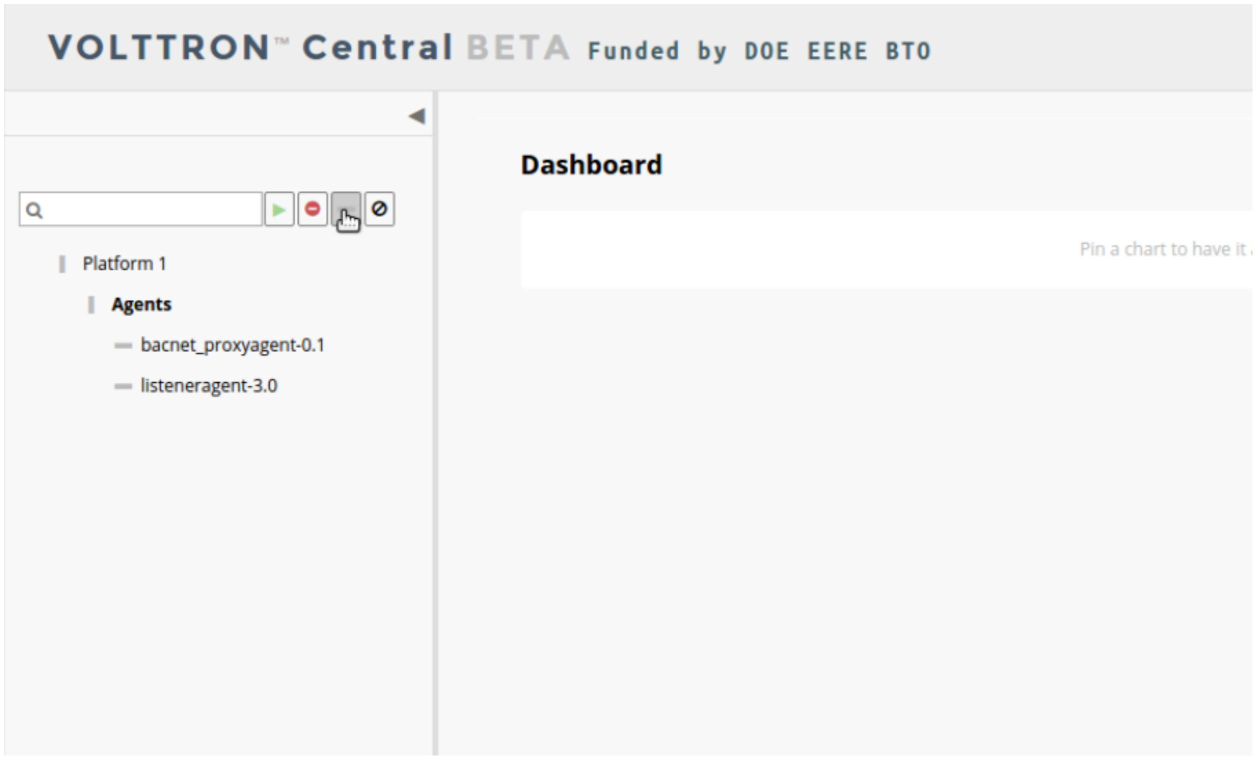
Information about the health status also may be found by hovering the cursor over the item.



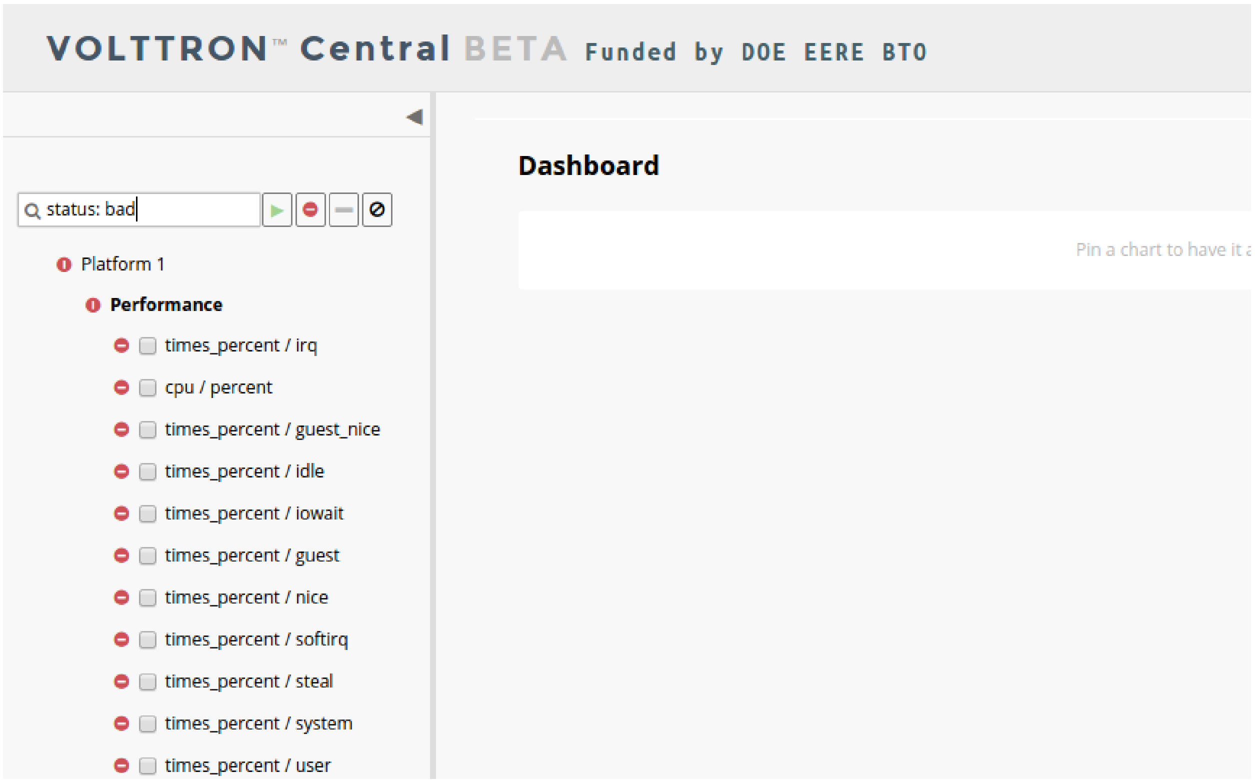
Filter the Tree

The tree can be filtered by typing in the search field at the top or clicking on a status button next to the search field.



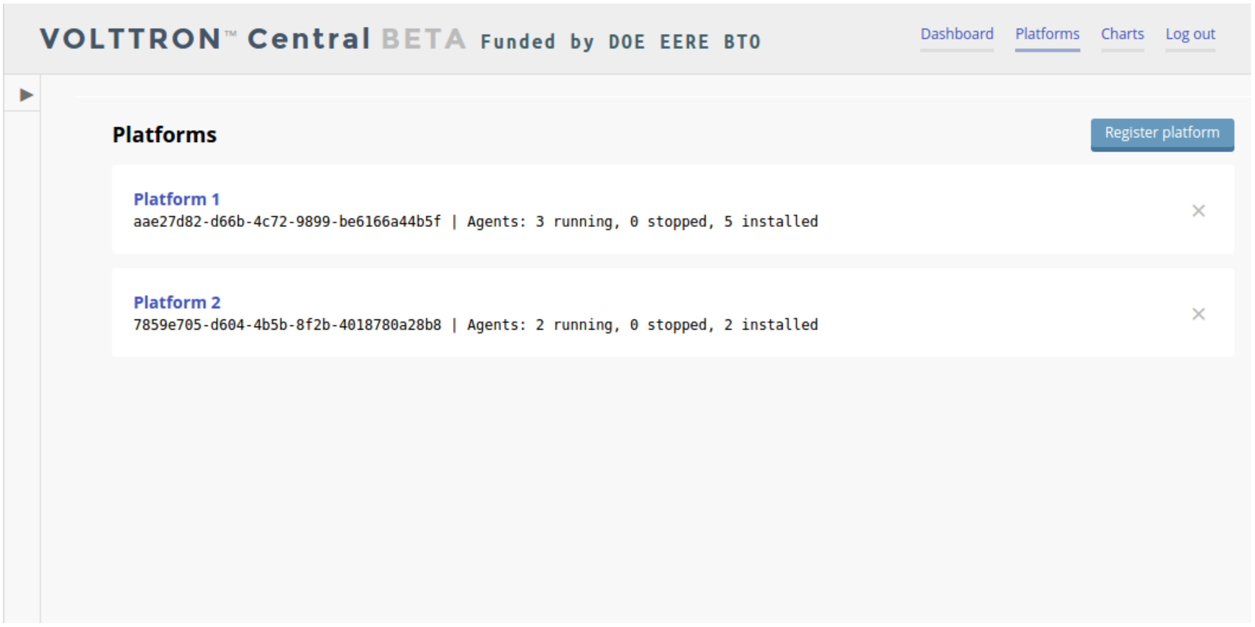


Meta terms such as “status” can also be used as filter keys. Type the keyword “status” followed by a colon, and then the word “good,” “bad,” or “unknown.”



Platforms Screen

This screen lists the registered VOLTTRON platforms and allows new platforms to be registered by clicking the Register Platform button. Each platform is listed with its unique ID and the number and status of its agents. The platform’s name is a link that can be clicked on to go to the platform management view.

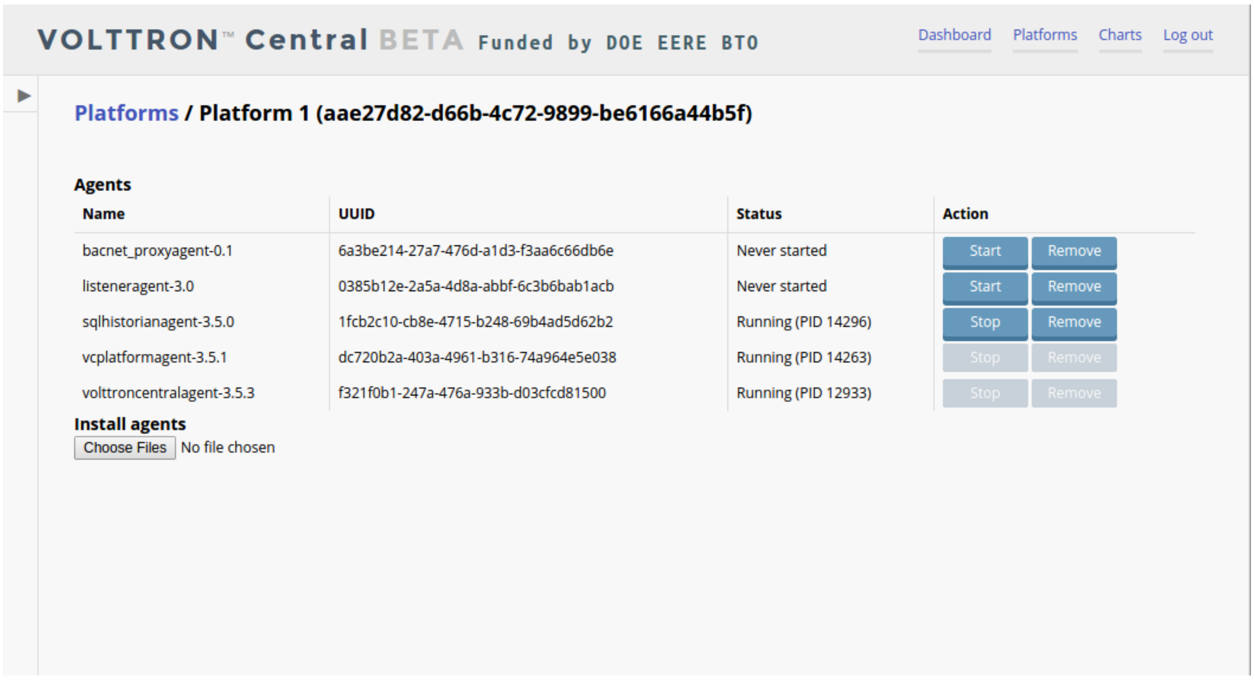


Platform 1
aae27d82-d66b-4c72-9899-be6166a44b5f | Agents: 3 running, 0 stopped, 5 installed

Platform 2
7859e705-d604-4b5b-8f2b-4018780a28b8 | Agents: 2 running, 0 stopped, 2 installed

Platform View

From the platforms screen, click on the name link of a platform to manage it. Managing a platform includes installing, starting, stopping, and removing its agents.



Platforms / Platform 1 (aae27d82-d66b-4c72-9899-be6166a44b5f)

Agents

Name	UUID	Status	Action
bacnet_proxyagent-0.1	6a3be214-27a7-476d-a1d3-f3aa6c66db6e	Never started	Start Remove
listeneragent-3.0	0385b12e-2a5a-4d8a-abbf-6c3b6bab1acb	Never started	Start Remove
sqlhistorianagent-3.5.0	1fcb2c10-cb8e-4715-b248-69b4ad5d62b2	Running (PID 14296)	Stop Remove
vcplatformagent-3.5.1	dc720b2a-403a-4961-b316-74a964e5e038	Running (PID 14263)	Stop Remove
volttroncentralagent-3.5.3	f321f0b1-247a-476a-933b-d03cfcd81500	Running (PID 12933)	Stop Remove

Install agents

Choose Files No file chosen

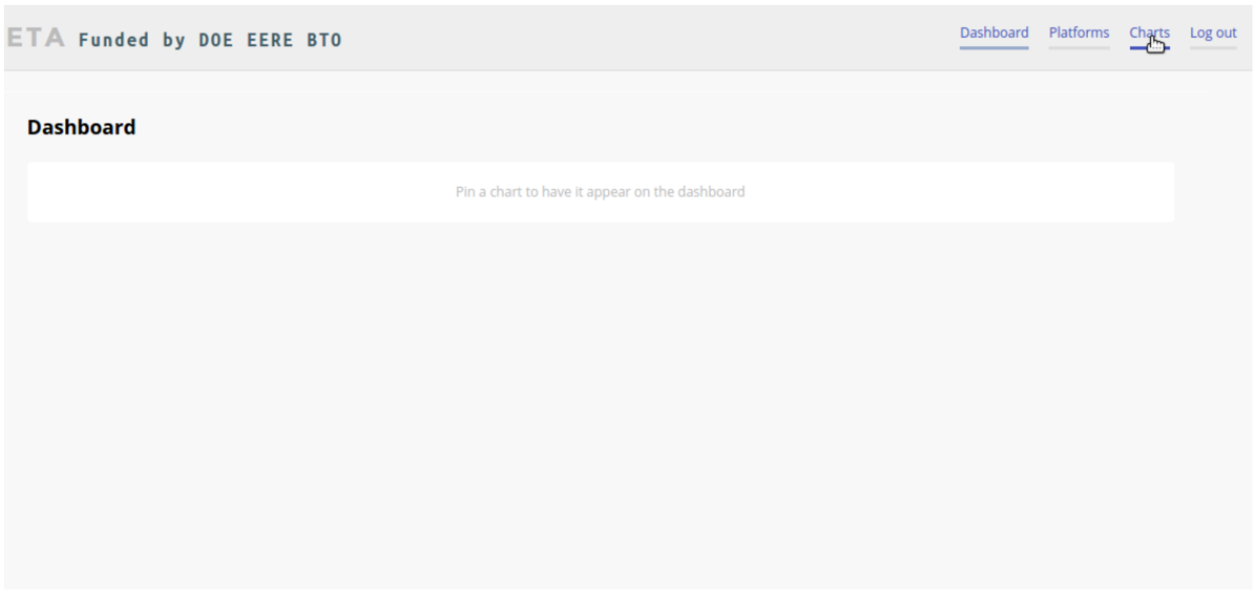
To install a new agent, all you need is the agent's wheel file. Click on the button and choose the file to upload it and install the agent.

To start, stop, or remove an agent, click on the button next to the agent in the list. Buttons may be disabled if the user lacks the correct permission to perform the action or if the action can't be performed on a specific type of agent. For instance, platform agents and VOLTTRON Central agents can't be removed or stopped, but they can be restarted if they've been interrupted.

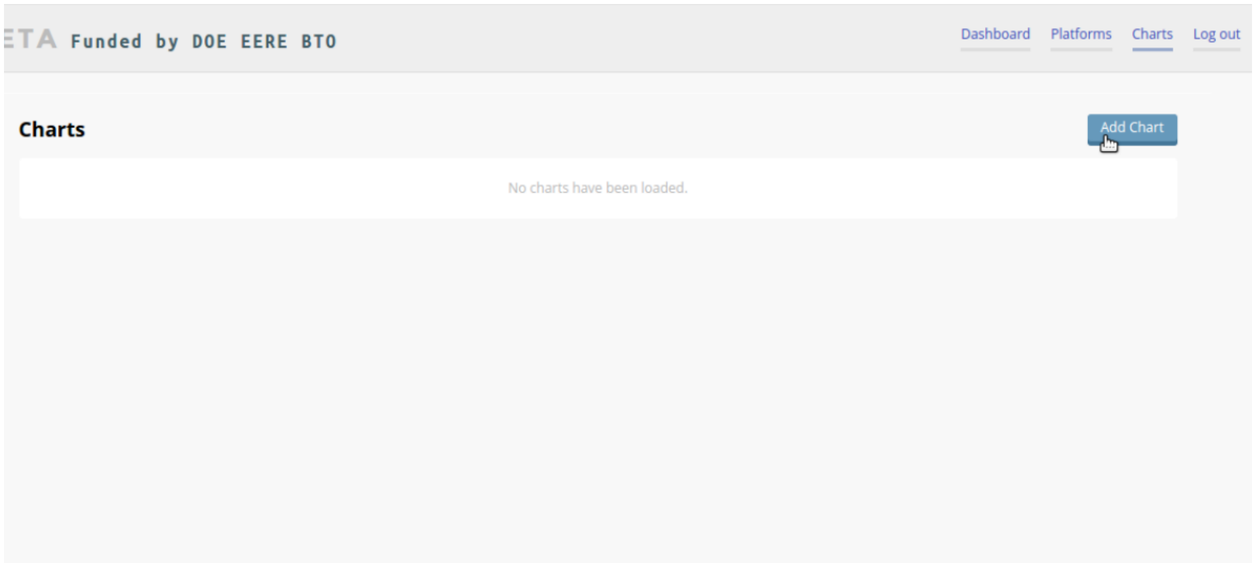
Add Charts

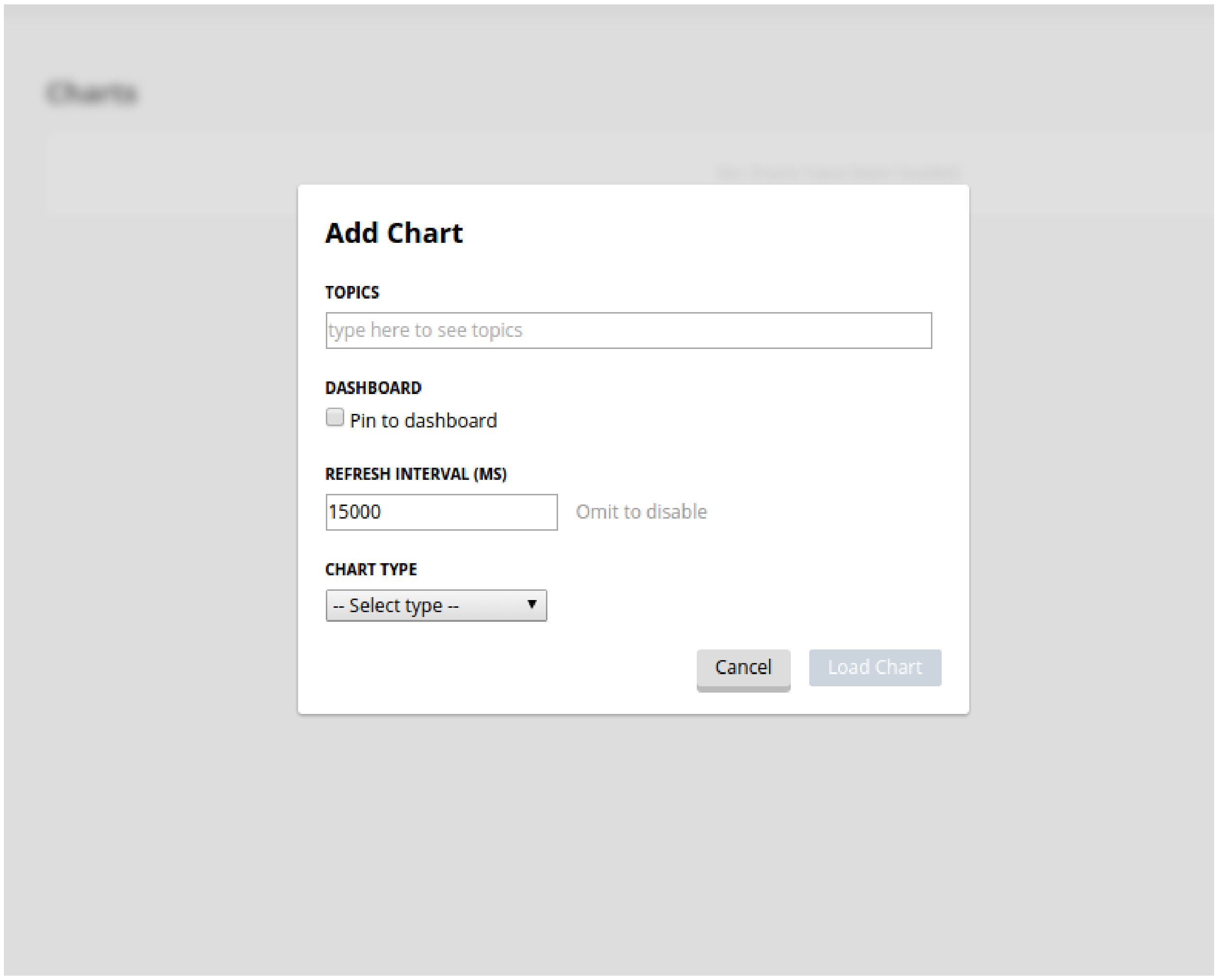
Performance statistics and device points can be added to charts either from the Charts page or from the platforms tree in the side panel.

Click the Charts link at the top-right corner of the screen to go to the Charts page.


ETA Funded by DOE EERE BTO
Dashboard
Platforms
Charts
Log out
Dashboard
Pin a chart to have it appear on the dashboard

From the Charts page, click the Add Chart button to open the Add Chart window.




Add Chart

TOPICS

type here to see topics

DASHBOARD

Pin to dashboard

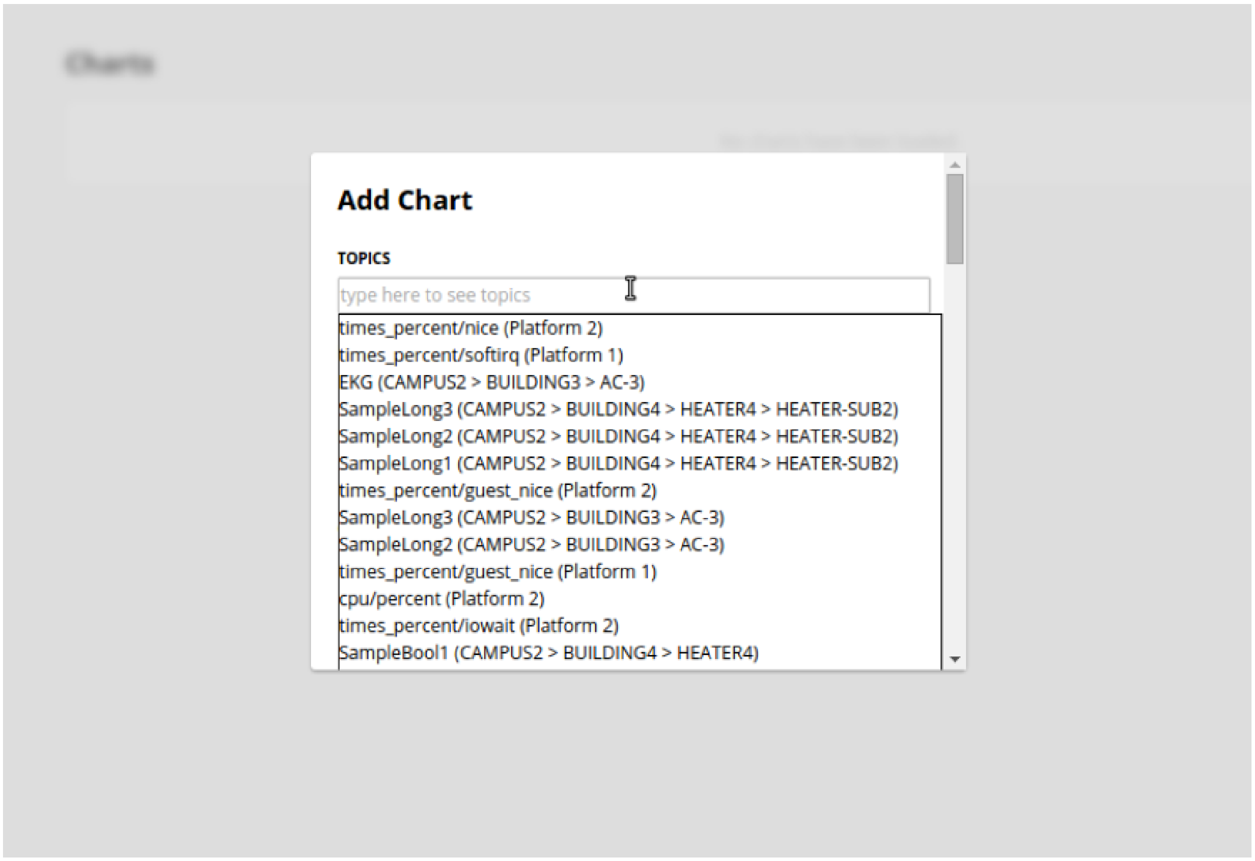
REFRESH INTERVAL (MS)

15000 Omit to disable

CHART TYPE

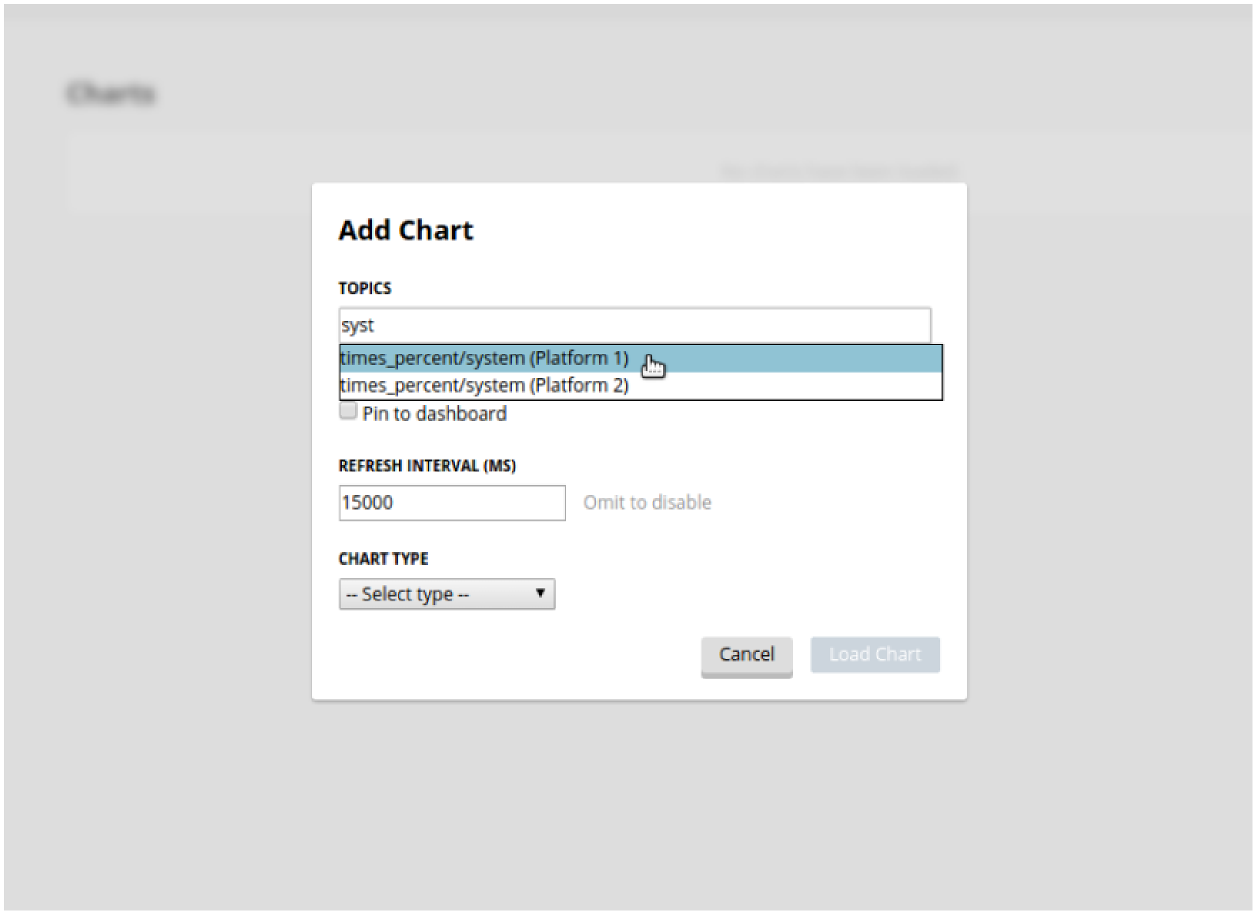
-- Select type --

Click in the topics input field to make the list of available chart topics appear.



times_percent/nice (Platform 2)
times_percent/softirq (Platform 1)
EKG (CAMPUS2 > BUILDING3 > AC-3)
SampleLong3 (CAMPUS2 > BUILDING4 > HEATER4 > HEATER-SUB2)
SampleLong2 (CAMPUS2 > BUILDING4 > HEATER4 > HEATER-SUB2)
SampleLong1 (CAMPUS2 > BUILDING4 > HEATER4 > HEATER-SUB2)
times_percent/guest_nice (Platform 2)
SampleLong3 (CAMPUS2 > BUILDING3 > AC-3)
SampleLong2 (CAMPUS2 > BUILDING3 > AC-3)
times_percent/guest_nice (Platform 1)
cpu/percent (Platform 2)
times_percent/iowait (Platform 2)
SampleBool1 (CAMPUS2 > BUILDING4 > HEATER4)

Scroll and select from the list, or type in the field to filter the list, and then select.



syst

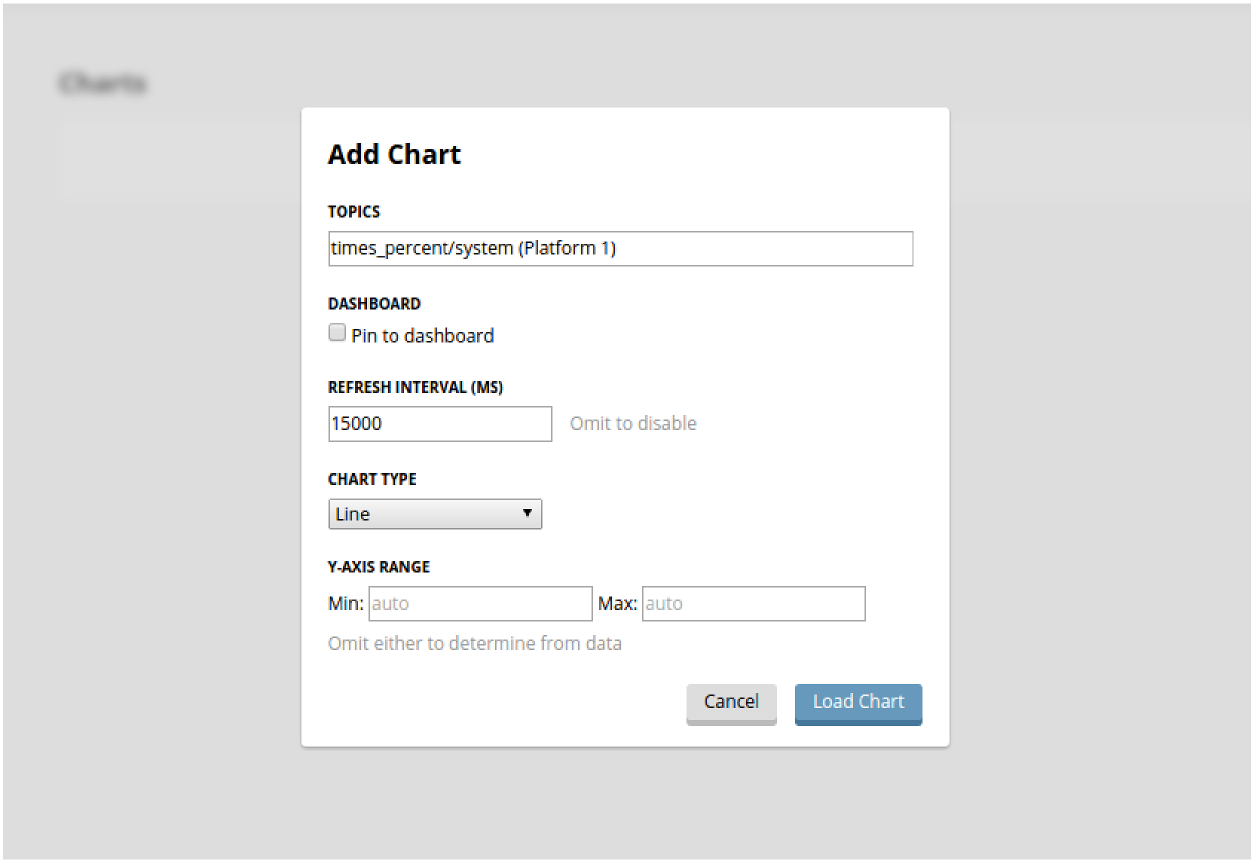
times_percent/system (Platform 1)

times_percent/system (Platform 2)

15000

-- Select type --

Select a chart type and click the Load Chart button to close the window and load the chart.


Add Chart

TOPICS: times_percent/system (Platform 1)

DASHBOARD: Pin to dashboard

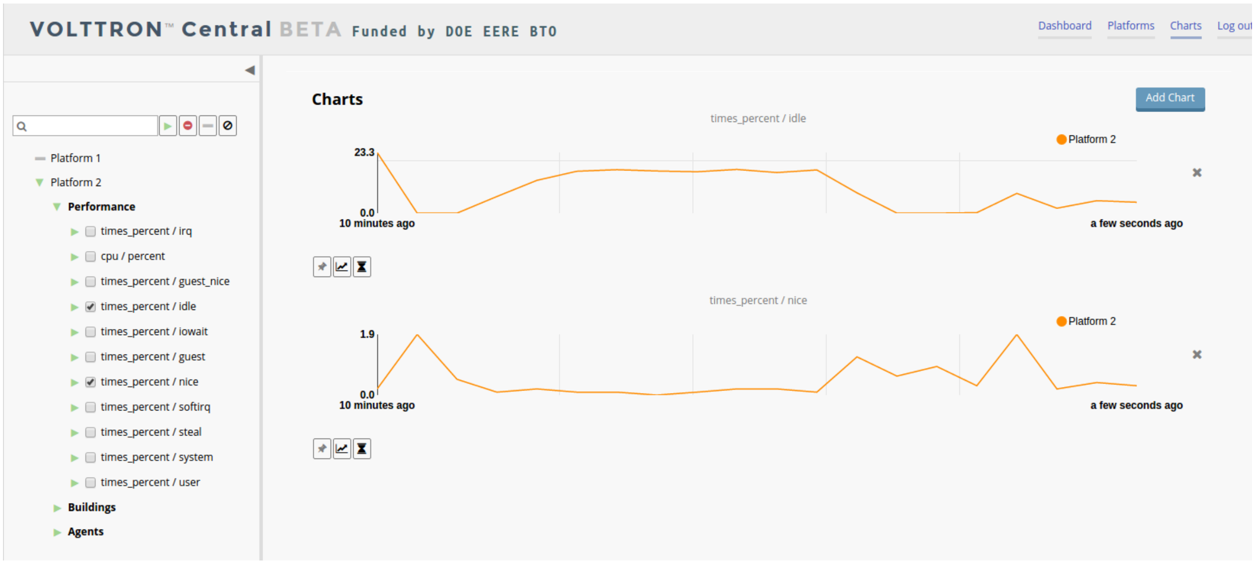
REFRESH INTERVAL (MS): 15000 Omit to disable

CHART TYPE: Line

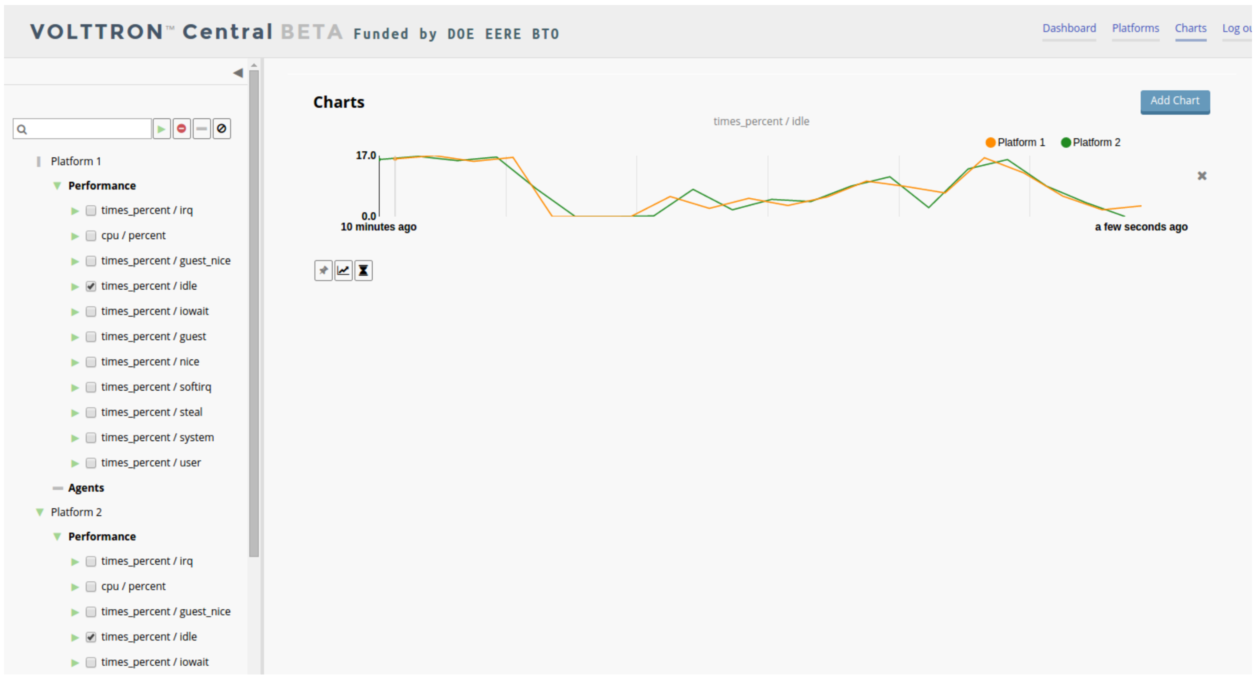
Y-AXIS RANGE: Min: auto Max: auto

Omit either to determine from data

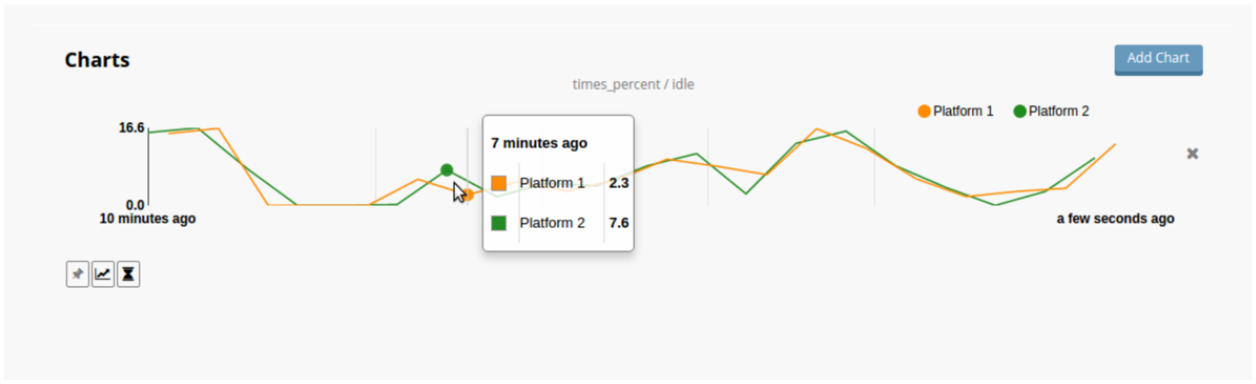
To add charts from the side panel, check boxes next to items in the tree.



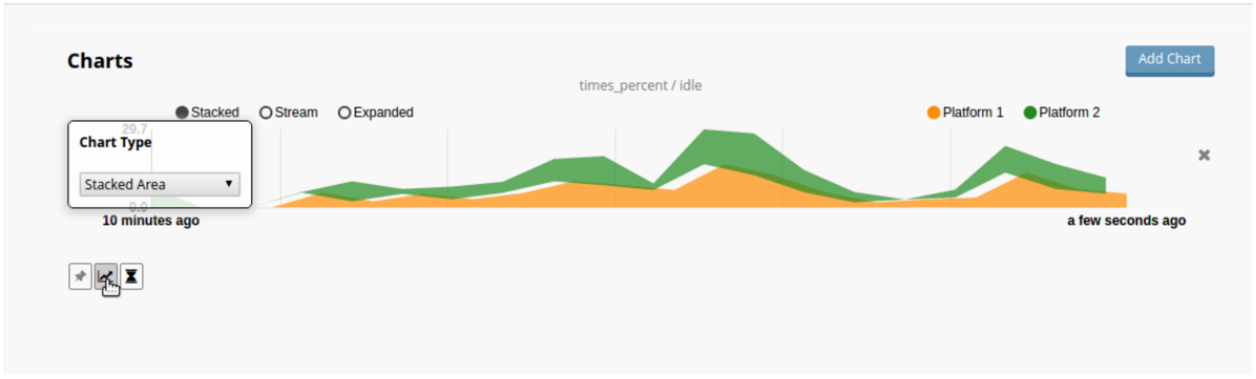
Choose points with the same name from multiple platforms or devices to plot more than one line in a chart.



Move the cursor arrow over the chart to inspect the graphs.

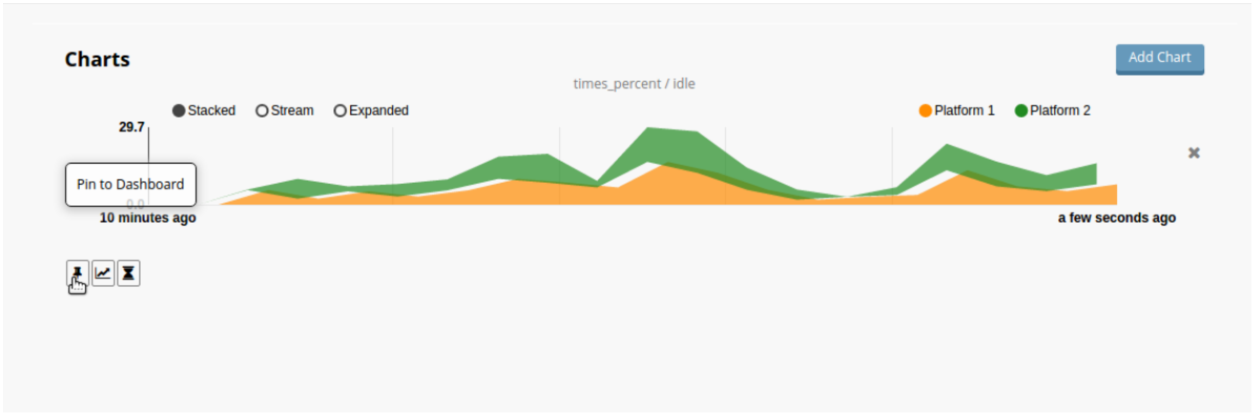


To change the chart's type, click on the Chart Type button and choose a different option.



Dashboard Charts

To pin a chart to the Dashboard, click the Pin Chart button to toggle it. When the pin image is black and upright, the chart is pinned; when the pin image is gray and diagonal, the chart is not pinned and won't appear on the Dashboard.



Charts that have been pinned to the Dashboard are saved to the database and will automatically load when the user logs in to VOLTTRON Central. Different users can save their own configurations of dashboard charts.

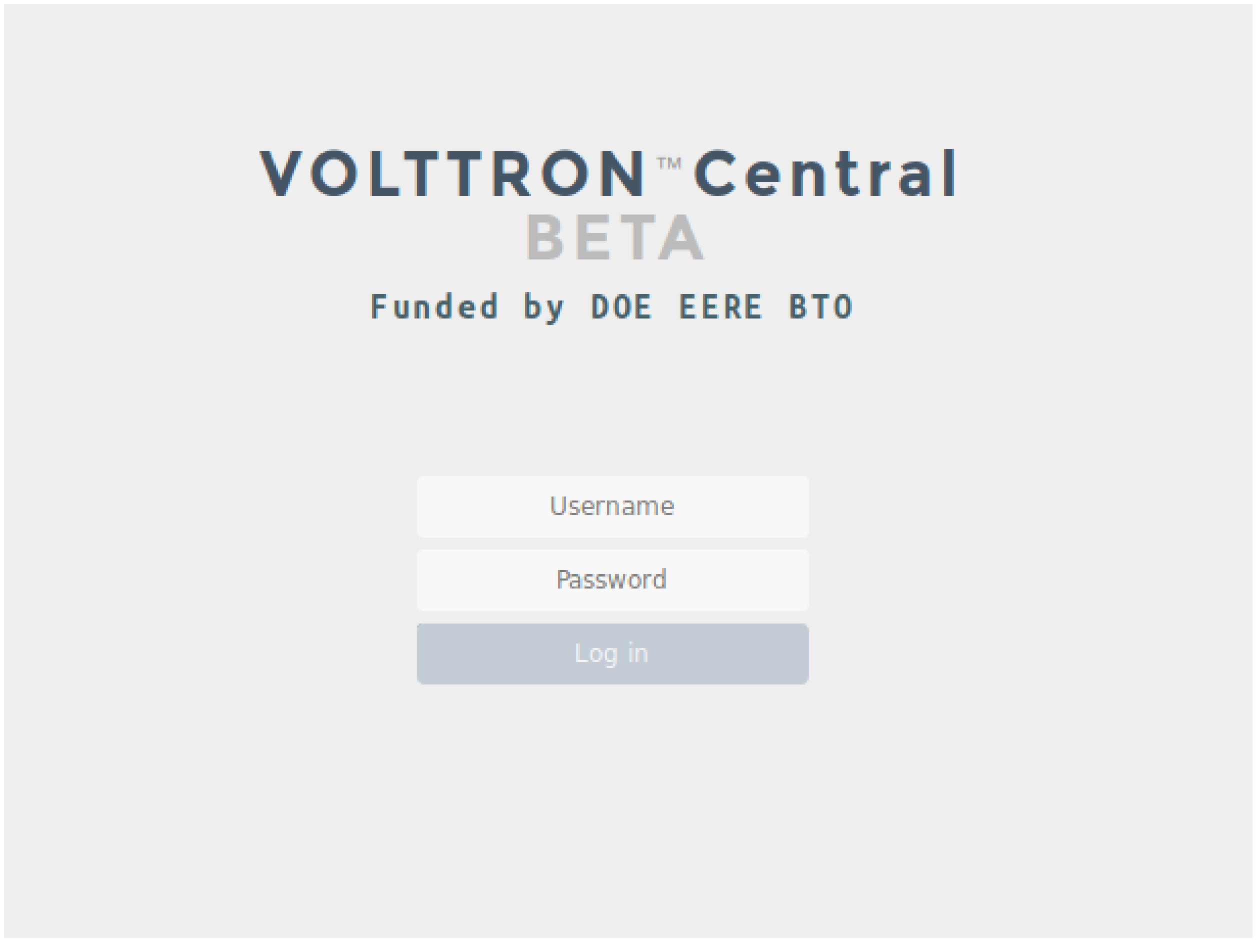
Remove Charts

To remove a chart, uncheck the box next to the item in the tree or click the X button next to the chart on the Charts page. Removing a chart removes it from the Charts page and the Dashboard.

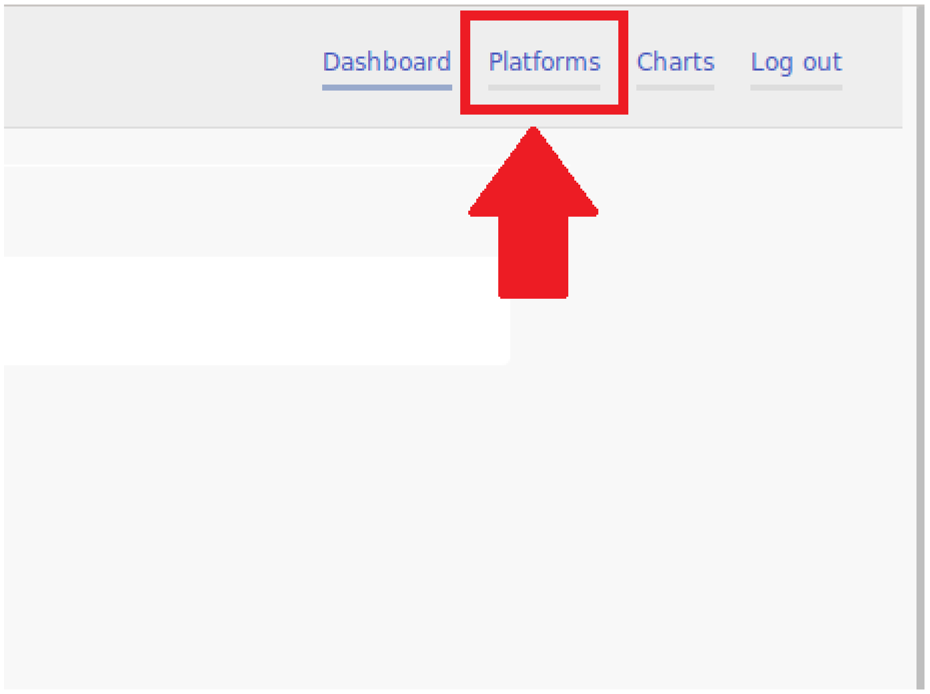
VOLTTRON Central

Navigate to https://volttron-pc:8443/vc/index.html

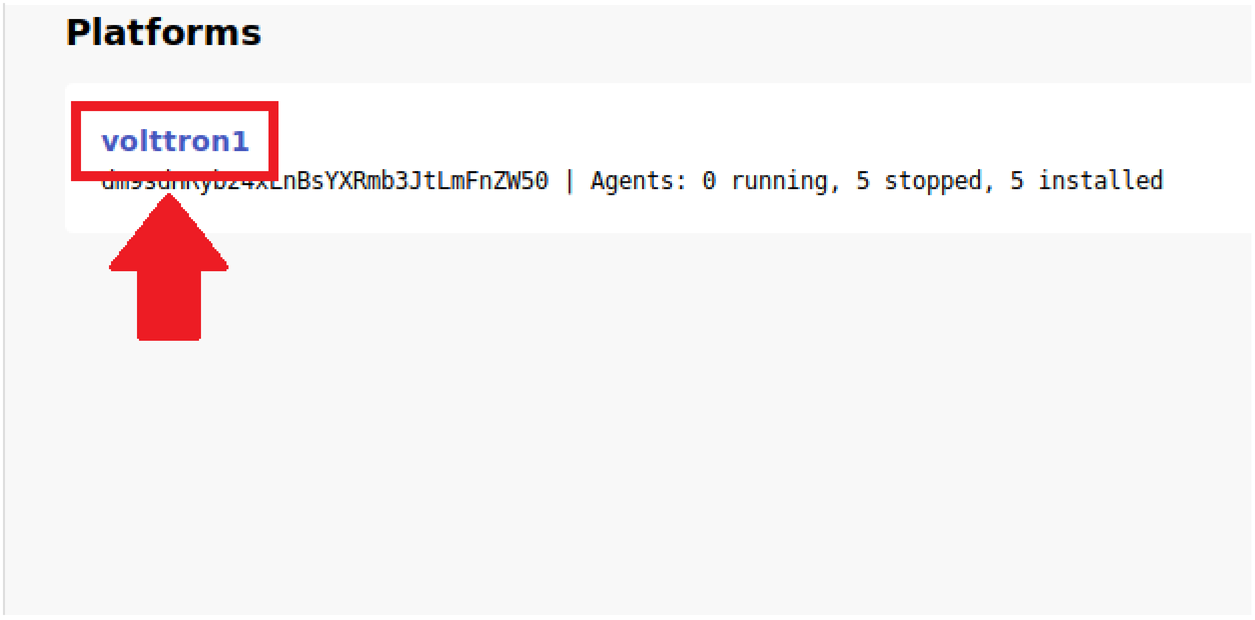
Log in using the username and password you set up on the admin web page.



Once you have logged in, click on the Platforms tab in the upper right corner of the window.



Once in the Platforms screen, click on the name of the platform.

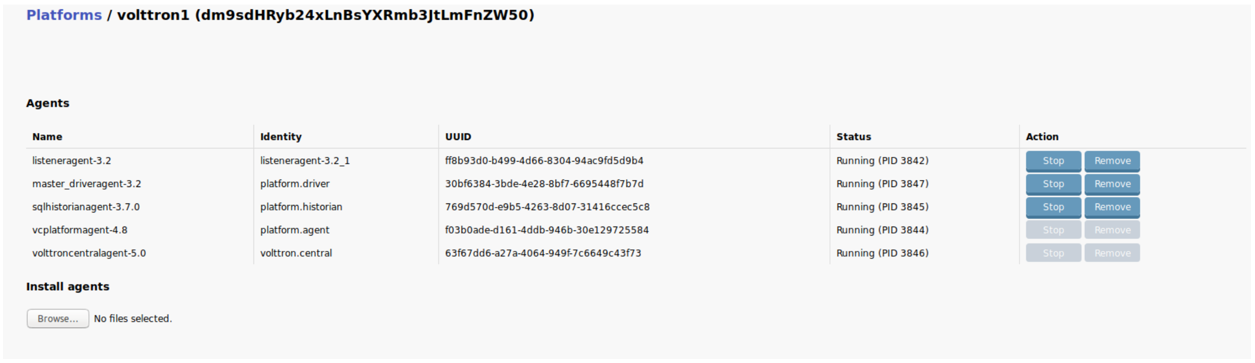


Platforms

volttron1

dm9sdHRyb24xLnBsYXRmb3JtLmFnZW50 | Agents: 0 running, 5 stopped, 5 installed

You will now see a list of agents. They should all be running.



Platforms / volttron1 (dm9sdHRyb24xLnBsYXRmb3JtLmFnZW50)

Agents

Name	Identity	UUID	Status	Action
listeneragent-3.2	listeneragent-3.2_1	ff8b93d0-b499-4d66-8304-94ac9fd5d9b4	Running (PID 3842)	Stop Remove
master_driveragent-3.2	platform.driver	30bf6384-3bde-4e28-8bf7-6695448f7b7d	Running (PID 3847)	Stop Remove
sqlhistorianagent-3.7.0	platform.historian	769d570d-e9b5-4263-8d07-31416ccec5c8	Running (PID 3845)	Stop Remove
vcplatformagent-4.8	platform.agent	f03b0ade-d161-4ddb-946b-30e129725584	Running (PID 3844)	Stop Remove
volttroncentralagent-5.0	volttron.central	63f67dd6-a27a-4064-949f-7c6649c43f73	Running (PID 3846)	Stop Remove

Install agents

Browse... No files selected.

For more information on VOLTTRON Central, please see:

- *VOLTTRON Central Management*
- *VOLTTRON Central Demo*

1.19 Linux System Hardening

1.19.1 Introduction

VOLTTRON is built with modern security principles in mind [security-wp] and implements many security features for hosted agents. However, VOLTTRON is built on top of Linux and the underlying Linux platform also needs to be secured in order to declare the resulting control system as "secure."

Any system is only as secure as its weakest link. This document is dedicated to making recommendations for hardening of the underlying Linux platform that VOLTTRON is deployed to.

Warning: No system can be 100% secure and the cyber security strategy that is recommended in this document is based on risk management. For the following guidance, it is intended that the user consider the risk, impact of risks, and perform the appropriate corresponding mitigation techniques.

1.19.2 Recommendations

Here are the non-exhaustive recommendations for Linux hardening from the VOLTTRON team:

- Physical Security: Keep the system in locked cabinets or a locked room. Limit physical access to systems and to the networks to which they are attached. The goal should be to avoid physical access by untrusted personnel. This could be extended to blocking or locking USB ports, removable media drives, etc.

  Drive encryption could be used to avoid access via alternate-media booting (off USB stick or DVD) if physical access can't be guaranteed. The downside of drive encryption would be needing to enter a passphrase to start system. Alternately, the *Trusted Platform Module* (TPM) may be used, but the drive might still be accessible to those with physical access. Enable chassis intrusion detection and reporting if supported. If available, use a physical tamper seal along with or in place of an interior switch.

- Low level device Security: Keep firmware of all devices (including BIOS) up-to-date. Password-protect the BIOS. Disable unneeded/unnecessary devices:
  - serial
  - parallel
  - USB (Leaving a USB port enabled may be helpful if a breach occurs to allow saving forensic data to an external drive.)
  - Firewire, etc.
  - ports
  - optical drives
  - wireless devices, such as Wi-Fi and Bluetooth
- Boot security:
  - Disable auto-mounting of external devices
  - Restrict the boot device:
    - Disable PXE and other network boot options (unless that is the primary boot method)
    - Disable booting from USB and other removable drives
  - Secure the boot loader:
    - Require an administrator password to do anything but start the default kernel
    - Do not allow editing of kernel parameters
    - Disable, remove, or password-protect emergency/recovery boot entries
- Security Updates: First and foremost, configure the system to automatically download security updates. Most security updates can be installed without rebooting the system, but some updated (e.g. shared libraries, kernel, etc) require the system to be rebooted. If possible, configure the system to install the security updates automatically and reboot at a particular time. We also recommend reserving the reboot time (e.g. 1:30AM on a Saturday morning) using the Actuator Agent so that no control actions can happen during that time.
- System Access only via Secured Protocols:
  - Disallow all clear text access to VOLTTRON systems

  - No telnet, no rsh, no ftp and no exceptions!
  - Use ssh to gain console access, and scp/sftp to get files in and out of the system
  - Disconnect excessively idle SSH Sessions
- Disable remote login for "root" users. Do not allow a user to directly access the system as the "root" user from a remote network location. Root access to privileged operations can be accomplished using `sudo`. This adds an extra level of security by restricting access to privileged operations and tracking those operations through the system log.
- Manage users and usernames, limit the number of user accounts, use complex usernames rather than first names.
- Authentication: If possible, use two factor authentication to allow access to the system. Informally, two factor authentication uses a combination of "something you know" and "something you have" to allow access to the system. RSA *SecurID* tokens are commonly used for two factor authentication but other tools are available. When not using two-factor authentication, use strong passwords and do not share accounts.
- Scan for weak passwords. Use password cracking tools such as John the Ripper or Nmap with password cracking modules to look for weak passwords.
- Utilize *Pluggable Authentication Modules* (PAM) to strengthen passwords and the login process. We recommend:
  - *pam_abl*: Automated blacklisting on repeated failed authentication attempts
  - *pam_captcha*: A visual text-based CAPTCHA challenge module for PAM
  - *pam_passwdqc*: A password strength checking module for PAM-aware password changing programs
  - *pam_cracklib*: PAM module to check the password against dictionary words
  - *pam_pwhistory*: PAM module to remember last passwords
- Disable unwanted services. Most desktop and server Linux distributions come with many unnecessary services enabled. Disable all unnecessary services. Refer to your distribution's documentation to discover how to check and disable these services.
- Just as scanning for weak passwords is a step to more secure systems; regular network scans using Nmap to find what network services are being offered is another step towards a more secure system.

Warning: use *Nmap* or similar tools very carefully on BACnet and modbus environments. These scanning tools are known to crash/reset BACnet and modbus devices.

- Control incoming and outgoing network traffic. Use the built-in host-based firewall to control who/what can connect to this system. Many *iptables* frontends offer a set of predefined rules that provide a default deny policy for incoming connections and provide rules to prevent or limit other well known attacks (i.e. rules that limit certain responses that might amplify a DDoS attack). *ufw* (uncomplicated firewall) is a good example.

  Examples:
  - If the system administrators for the VOLTTRON device are all located in `10.10.10.0/24` subnetwork, then allow SSH and SCP logins from only that IP address range.
  - If the VOLTTRON system exports data to a historian at `10.20.20.1` using TCP over port 443, allow outgoing traffic to that port on that server.

  The idea here is to limit the attack surface of the system. The smaller the surface, the better we can analyze the communication patterns of the system and detect anomalies.

Note: While some system administrators disable network-based diagnostic tools such as ICMP ECHO responses, the VOLTTRON team believes that this hampers usability. As an example, monitoring which incoming and outgoing firewall rules are triggering can be accomplished with this command:

```
watch --interval=5 'iptables -nvL | grep -v "0     0"'
```

- Rate limit incoming connections to discourage brute force hacking attempts. Use a tool such as fail2ban to dynamically manage firewall rules to rate limit incoming connections and discourage brute force hacking attempts. sshguard is similar to *fail2ban* but only used for ssh connections. Further rate limiting can be accomplished at the firewall level. As an example, you can restrict the number of connections used by a single IP address to your server using iptables. Only allow 4 ssh connections per client system:

  ```
  iptables -A INPUT -p tcp --syn --dport 22 -m connlimit --connlimit-above 4 -j DROP
  ```

  You can limit the number of connections per minute. The following example will drop incoming connections if an IP address makes more than 10 connection attempts to port 22 within 60 seconds:

  ```
  iptables -A INPUT -p tcp -dport 22 -i eth0 -m state --state NEW -m recent --set
  iptables -A INPUT -p tcp -dport 22 -i eth0 -m state --state NEW -m recent --
  ↪update --seconds 60 --hitcount 10 -j DROP
  ```

- Use a file system integrity tool to monitor for unexpected file changes. Tools such as tripwire monitor filesystems for changed files. Another file integrity checking tool to consider is AIDE (Advanced Intrusion Detect Environment).
- Use filesystem scanning tools periodically to check for exploits. Available tools such as checkrootkit, rkhunter and others should be used to check for known exploits on a periodic basis and report their results.
- VOLTTRON does not use Apache or require it. If Apache is being used, we recommend using the *mod_security* and *mod_evasive* modules.

Raspberry Pi

System hardening recommendations for Raspberry Pi closely match those for other Linux operating systems such as Ubuntu. VOLTTRON has only been officially tested with Raspbian, and there is one important consideration, which is noted in the Raspbian documentation as well:

Warning: The Raspbian operating system includes only the default *pi* user on install, which uses a well-known default password. For any operational deployment, it is recommended to create a new user with a complex password (this user must have sudoers permissions.

Summarizing the process of creating a new user *alice* from the Raspberry Pi documentation:

```
sudo adduser alice
sudo usermod -a -G adm,dialout,cdrom,sudo,audio,video,plugdev,games,users,input,
↪netdev,gpio,i2c,spi alice
sudo su - alice
sudo visudo /etc/sudoers.d/010_pi-nopasswd
```

When the editor opens for the sudoer's file, add an entry for *alice*:

```
alice ALL=(ALL) PASSWD: ALL
```

Also, update the default *pi* user's default password:

```
pi@raspberrypi:~/volttron$ passwd
Changing password for pi.
(current) UNIX password:
Enter new UNIX password:
Retype new UNIX password:
passwd: password updated successfully
```

Note: The Raspberry Pi documentation states that ideally one would remove the *pi* user from the system, however this is not currently recommended as some aspects of the Raspberry Pi OS are tied to the *pi* user. This will be changed in the future.

For more information, please visit the Raspberry Pi security site.

System Monitoring

- Monitor system state and resources. Use a monitoring tool such as Xymon or Big Brother to remotely monitor the system resources and state. Set the monitoring tools to alert the system administrators if anomalous use of resources (e.g. connections, memory, etc) are detected. An administrator can also use Unix commands such as *netstat* to look for open connections periodically.
- Watch system logs and get logs off the system. Use a utility such as logwatch or logcheck to get a daily summary of system activity via email. For Linux distributions that use *systemd* (such as Ubuntu), use journalwatch to accomplish the same task.

  Additionally, use a remote syslog server to collect logs from all VOLTTRON systems in the field at a centralized location for analysis. A tool such as *Splunk* is ideal for this task and comes with many built-in analysis applications. Another benefit of sending logs remotely off the platform is the ability to inspect the logs even when the platform may be compromised.
- An active intrusion sensor such as PSAD can be used to look for intrusions as well.

Security Testing

Every security control discussed in the previous sections must be tested to determine correct operation and impact. For example, if we inserted a firewall rule to ban connections from an IP address such as 10.10.10.2, then we need to test that the connections actually fail.

In addition to functional correctness testing, common security testing tools such as Nessus and Nmap should be used to perform cyber security testing.

1.19.3 Conclusion

No system is 100% secure unless it is disconnected from the network and is in a physically secure location. The VOLTTRON team recommends a risk-based cyber security approach that considers each risk, and the impact of an exploit. Mitigating technologies can then be used to mitigate the most impactful risks first. VOLTTRON is built with security in mind from the ground up, but it is only as secure as the operating system that it runs on top of. This document is intended to help VOLTTRON users to secure the underlying Linux operating system to further improve the robustness of the VOLTTRON platform. Any security questions should be directed to volttron@pnnl.gov.

1.20 Agents Overview

Agents in VOLTTRON can be loosely defined as software modules communicating on the platform which perform some function on behalf of the user. Agents may perform a huge variety of tasks, but common use cases involve data collection, control of ICS and IOT devices, and various platform management tasks. Agents implemented using the VOLTTRON agent framework inherit a number of capabilities, including message bus connectivity and agent lifecycle.

Agents deployed on VOLTTRON can perform one or more roles which can be broadly classified into the following groups:

- Platform Agents: Agents which are part of the platform and provide a service to other agents. Examples are the Actuator and Master Driver agents which serve as interfaces between control agents and drivers.
- Control Agents: These agents implement algorithms to control the devices of interest and interact with other resources to achieve some goal.
- Service Agents: These agents perform various data collection or platform management services. Agents in this category include weather service agents which collect weather data from remote sources or operations agents which help users maintain situational awareness of their deployment.
- Cloud Agents: These agents represent a remote application which needs access to the messages and data on the platform. This agent would subscribe to topics of interest to the remote application and would also allow it publish data to the platform.

The platform includes some valuable services which can be leveraged by agents:

- Message Bus: All agents and services publish and subscribe to topics on the message bus. This provides a single interface that abstracts the details of devices and agents from each other. Components in the platform basically produce and consume events.
- Configuration Store: Using the configuration store, agent operations can be altered ad-hoc without significant disruption or downtime.
- Historian Framework: Historian agents automatically collect data from a subset of topics on the message bus and store them in a data store of choice. Currently SQL, MongoDB, CrateDB and other historians exist, and more can be developed to fit the needs of a deployment by inheriting from the base historian. The base historian has been developed to be fast and reliable, and to handle many common pitfalls of data collection over a network.
- Weather Information: These agents periodically retrieve data from the a remote weather API then format the response and publish it to the platform message bus on a weather topic.
- Device interfaces: Drivers publish device data onto the message bus and send control signals issued from control agents to the corresponding device. Drivers are capable of handling the locking of devices to prevent multiple conflicting directives.
- Application Scheduling: This service allows the scheduling of agents' access to devices in order to prevent conflicts.
- Logging service: Agents can publish arbitrary strings to a logging topic and this service will push them to a historian for later analysis.

1.21 Core Services

Agents in the *services/core* directory support the most common use cases of the platform. For details on each, please refer to the corresponding documents.

1.21.1 Master Driver Agent

The Master Driver Agent manages all device communication. To communicate with devices you must setup and deploy the Master Driver Agent. For more information on the Master Driver Agent's operations, read about the *Master Driver* in the driver framework docs.

Configuring the Master Driver

Configuration for each device consists of 3 parts:

- Master Driver Agent configuration file - lists all driver configuration files to load
- Driver configuration file - contains the general driver configuration and device settings
- Device Register configuration file - contains the settings for each individual data point on the device

For each device, you must create a driver configuration file, device register configuration file, and an entry in the Master Driver Agent configuration file.

Once configured, the Master Driver Agent is configured and deployed in a manner similar to any other agent:

```
python scripts/install-agent.py -s services/core/MasterDriverAgent -c <master driver␣
↪config file>
```

Requirements

VOLTTRON drivers operated by the master driver may have additional requirements for installation. Required libraries:

```
BACnet driver - bacpypes
Modbus driver - pymodbus
Modbus_TK driver - modbus-tk
DNP3 and IEEE 2030.5 drivers - pydnp3
```

The easiest way to install the requirements for drivers included in the VOLTTRON repository is to use `bootstrap.py` (see *platform installation for more detail*)

Master Driver Agent Configuration

The Master Driver Agent configuration consists of general settings for all devices. The default values of the Master Driver should be sufficient for most users. The user may optionally change the interval between device scrapes with the driver_scrape_interval.

The following example sets the driver_scrape_interval to 0.05 seconds or 20 devices per second:

```
{
    "driver_scrape_interval": 0.05,
    "publish_breadth_first_all": false,
    "publish_depth_first": false,
    "publish_breadth_first": false,
    "publish_depth_first_all": true,
    "group_offset_interval": 0.0
}
```

- **driver_scrape_interval** - Sets the interval between devices scrapes. Defaults to 0.02 or 50 devices per second. Useful for when the platform scrapes too many devices at once resulting in failed scrapes.

- **group_offset_interval** - Sets the interval between when groups of devices are scraped. Has no effect if all devices are in the same group.

In order to improve the scalability of the platform unneeded device state publishes for all devices can be turned off. All of the following setting are optional and default to *True*.

- **publish_depth_first_all** - Enable "depth first" publish of all points to a single topic for all devices.
- **publish_breadth_first_all** - Enable "breadth first" publish of all points to a single topic for all devices.
- **publish_depth_first** - Enable "depth first" device state publishes for each register on the device for all devices.
- **publish_breadth_first** - Enable "breadth first" device state publishes for each register on the device for all devices.

An example master driver configuration file can be found in the VOLTTRON repository in *services/core/MasterDriverAgent/master-driver.agent*.

Driver Configuration File

Note: The terms *register* and *point* are used interchangeably in the documentation and in the configuration setting names. They have the same meaning in the context of VOLTTRON drivers.

Each device configuration has the following form:

```
{
    "driver_config": {"device_address": "10.1.1.5",
                      "device_id": 500},
    "driver_type": "bacnet",
    "registry_config":"config://registry_configs/vav.csv",
    "interval": 60,
    "heart_beat_point": "heartbeat",
    "group": 0
}
```

The following settings are required for all device configurations:

- **driver_config** - Driver specific setting go here. See below for driver specific settings.
- **driver_type** - Type of driver to use for this device: bacnet, modbus, fake, etc.
- **registry_config** - Reference to a configuration file in the configuration store for registers on the device. See the *Registry-Configuration-File* section below or and the *Adding Device Configurations to the Configuration Store* section in the driver framework docs.

These settings are optional:

- **interval** - Period which to scrape the device and publish the results in seconds. Defaults to 60 seconds.
- **heart_beat_point** - A Point which to toggle to indicate a heartbeat to the device. A point with this `Volttron Point Name` must exist in the registry. If this setting is missing the driver will not send a heart beat signal to the device. Heart beats are triggered by the *Actuator Agent* which must be running to use this feature.
- **group** - Group this device belongs to. Defaults to 0

These settings are used to create the topic that this device will be referenced by following the VOLTTRON convention of `{campus}/{building}/{unit}`. This will also be the topic published on, when the device is periodically scraped for it's current state.

The topic used to reference the device is derived from the name of the device configuration in the store. See the *Adding Device Configurations to the Configuration Store* section of the driver framework docs.

Device Grouping

Devices may be placed into groups to separate them logically when they are scraped. This is done by setting the *group* in the device configuration. *group* is a number greater than or equal to 0. Only number of devices in the same group and the *group_offset_interval* are considered when determining when to scrape a device.

This is useful in two cases:

- If you need to ensure that certain devices are scraped in close proximity to each other you can put them in their own group. If this causes devices to be scraped too quickly the groups can be separated out time wise using the *group_offset_interval* setting.
- You may scrape devices on different networks in parallel for performance. For instance BACnet devices behind a single MSTP router need to be scraped slowly and serially, but devices behind different routers may be scraped in parallel. Grouping devices by router will do this automatically.

The *group_offset_interval* is applied by multiplying it by the *group* number. If you intend to use *group_offset_interval* only use consecutive *group* values that start with 0.

Registry Configuration File

Registry configuration files setup each individual point on a device. Typically this file will be in CSV format, but the exact format is driver specific. See the section for a particular driver for the registry configuration format.

The following is a simple example of a Modbus registry configuration file:

Table 3: Catalyst 371

Reference Point Name	Volttron Point Name	Units	Units Details	Modbus Register	Writable	Point Address	Default Value	Notes
CO2Sensor	ReturnAirCO2	PPM	0.00-2000.00	>f	FALSE	1001		CO2 Reading 0.00-2000.0 ppm
CO2Stpt	ReturnAirCO2Stpt	PPM	1000.00 (default)	>f	TRUE	1011	1000	Setpoint to enable demand control ventilation
HeatCall2	HeatCall2	On / Off	on/off	BOOL	FALSE	1114		Status indicator of heating stage 2 need

Device State Publishes

By default, the value of each register on a device is published 4 different ways when the device state is published. Consider the following settings in a driver configuration stored under the name `devices/pnnl/isb1/vav1`:

```
{
    "driver_config": {"device_address": "10.1.1.5",
                      "device_id": 500},

    "driver_type": "bacnet",
    "registry_config":"config://registry_configs/vav.csv",
}
```

In the *vav.csv* file is a register with the name *temperature*. For these examples the current value of the register on the device happens to be 75.2 and the meta data is

```
{"units": "F"}
```

When the driver publishes the device state the following 2 things will be published for this register:

A "depth first" publish to the topic *devices/pnnl/isb1/vav1/temperature* with the following message:

```
[75.2, {"units": "F"}]
```

A "breadth first" publish to the topic *devices/temperature/vav1/isb1/pnnl* with the following message:

```
[75.2, {"units": "F"}]
```

These publishes can be turned off by setting *publish_depth_first* and *publish_breadth_first* to *false* respectively.

Also these two publishes happen once for all registers:

A "depth first" publish to the topic *devices/pnnl/isb1/vav1/all* with the following message:

```
[{"temperature": 75.2, ...}, {"temperature":{"units": "F"}, ...}]
```

A "breadth first" publish to the topic *devices/all/vav1/isb1/pnnl* with the following message:

```
[{"temperature": 75.2, ...}, {"temperature":{"units": "F"}, ...}]
```

These publishes can be turned off by setting *publish_depth_first_all* and *publish_breadth_first_all* to `false` respectively.

Device Scalability Settings

In order to improve the scalability of the platform unneeded device state publishes for a device can be turned off. All of the following setting are optional and will override the value set in the main master driver configuration.

- **publish_depth_first_all** - Enable "depth first" publish of all points to a single topic.
- **publish_breadth_first_all** - Enable "breadth first" publish of all points to a single topic.
- **publish_depth_first** - Enable "depth first" device state publishes for each register on the device.
- **publish_breadth_first** - Enable "breadth first" device state publishes for each register on the device.

It is common practice to set *publish_breadth_first_all*, *publish_depth_first*, and *publish_breadth_first* to *False* unless they are specifically needed by an agent running on the platform.

Note: All Historian Agents require *publish_depth_first_all* to be set to *True* in order to capture data.

Master Driver Override

By default, every user is allowed write access to the devices by the master driver. The override feature will allow the user (for example, building administrator) to override this default behavior and enable the user to lock the write access on the devices for a specified duration of time or indefinitely.

Set Override On

The Master Driver's `set_override_on` RPC method can be used to set the override condition for all drivers with topic matching the provided pattern. This can be specific devices, groups of devices, or even all configured devices. The pattern matching is based on bash style filename matching semantics.

Parameters:

- **pattern: Override pattern to be applied. For example,**
  - If the pattern is `campus/building1/*` the override condition is applied for all the devices under *campus/building1/*.
  - If the pattern is `campus/building1/ahu1` the override condition is applied for only the *campus/building1/ahu1* device. The pattern matching is based on bash style filename matching semantics.
- duration: Time duration for the override in seconds. If duration <= 0.0, it implies an indefinite duration.
- failsafe_revert: Flag to indicate if all the devices falling under the override condition has to be set to its default state/value immediately.
- staggered_revert: If this flag is set, reverting of devices will be staggered.

Example `set_override_on` RPC call:

```
self.vip.rpc.call(PLATFORM_DRIVER, "set_override_on", <override pattern>, <override␣
↪duration>)
```

Set Override Off

The override condition can also be toggled off based on a provided pattern using the Master Driver's `set_override_off` RPC call.

Parameters:

- **pattern: Override pattern to be applied. For example,**
  - If the pattern is `campus/building1/*` the override condition is removed for all the devices under *campus/building1/*.
  - If the pattern is `campus/building1/ahu1` the override condition is removed for only for the *campus/building1/ahu1* device. The pattern matching is based on bash style filename matching semantics.

Example `set_override_off` RPC call:

```
self.vip.rpc.call(PLATFORM_DRIVER, "set_override_off", <override pattern>)
```

Get Override Devices

A list of all overridden devices can be obtained with the Master Driver's `get_override_devices` RPC call.

This method call has no additional parameters.

Example `get_override_devices` RPC call:

```
self.vip.rpc.call(PLATFORM_DRIVER, "get_override_devices")
```

Get Override Patterns

A list of all patterns which have been requested for override can be obtained with the Master Driver's `get_override_patterns` RPC call.

This method call has no additional parameters

Example "get_override_patterns" RPC call:

```
self.vip.rpc.call(PLATFORM_DRIVER, "get_override_patterns")
```

Clear Overrides

All overrides set by RPC calls described above can be toggled off at using a single `clear_overrides` RPC call.

This method call has no additional parameters

Example "clear_overrides" RPC call:

```
self.vip.rpc.call(PLATFORM_DRIVER, "clear_overrides")
```

For more information, view the *Global Override Specification*

Global Override Specification

This document describes the specification for the global override feature. By default, every user is allowed write access to the devices by the master driver. The override feature will allow the user (for example, a building administrator) to override this default behavior and enable the user to lock the write access on the devices for a specified duration of time or indefinitely.

Functional Capabilities

1. User shall be able to specify the following when turning on the override behavior on the devices.
    - Override pattern examples:
        - If pattern is `campus/building1/*` - Override condition is turned on for all the devices under *campus/building1/*.
        - If pattern is `campus/building1/ahu1` - Override condition is turned on for only *campus/building1/ahu1*
        - The pattern matching shall use bash style filename matching semantics.
    - Time duration over which override behavior is applicable - If the time duration is negative, then the override condition is applied indefinitely.
    - Optional *revert-to-fail-safe-state* flag - If the flag is set, the master driver shall set all the set points falling under the override condition to its default state/value immediately. This is to ensure that the devices are in fail-safe state when the override/lock feature is removed. If the flag is not set, the device state/value is untouched.
    - Optional staggered revert flag - If this flag is set, reverting of devices will be staggered.
2. User shall be able to disable/turn off the override behavior on devices by specifying:
    - Pattern on which the override/lock feature has be disabled. (example: `campus/building/\*`)

3. User shall be able to get a list of all the devices with the override condition set.
4. User shall be able to get a list of all the override patterns that are currently active.
5. User shall be able to clear all the overrides.
6. Any changes to override patterns list shall be stored in the config store. On startup, list of override patterns and corresponding end times are retrieved from the config store. If the end time is indefinite or greater than current time for any pattern, then override is set on the matching devices for remaining duration of time.
7. Whenever a device is newly configured, a check is made to see if it is part of the overridden patterns. If yes, it is added to list of overridden devices.
8. When a device is being removed, a check is made to see if it is part of the overridden devices. If yes, it is removed from the list of overridden devices.

Driver RPC Methods

- *set_override_on(pattern, duration=0.0, failsafe_revert=True, staggered_revert=True)* - Turn on override condition on all the devices matching the pattern. Time duration for the override condition has to be in seconds. For indefinite duration, the time duration has to be <= 0.0.
- *set_override_off(pattern)* - Turn off override condition on all the devices matching the pattern. The specified pattern will be removed from the override patterns list. All the devices falling under the given pattern will be removed from the list of overridden devices.
- *get_override_devices()* - Get a list of all the devices with override condition.
- *get_override_patterns()* - Get a list of override patterns that are currently active.
- *clear_overrides()* - Clear all the overrides.

1.21.2 Market Service Agent

Introduction

The Market Service Agent implements a variation of a double-blind auction, in which each market participant bids to buy or sell a commodity for a given price.

In contrast to other common implementations, participants do not bid single price-quantity pairs. Instead, they bid a price-quantity curve, or "flexibility curve" into their respective markets. Market participants may be both buyers in one market and sellers in another.

Settling of the market is a "single shot" process that begins with bidding that progresses from the bottom up and concludes with a clearing of the markets from the top down. This is termed "single shot" because there is no iteration required to find the clearing price or quantity at any level of the market structure.

Once the market has cleared, the process begins again for the next market interval, and new bids are submitted based on the updated states of the agents.

Requirements

The Market Service Agent requires the *Transitions* (version 0.6.9) and *NumPy* (version 1.15.4) packages. These packages can be installed in an activated environment with:

```
pip install transitions==0.6.9
pip install numpy==1.15.4
```

Market Timing

The MarketServiceAgent is driven by the Director. The Director drives the MarketServiceAgent through a timed loop. The Director has just a few parameters that are configured by default with adequate values. They are:

1. The market_period with a default value of 5 minutes
2. The reservation_delay with a default value of 0 minutes
3. The offer_delay with a default value of 2 minutes

The timing loop works as follows:

- The market period begins.
- A request for reservations is published after the reservation delay.
- A request for offers/bids is published after the offer delay.
- The aggregate demand curve is published as soon all the buy offers are completed for the market.
- The aggregate supply curve is published as soon all the sell offers are completed for the market.
- The cleared price is published as soon as all bids have been received.
- Error messages are published when discovered and usually occur at the end of one of the delays.
- The cycle repeats.

How to Use the MarketServiceAgent

A given agent participates in one or more markets by inheriting from the *base MarketAgent*. The base MarketAgent handles all of the communication between the agent and the MarketServiceAgent. The agent only needs to join each market with the `join_market` method and then respond to the appropriate callback methods. The callback methods are described at the *base MarketAgent*.

1.21.3 DNP3 Agent

DNP3 (Distributed Network Protocol) is a set of communications protocols that are widely used by utilities such as electric power companies, primarily for SCADA purposes. It was adopted in 2010 as IEEE Std 1815-2010, later updated to 1815-2012.

VOLTTRON's DNP3 Agent is an implementation of a DNP3 Outstation as specified in IEEE Std 1815-2012. It engages in bidirectional network communications with a DNP3 Master, which might be located at a power utility.

Like some other VOLTTRON protocol agents (e.g. IEEE2030_5Agent), the DNP3 Agent can optionally be front-ended by a DNP3 device driver running under VOLTTRON's MasterDriverAgent. This allows a DNP3 Master to be treated like any other device in VOLTTRON's ecosystem.

The VOLTTRON DNP3 Agent implementation of an Outstation is built on PyDNP3, an open-source library from Kisensum containing Python language bindings for Automatak's C++ opendnp3 library, the de facto reference implementation of DNP3.

The DNP3 Agent exposes DNP3 application-layer functionality, creating an extensible base from which specific custom behavior can be designed and supported. By default, the DNP3 Agent acts as a simple transfer agent, publishing data received from the Master on the VOLTTRON Message Bus, and responding to RPCs from other VOLTTRON agents by sending data to the Master.

Requirements

PyDNP3 can be installed in an activated environment with:

```
pip install pydnp3
```

RPC Calls

The DNP3 Agent exposes the following VOLTTRON RPC calls:

```
def get_point(self, point_name):
    """
        Look up the most-recently-received value for a given output point.

    @param point_name: The point name of a DNP3 PointDefinition.
    @return: The (unwrapped) value of a received point.
    """

def get_point_by_index(self, group, index):
    """
        Look up the most-recently-received value for a given point.

    @param group: The group number of a DNP3 point.
    @param index: The index of a DNP3 point.
    @return: The (unwrapped) value of a received point.
    """

def get_points(self):
    """
        Look up the most-recently-received value of each configured output point.

    @return: A dictionary of point values, indexed by their VOLTTRON point names.
    """

def set_point(self, point_name, value):
    """
        Set the value of a given input point.

    @param point_name: The point name of a DNP3 PointDefinition.
    @param value: The value to set. The value's data type must match the one in the␣
↪DNP3 PointDefinition.
    """

def set_points(self, point_list):
    """
        Set point values for a dictionary of points.

    @param point_list: A dictionary of {point_name: value} for a list of DNP3 points␣
↪to set.
    """

def config_points(self, point_map):
    """
        For each of the agent's points, map its VOLTTRON point name to its DNP3 group␣
↪and index.
```

(continues on next page)

(continued from previous page)

```
    @param point_map: A dictionary that maps a point's VOLTTRON point name to its␣
↪DNP3 group and index.
    """

def get_point_definitions(self, point_name_list):
    """
        For each DNP3 point name in point_name_list, return a dictionary with each of␣
↪the point definitions.

        The returned dictionary looks like this:

        {
            "point_name1": {
                "property1": "property1_value",
                "property2": "property2_value",
                ...
            },
            "point_name2": {
                "property1": "property1_value",
                "property2": "property2_value",
                ...
            }
        }

        If a definition cannot be found for a point name, it is omitted from the␣
↪returned dictionary.

    :param point_name_list: A list of point names.
    :return: A dictionary of point definitions.
    """
```

Pub/Sub Calls

The DNP3 Agent uses two topics when publishing data to the VOLTTRON message bus:

- **Point Values (default topic: 'dnp3/point')**: As the DNP3 Agent communicates with the Master, it publishes received point values on the VOLTTRON message bus.
- **Outstation status (default topic: dnp3/status)**: If the status of the DNP3 Agent outstation changes, for example if it is restarted, it publishes its new status on the VOLTTRON message bus.

Data Dictionary of Point Definitions

The DNP3 Agent loads and uses a data dictionary of point definitions, which are maintained by agreement between the (DNP3 Agent) Outstation and the DNP3 Master. The data dictionary is stored in the agent's registry.

Current Point Values

The DNP3 Agent tracks the most-recently-received value for each point definition in its data dictionary, regardless of whether the point value's source is a VOLTTRON RPC call or a message from the DNP3 Master.

Agent Configuration

The DNP3Agent configuration file specifies the following fields:

- **local_ip**: (string) Outstation's host address (DNS resolved). Default: `0.0.0.0`.
- **port**: (integer) Outstation's port number - the port that the remote endpoint (Master) is listening on. Default: 20000.
- **point_topic**: (string) VOLTTRON message bus topic to use when publishing DNP3 point values. Default: `dnp3/point`.
- **outstation_status_topic**: (string) Message bus topic to use when publishing outstation status. Default: `dnp3/outstation_status`.
- **outstation_config**: (dictionary) Outstation configuration parameters. All are optional. Parameters include:
  - **database_sizes**: (integer) Size of each outstation database buffer. Default: 10.
  - **event_buffers**: (integer) Size of the database event buffers. Default: 10.
  - **allow_unsolicited**: (boolean) Whether to allow unsolicited requests. Default: `True`.
  - **link_local_addr**: (integer) Link layer local address. Default: 10.
  - **link_remote_addr**: (integer) Link layer remote address. Default: 1.
  - **log_levels**: (list) List of bit field names (*OR'd* together) that filter what gets logged by DNP3. Default: `NORMAL`. Possible values: `ALL`, `ALL_APP_COMMS`, `ALL_COMMS`, `NORMAL`, `NOTHING`.
  - **threads_to_allocate**: (integer) Threads to allocate in the manager's thread pool. Default: 1.

A sample DNP3 Agent configuration file is available in *services/core/DNP3Agent/dnp3agent.config*.

VOLTTRON DNP3 Device Driver

VOLTTRON's DNP3 device driver exposes *get_point*/*set_point* RPC calls and scrapes for DNP3 points.

The driver periodically issues DNP3Agent RPC calls to refresh its cached representation of DNP3 data. It issues RPC calls to the DNP3 Agent as needed when responding to *get_point*, *set_point* and *scrape_all* calls.

For information about the DNP3 driver, see *DNP3 Driver*.

Installing the DNP3 Agent

To install DNP3Agent, please consult the installation advice in *services/core/DNP3Agent/README.md*. *README.md* specifies a default agent configuration, which can be overridden as needed.

An agent installation script is available:

```
$ export VOLTTRON_ROOT=<volttron github install directory>
$ cd $VOLTTRON_ROOT
$ source services/core/DNP3Agent/install_dnp3_agent.sh
```

When installing the Mesa Agent, please note that the agent's point definitions must be loaded into the agent's config store. See *install_dnp3_agent.sh* for an example of how to load them.

1.21.4 Mesa Agent

The Mesa Agent is a VOLTTRON agent that handles MESA-ESS DNP3 outstation communications. It subclasses and extends the functionality of VOLTTRON's DNP3 Agent. Like the DNP3 Agent, the Mesa Agent models a DNP3 outstation, communicating with a DNP3 master.

For a description of DNP3 and the VOLTTRON DNP3 agent, please refer to the *DNP3 Agent documentation*.

VOLTTRON's Mesa Agent and DNP3 Agent are implementations of a DNP3 Outstation as specified in IEEE Std 1815-2012. They engage in bidirectional network communications with a DNP3 Master, which might be located at a power utility.

MESA-ESS is an extension and enhancement to DNP3. It builds on the basic DNP3 communications protocol, adding support for more complex structures, including functions, arrays, curves and schedules. The draft specification for MESA-ESS, as well as a spreadsheet of point definitions, can be found at http://mesastandards.org/mesa-standards/.

VOLTTRON's DNP3 Agent and Mesa Agent implementations of an Outstation are built on *pydnp3*, an open-source library from Kisensum containing Python language bindings for Automatak's C++ opendnp3 library, the de-facto reference implementation of DNP3.

MesaAgent exposes DNP3 application-layer functionality, creating an extensible base from which specific custom behavior can be designed and supported, including support for MESA functions, arrays and selector blocks. By default, the Mesa Agent acts as a simple transfer agent, publishing data received from the Master on the VOLTTRON Message Bus, and responding to RPCs from other VOLTTRON agents by sending data to the Master. Properties of the point and function definitions also enable the use of more complex controls for point data capture and publication.

The Mesa Agent was developed by Kisensum for use by 8minutenergy, which provided generous financial support for the open-source contribution to the VOLTTRON platform, along with valuable feedback based on experience with the agent in a production context.

RPC Calls

The Mesa Agent exposes the following VOLTTRON RPC calls:

```python
def get_point(self, point_name):
    """
        Look up the most-recently-received value for a given output point.

    @param point_name: The point name of a DNP3 PointDefinition.
    @return: The (unwrapped) value of a received point.
    """

def get_point_by_index(self, data_type, index):
    """
        Look up the most-recently-received value for a given point.

    @param data_type: The data_type of a DNP3 point.
    @param index: The index of a DNP3 point.
    @return: The (unwrapped) value of a received point.
    """

def get_points(self):
    """
        Look up the most-recently-received value of each configured output point.

    @return: A dictionary of point values, indexed by their point names.
    """
```

(continues on next page)

(continued from previous page)

```
def get_configured_points(self):
    """
        Look up the most-recently-received value of each configured point.

    @return: A dictionary of point values, indexed by their point names.
    """

def set_point(self, point_name, value):
    """
        Set the value of a given input point.

    @param point_name: The point name of a DNP3 PointDefinition.
    @param value: The value to set. The value's data type must match the one in the␣
↪DNP3 PointDefinition.
    """

def set_points(self, point_dict):
    """
        Set point values for a dictionary of points.

    @param point_dict: A dictionary of {point_name: value} for a list of DNP3 points␣
↪to set.
    """

def config_points(self, point_map):
    """
        For each of the agent's points, map its VOLTTRON point name to its DNP3 group␣
↪and index.

    @param point_map: A dictionary that maps a point's VOLTTRON point name to its␣
↪DNP3 group and index.
    """

def get_point_definitions(self, point_name_list):
    """
        For each DNP3 point name in point_name_list, return a dictionary with each of␣
↪the point definitions.

        The returned dictionary looks like this:

        {
            "point_name1": {
                "property1": "property1_value",
                "property2": "property2_value",
                ...
            },
            "point_name2": {
                "property1": "property1_value",
                "property2": "property2_value",
                ...
            }
        }

        If a definition cannot be found for a point name, it is omitted from the␣
↪returned dictionary.

    :param point_name_list: A list of point names.
```

(continues on next page)

(continued from previous page)

```
    :return: A dictionary of point definitions.
    """

def get_selector_block(self, point_name, edit_selector):
    """
        Return a dictionary of point values for a given selector block.

    :param point_name: Name of the first point in the selector block.
    :param edit_selector: The index (edit selector) of the block.
    :return: A dictionary of point values.
    """

def reset(self):
    """
        Reset the agent's internal state, emptying point value caches. Used during␣
↪iterative testing.
    """
```

Pub/Sub Calls

MesaAgent uses three topics when publishing data to the VOLTTRON message bus:

- **Point Values (default topic: dnp3/point)**: As the Mesa Agent communicates with the Master, it publishes received point values on the VOLTTRON message bus.
- **Functions (default topic: mesa/function)**: When the Mesa Agent receives a function step with a "publish" action value, it publishes the current state of the function (all steps received to date) on the VOLTTRON message bus.
- **Outstation status (default topic: mesa/status)**: If the status of the Mesa Agent outstation changes, for example if it is restarted, it publishes its new status on the VOLTTRON message bus.

Data Dictionaries of Point and Function Definitions

The Mesa Agent loads and uses data dictionaries of point and function definitions, which are maintained by agreement between the (Mesa Agent) Outstation and the DNP3 Master. The data dictionaries are stored in the agent's registry.

Current Point Values

The Mesa Agent tracks the most-recently-received value for each point definition in its data dictionary, regardless of whether the point value's source is a VOLTTRON RPC call or a message from the DNP3 Master.

Agent Configuration

The Mesa Agent configuration specifies the following fields:

- **local_ip**: (string) Outstation's host address (DNS resolved). Default: `0.0.0.0`.
- **port**: (integer) Outstation's port number - the port that the remote endpoint (Master) is listening on. Default: 20000.
- **point_topic**: (string) VOLTTRON message bus topic to use when publishing DNP3 point values. Default: `dnp3/point`.

- **function_topic**: (string) Message bus topic to use when publishing MESA-ESS functions. Default: `mesa/function`.
- **outstation_status_topic**: (string) Message bus topic to use when publishing outstation status. Default: `mesa/outstation_status`.
- **all_functions_supported_by_default**: (boolean) When deciding whether to reject points for unsupported functions, ignore the values of their 'supported' points: simply treat all functions as supported. Used primarily during testing. Default: `False`.
- **function_validation**: (boolean) When deciding whether to support sending single points to the Mesa Agent. If *function_validation* is `True`, the Mesa Agent will raise an exception when receiving any invalid point in current function. If *function_validation* is `False`, Mesa Agent will reset current function to None instead of raising the exception. Default: `False`.
- **outstation_config**: (dictionary) Outstation configuration parameters. All are optional. Parameters include:
  - **database_sizes**: (integer) Size of each outstation database buffer. Default: 10.
  - **event_buffers**: (integer) Size of the database event buffers. Default: 10.
  - **allow_unsolicited**: (boolean) Whether to allow unsolicited requests. Default: `True`.
  - **link_local_addr**: (integer) Link layer local address. Default: 10.
  - **link_remote_addr**: (integer) Link layer remote address. Default: 1.
  - **log_levels**: (list) List of bit field names (OR'd together) that filter what gets logged by DNP3. Default: [NORMAL]. Possible values: `ALL`, `ALL_APP_COMMS`, `ALL_COMMS`, `NORMAL`, `NOTHING`.
  - **threads_to_allocate**: (integer) Threads to allocate in the manager's thread pool. Default: 1.

A sample Mesa Agent configuration file is available in `services/core/DNP3Agent/mesaagent.config`.

Installing the Mesa Agent

To install the Mesa Agent, please consult the installation advice in `services/core/DNP3Agent/README.md`, which includes advice on installing `pydnp3`, a library upon which the DNP3 Agent depends.

After installing libraries as described in the Mesa Agent *README.md* file, the agent can be installed from a command-line shell as follows:

```
$ export VOLTTRON_ROOT=<volttron github install directory>
$ cd $VOLTTRON_ROOT
$ source services/core/DNP3Agent/install_mesa_agent.sh
```

README.md specifies a default agent configuration, which can be overridden as needed.

Here are some things to note when installing the Mesa Agent:

- The Mesa Agent source code resides in, and is installed from, a DNP3 subdirectory, thus allowing it to be implemented as a subclass of the base DNP3 agent class. When installing the Mesa Agent, inform the install script that it should build from the *mesa* subdirectory by exporting the following environment variable:

```
$ export AGENT_MODULE=dnp3.mesa.agent
```

- The agent's point and function definitions must be loaded into the agent's config store. See the `install_mesa_agent.sh` script for an example of how to load them.

1.21.5 External Data Publisher Agent

The External Data Publisher agent (ExternalData) was created to fetch data from remote APIs based on configured values and publish the remote data on the VOLTTRON message bus. The agent is primarily an agent wrapper around the requests library that sends the request then broadcast it via VIP pub/sub publish.

Configuration Options

The following JSON configuration file shows all the options currently supported by the ExternalData agent. Configuration values specify the interval between remote data polling requests, default authentication for remote API calls, VOLTTRON message bus publish topics, and for defining the remote API request behavior. Below is an example configuration file with additional parameter documentation.

```
{
    #Interval at which to scrape the sources.
    "interval":300,

    #Global topic prefix for all publishes.
    "global_topic_prefix": "record",

    #Default user name and password if all sources require the same
    #credentials. Can be overridden in individual sources.
    #"default_user":"my_user_name",
    #"default_password" : "my_password",

    "sources":
    [
     {
        #Valid types are "csv", "json", and "raw"
        #Defaults to "raw"
        "type": "csv",
        #Source URL for CSV data.
        "url": "https://example.com/example",

        #URL parameters for data query (optional).
        # See https://en.wikipedia.org/wiki/Query_string
        "params": {"period": "currentinterval",
                   "format": "csv"},

        #Topic to publish on.
        "topic": "example/examplecsvdata1",

        #Column used to break rows in CSV out into separate publishes.
        #The key will be removed from the row data and appended to the end
        # of the publish topic.
        # If this option is missing the entire CSV will be published as a list
        # of objects.
        #If the column does not exist nothing will be published.
        "key": "Key Column",

        #Attempt to parse these columns in the data into numeric types.
        #Currently columns are parsed with ast.literal_eval()
        #Values that fail to parse are left as strings unless the
        # values is an empty string. Empty strings are changed to None.
        "parse": ["Col1", "Col2"],
```

(continues on next page)

(continued from previous page)

```
        #Source specific authentication.
        "user":"username",
        "password" : "password"
    },
    {
        #Valid types are "csv", "json", and "raw"
        #Defaults to "raw"
        "type": "csv",
        #Source URL for CSV data.
        "url": "https://example.com/example_flat",

        #URL parameters for data query (optional).
        # See https://en.wikipedia.org/wiki/Query_string
        "params": {"format": "csv"},

        #Topic to publish on. (optional)
        "topic": "example/examplecsvdata1",

        #If the rows in a csv represent key/value pairs use this
        #setting to reduce this format to a single object for publishing.
        "flatten": true,

        #Attempt to parse these columns in the data into numeric types.
        #Currently columns are parsed with ast.literal_eval()
        #Values that fail to parse are left as strings unless the
        # values is an empty string. Empty strings are changed to None.
        "parse": ["Col1", "Col2"]
    },
    {
        #Valid types are "csv", "json", and "raw"
        #Defaults to "raw"
        "type": "json",
        #Source URL for JSON data.
        "url": "https://example.com/api/example1",

        #URL parameters for data query (optional)
        # See https://en.wikipedia.org/wiki/Query_string
        "params": {"format": "json"},

        #Topic to publish on. (optional)
        "topic": "example/exampledata1",

        #Path to desired data withing the JSON. Optional.
        #Elements in a path may be either a string or an integer.
        #Useful for peeling off unneeded layers around the wanted data.
        "path": ["parentobject", "0"],

        #After resolving the path above if the resulting data is a list
        # the key is the path to a value in a list item. Each item in the list
        # is published separately with the key appended to the end of the topic.
        # Elements in a key may be a string or an integer. (optional)
        "key": ["Location", "$"],

        #Source specific authentication.
        "user":"username",
        "password" : "password"
    }
```

(continues on next page)

(continued from previous page)

```
    ]
}
```

1.21.6 IEEE 2030.5 DER Agent

The IEEE 2030.5 Agent (IEEE2030_5 in the VOLTTRON repository) implements a IEEE 2030.5 server that receives HTTP *POST*/*PUT* requests from IEEE 2030.5 devices. The requests are routed to the IEEE 2030.5 Agent over the VOLTTRON message bus by VOLTTRON's Master Web Service. The IEEE 2030.5 Agent returns an appropriate HTTP response. In some cases (e.g., DERControl requests), this response includes a data payload.

The IEEE 2030.5 Agent maps IEEE 2030.5 resource data to a VOLTTRON IEEE 2030.5 data model based on SunSpec, using block numbers and point names as defined in the SunSpec Information Model, which in turn is harmonized with 61850. The data model is given in detail below.

Each device's data is stored by the IEEE 2030.5 Agent in an *EndDevice* memory structure. This structure is not persisted to a database. Each *EndDevice* retains only the most recently received value for each field.

The IEEE2030_5 Agent exposes RPC calls for getting and setting EndDevice data.

VOLTTRON IEEE 2030.5 Device Driver

The *IEEE 2030.5 device driver* is a new addition to VOLTTRON Master Driver Agent's family of standard device drivers. It exposes `get_point/set_point calls` for IEEE 2030.5 EndDevice fields.

The IEEE 2030.5 device driver periodically issues IEEE2030_5 Agent RPC calls to refresh its cached representation of EndDevice data. It issues RPC calls to IEEE2030_5Agent as needed when responding to `get_point`, `set_point` and `scrape_all` calls.

Field Definitions

These field IDs correspond to the ones in the IEEE 2030.5 device driver's configuration file, `ieee2030_5.csv`. They have been used in that file's "Volttron Point Name" column and also in its "Point Name" column.

Field ID	IEEE 2030.5 Resource/Property	Description	Units	Type
b1_Md	**device_information** mfModel	Model (32 char lim).		string
b1_Opt	**device_information** lfdi	Long-form device identifier (32 char lim).		string
b1_SN	**abstract_device** sfdi	Short-form device identifier (32 char lim).		string
b1_Vr	**device_information** mfHwVer	Version (16 char lim).		string
b113_A	**mirror_meter_reading** PhaseCurrentAvg	AC current.	A	float
b113_DCA	**mirror_meter_reading** InstantPackCurrent	DC current.	A	float
b113_DCV	**mirror_meter_reading** LineVoltageAvg	DC voltage.	V	float
b113_DCW	**mirror_meter_reading** PhasePowerAvg	DC power.	W	float
b113_PF	**mirror_meter_reading** PhasePFA	AC power factor.	%	float
b113_WH	**mirror_meter_reading** EnergyIMP	AC energy.	Wh	float
b120_AhrRtg	**der_capability** rtgAh	Usable capacity of the battery. Maximum charge minus minimum charge.	Ah	float
b120_ARtg	**der_capability** rtgA	Maximum RMS AC current level capability of the inverter.	A	float
b120_MaxChaRte	**der_capability** rtgMaxChargeRate	Maximum rate of energy transfer into the device.	W	float
b120_MaxDisChaRte	**der_capability** rtgMaxDischargeRate	Maximum rate of energy transfer out of the device.	W	float
b120_WHRtg	**der_capability** rtgWh	Nominal energy rating of the storage device.	Wh	float
b120_WRtg	**der_capability** rtgW	Continuous power output capability of the inverter.	W	float

Revising and Expanding the Field Definitions

The IEEE 2030.5-to-SunSpec field mappings in this implementation are a relatively thin subset of all possible field definitions. Developers are encouraged to expand the definitions.

The procedure for expanding the field mappings requires you to make changes in two places:

1. Update the driver's point definitions in `services/core/MasterDriverAgent/master_driver/ieee2030_5.csv`
2. Update the IEEE 2030.5-to-SunSpec field mappings in `services/core/IEEE2030_5Agent/ieee2030_5/end_device.py` and `__init__.py`

When updating VOLTTRON's IEEE 2030.5 data model, please use field IDs that conform to the SunSpec block-number-and-field-name model outlined in the SunSpec Information Model Reference (see the link below).

View the *IEEE 2030.5 agent specification document* to learn more about IEEE 2030.5 and the IEEE 2030.5 agent and driver.

IEEE 2030.5 DER Support

Version 1.0

Smart Energy Profile 2.0 (SEP 2.0, IEEE 2030.5) specifies a REST architecture built around the core HTTP verbs: GET, HEAD, PUT, POST and DELETE. A specification for the IEEE 2030.5 protocol can be found here.

IEEE 2030.5 EndDevices (clients) POST XML resources representing their state, and GET XML resources containing command and control information from the server. The server never reaches out to the client unless a "subscription" is registered and supported for a particular resource type. This implementation does not use IEEE 2030.5 registered subscriptions.

The IEEE 2030.5 specification requires HTTP headers, and it explicitly requires RESTful response codes, for example:

- 201 - "Created"
- 204 - "No Content"
- 301 - "Moved Permanently"
- etc.

IEEE 2030.5 message encoding may be either XML or EXI. Only XML is supported in this implementation.

IEEE 2030.5 requires HTTPS/TLS version 1.2 along with support for the cipher suite TLS_ECDHE_ECDSA_WITH_AES_128_CCM_8. Production installation requires a certificate issued by a IEEE 2030.5 CA. The encryption requirement can be met by using a web server such as Apache to proxy the HTTPs traffic.

IEEE 2030.5 discovery, if supported, must be implemented by an xmDNS server. Avahi can be modified to perform this function.

Function Sets

IEEE 2030.5 groups XML resources into "Function Sets." Some of these function sets provide a core set of functionality used across higher-level function sets. This implementation implements resources from the following function sets:

- Time
- Device Information

- Device Capabilities
- End Device
- Function Set Assignments
- Power Status
- Distributed Energy Resources

Distributed Energy Resources (DERs)

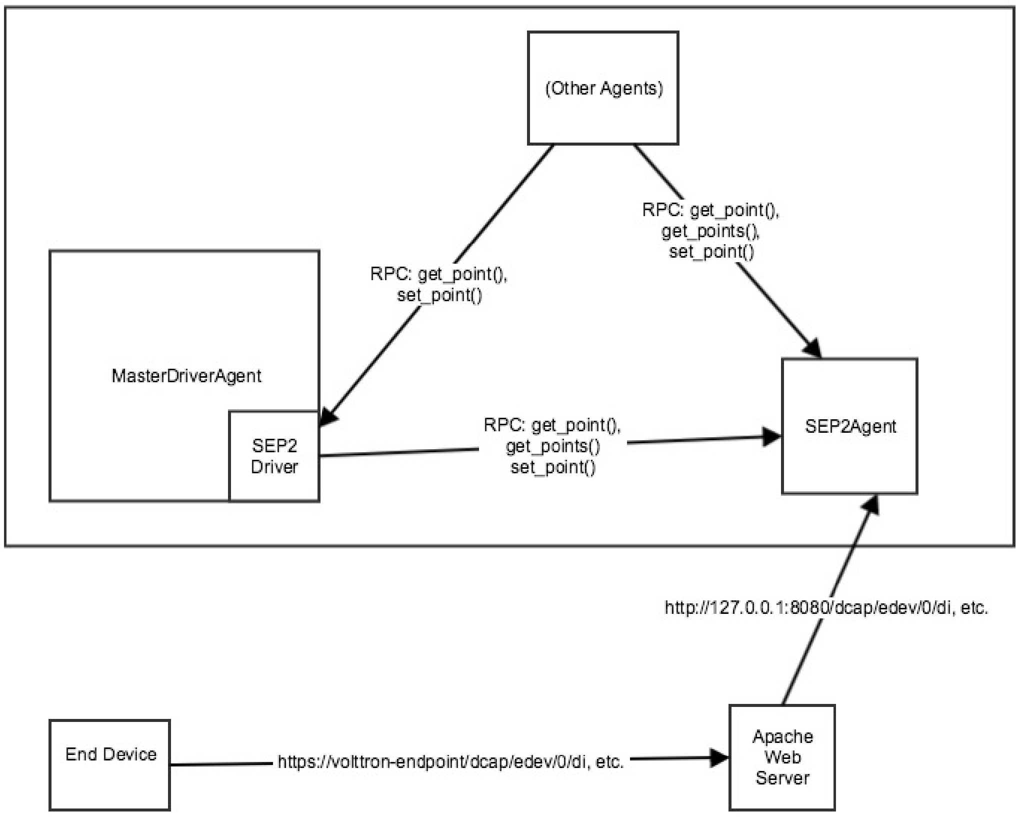
Distributed energy resources (DERs) are devices that generate energy, e.g., solar inverters, or store energy, e.g., battery storage systems, electric vehicle supply equipment (EVSEs). These devices are managed by a IEEE 2030.5 DER server using DERPrograms which are described by the IEEE 2030.5 specification as follows:

> Servers host one or more DERPrograms, which in turn expose DERControl events to DER clients. DERControl instances contain attributes that allow DER clients to respond to events that are targeted to their device type. A DERControl instance also includes scheduling attributes that allow DER clients to store and process future events. These attributes include start time and duration, as well an indication of the need for randomization of the start and / or duration of the event. The IEEE 2030.5 DER client model is based on the SunSpec Alliance Inverter Control Model [SunSpec] which is derived from IEC 61850-90-7 [61850] and [EPRI].

EndDevices post multiple IEEE 2030.5 resources describing their status. The following is an example of a Power Status resource that might be posted by an EVSE (vehicle charging station):

```
<PowerStatus xmlns="http://zigbee.org/sep" xmlns:xsi="http://www.w3.org/2001/
↪XMLSchema-instance" href="/sep2/edev/96/ps">
    <batteryStatus>4</batteryStatus>
    <changedTime>1487812095</changedTime>
    <currentPowerSource>1</currentPowerSource>
    <estimatedChargeRemaining>9300</estimatedChargeRemaining>
    <PEVInfo>
        <chargingPowerNow>
            <multiplier>3</multiplier>
            <value>-5</value>
        </chargingPowerNow>
        <energyRequestNow>
            <multiplier>3</multiplier>
            <value>22</value>
        </energyRequestNow>
        <maxForwardPower>
            <multiplier>3</multiplier>
            <value>7</value>
        </maxForwardPower>
        <minimumChargingDuration>11280</minimumChargingDuration>
        <targetStateOfCharge>10000</targetStateOfCharge>
        <timeChargeIsNeeded>9223372036854775807</timeChargeIsNeeded>
        <timeChargingStatusPEV>1487812095</timeChargingStatusPEV>
    </PEVInfo>
</PowerStatus>
```

Design Details


(Other Agents)
RPC: get_point(), set_point()
RPC: get_point(), get_points(), set_point()
MasterDriverAgent
SEP2 Driver
RPC: get_point(), get_points() set_point()
SEP2Agent
http://127.0.0.1:8080/dcap/edev/0/di, etc.
End Device
https://volttron-endpoint/dcap/edev/0/di, etc.
Apache Web Server

VOLTTRON's IEEE 2030.5 implementation includes a IEEE 2030.5 Agent and a IEEE 2030.5 device driver, as described below.

VOLTTRON IEEE 2030.5 Device Driver

The IEEE 2030.5 device driver is a new addition to VOLTTRON Master Driver Agent's family of standard device drivers. It exposes *get_point*/*set_point* calls for IEEE 2030.5 EndDevice fields.

The IEEE 2030.5 device driver periodically issues the IEEE 2030.5 Agent RPC calls to refresh its cached representation of EndDevice data. It issues RPC calls to the IEEE 2030.5 Agent as needed when responding to *get_point*, *set_point* and *scrape_all* calls.

Field Definitions

These field IDs correspond to the ones in the IEEE 2030.5 device driver's configuration file, *ieee2030_5.csv*. They have been used in that file's *Volttron Point Name* column and also in its *Point Name* column.

Field ID	IEEE 2030.5 Resource/Property	Description	Units	Type
b1_Md	**device_information** mfModel	Model (32 char lim).		string
b1_Opt	**device_information** lfdi	Long-form device identifier (32 char lim).		string
b1_SN	**abstract_device** sfdi	Short-form device identifier (32 char lim).		string
b1_Vr	**device_information** mfHwVer	Version (16 char lim).		string
b113_A	**mirror_meter_reading** PhaseCurrentAvg	AC current.	A	float
b113_DCA	**mirror_meter_reading** InstantPackCurrent	DC current.	A	float
b113_DCV	**mirror_meter_reading** LineVoltageAvg	DC voltage.	V	float
b113_DCW	**mirror_meter_reading** PhasePowerAvg	DC power.	W	float
b113_PF	**mirror_meter_reading** PhasePFA	AC power factor.	%	float
b113_WH	**mirror_meter_reading** EnergyIMP	AC energy.	Wh	float
b120_AhrRtg	**der_capability** rtgAh	Usable capacity of the battery. Maximum charge minus minimum charge.	Ah	float
b120_ARtg	**der_capability** rtgA	Maximum RMS AC current level capability of the inverter.	A	float
b120_MaxChaRte	**der_capability** rtgMaxChargeRate	Maximum rate of energy transfer into the device.	W	float
b120_MaxDisChaRte	**der_capability** rtgMaxDischargeRate	Maximum rate of energy transfer out of the device.	W	float
b120_WHRtg	**der_capability** rtgWh	Nominal energy rating of the storage device.	Wh	float
b120_WRtg	**der_capability** rtgW	Continuous power output capability of the inverter.	W	float

Revising and Expanding the Field Definitions

The IEEE 2030.5-to-SunSpec field mappings in this implementation are a relatively thin subset of all possible field definitions. Developers are encouraged to expand the definitions.

The procedure for expanding the field mappings requires you to make changes in two places:

1. Update the driver's point definitions in *services/core/MasterDriverAgent/master_driver/ieee2030_5.csv*
2. Update the IEEE 2030.5-to-SunSpec field mappings in *services/core/IEEE2030_5Agent/ieee2030_5/end_device.py* and *__init__.py*

When updating VOLTTRON's IEEE 2030.5 data model, please use field IDs that conform to the SunSpec block-number-and-field-name model outlined in the SunSpec Information Model Reference (see the link below).

For Further Information

SunSpec References:

- Information model specification: http://sunspec.org/wp-content/uploads/2015/06/SunSpec-Information-Models-12041.pdf
- Information model reference spreadsheet: http://sunspec.org/wp-content/uploads/2015/06/SunSpec-Information-Model-Reference.xlsx
- Inverter models: http://sunspec.org/wp-content/uploads/2015/06/SunSpec-Inverter-Models-12020.pdf
- Energy storage models: http://sunspec.org/wp-content/uploads/2015/06/SunSpec-Energy-Storage-Models-12032.pdf

Questions? Please contact:

- Rob Calvert (rob@kisensum.com) or James Sheridan (james@kisensum.com)

1.21.7 Obix History Agent

The Obix History Agent captures data history data from an Obix RESTful interface and publishes it to the message bus like a driver for capture by agents and historians. The Agent will setup its queries to ensure that data is only publishes once. For points queried for the first time it will go back in time and publish old data as configured.

The data will be collated into device *all* publishes automatically and will use a timestamp in the header based on the timestamps reported by the Obix interface. The publishes will be made in chronological order.

Units data is automatically read from the device.

For sending commands to devices see *Obix Driver Configuration*.

Agent Configuration

There are three arguments for the **driver_config** section of the device configuration file:

- **url** - URL of the interface.
- **username** - User name for site..
- **password** - Password for username.
- **check_interval** - How often to check for new data on each point.
- **path_prefix** - Path prefix for all publishes.

- **register_config** - Registry configuration file.
- **default_last_read** - Time, in hours, to go back and retrieve data for a point for the first time.

Here is an example device configuration file:

```
{
  "url": "http://example.com/obix/histories/EXAMPLE/",
  "username": "username",
  "password": "password",
  # Interval to query interface for updates in minutes.
  # History points are only published if new data is available
  # config points are gathered and published at this interval.
  "check_interval": 15,
  # Path prefix for all publishes
  "path_prefix": "devices/obix/history/",
  "register_config": "config://registry_config.csv",
  "default_last_read": 12
}
```

A sample Obix configuration file can be found in the VOLTTRON repository in `services/core/ObixHistoryPublish/config`

Registry Configuration File

Similar to a driver the Obix History Agent requires a registry file to select the points to publish.

The registry configuration file is a CSV file. Each row configures a point on the device.

The following columns are required for each row:

- **Device Name** - Name of the device to associate with this point.
- **Volttron Point Name** - The VOLTTRON Point name to use when publishing this value.
- **Obix Name** - Name of the point on the Obix interface. Escaping of spaces and dashes for use with the interface is handled internally.

Any additional columns will be ignored. It is common practice to include a *Notes* or *Unit Details* for additional information about a point.

The following is an example of a Obix History Agent registry configuration file:

Table 4: Obix

Device Name	Volttron Point Name	Obix Name
device1	Local Outside Dry Bulb	Local Outside Dry Bulb
device2	CG-1 Gas Flow F-2	CG-1 Gas Flow F-2
device2	Cog Plant Gas Flow F-1	Cog Plant Gas Flow F-1
device2	Boiler Plant Hourly Gas Usage	Boiler Plant Hourly Gas Usage
device3	CG-1 Water Flow H-1	CG-1 Water Flow H-1

A sample Obix History Agent configuration can be found in the VOLTTRON repository in *services/core/ObixHistoryPublish/registry_config.csv*

Automatic Obix Configuration File Creation

A script that will automatically create both a device and register configuration file for a site is located in the repository at *scripts/obix/get_obix_history_config.py*.

The utility is invoked with the command:

```
python get_obix_history_config.py <url> <registry_file> <driver_file> -u <username> -
↪p <password> -d <device name>
```

If either the registry_file or driver_file is omitted the script will output those files to stdout.

If either the username or password options are left out the script will ask for them on the command line before proceeding.

The device name option specifies a default device for every point in the configuration.

The registry file produced by this script assumes that the *Volttron Point Name* and the *Obix Name* have the same value. Also, it is assumed that all points should be read only. Users are expected to fix this as appropriate.

1.21.8 OpenADR 2.0b VEN Agent

OpenADR (Automated Demand Response) is a standard for alerting and responding to the need to adjust electric power consumption in response to fluctuations in grid demand.

OpenADR communications are conducted between Virtual Top Nodes (VTNs) and Virtual End Nodes (VENs). In this implementation a VOLTTRON agent, the VEN agent, acts as a VEN, communicating with its VTN by means of EIEvent and EIReport services in conformance with a subset of the OpenADR 2.0b specification. This document's VOLTTRON Interface section defines how the VEN agent relays information to, and receives data from, other VOLTTRON agents.

The OpenADR 2.0b specification (http://www.openadr.org/specification) is available from the OpenADR Alliance. This implementation also generally follows the DR program characteristics of the Capacity Program described in Section 9.2 of the OpenADR Program Guide (http://www.openadr.org/assets/openadr_drprogramguide_v1.0.pdf).

DR Capacity Bidding and Events

The OpenADR Capacity Bidding program relies on a pre-committed agreement about the VEN's load shed capacity. This agreement is reached in a bidding process transacted outside of the OpenADR interaction, typically with a long-term scope, perhaps a month or longer. The VTN can "call an event," indicating that a load-shed event should occur in conformance with this agreement. The VTN indicates the level of load shedding desired, when the event should occur, and for how long. The VEN responds with an *optIn* acknowledgment. (It can also *optOut*, but since it has been pre-committed, an *optOut* may incur penalties.)

Reporting

The VEN agent reports device status and usage telemetry to the VTN, relying on information received periodically from other VOLTTRON agents.

General Approach

Events:

- The VEN agent maintains a persistent record of DR events.
- Event updates (including creation) trigger publication of event JSON on the VOLTTRON message bus.
- Other VOLTTRON agents can also call a get_events() RPC to retrieve the current status of particular events, or of all active events.

Reporting:

- The VEN agent configuration defines telemetry values (data points) that can be reported to the VTN.
- The VEN agent maintains a persistent record of telemetry values over time.
- Other VOLTTRON agents are expected to call report_telemetry() to supply the VEN agent with a regular stream of telemetry values for reporting.
- Other VOLTTRON agents can receive notification of changes in telemetry reporting requirements by subscribing to publication of telemetry parameters.

VEN Agent VOLTTRON Interface

The VEN agent implements the following VOLTTRON PubSub and RPC calls.

PubSub: event update

```
def publish_event(self, an_event):
    """
        Publish an event.

        When an event is created/updated, it is published to the VOLTTRON bus
        with a topic that includes 'openadr/event_update'.

        Event JSON structure:
            {
                "event_id"       : String,
                "creation_time" : DateTime,
                "start_time"    : DateTime,
                "end_time"      : DateTime or None,
                "signals"       : String,     # Values: json string describing one or␣
↪more signals.
                "status"        : String,     # Values: unresponded, far, near,␣
↪active,
                                              #         completed, canceled.
                "opt_type"      : String      # Values: optIn, optOut, none.
            }

        If an event status is 'unresponded', the VEN agent is awaiting a decision on
        whether to optIn or optOut. The downstream agent that subscribes to this␣
↪PubSub
        message should communicate that choice to the VEN agent by calling respond_to_
↪event()
        (see below). The VEN agent then relays the choice to the VTN.

    @param an_event: an EiEvent.
    """
```

PubSub: telemetry parameters update

```
def publish_telemetry_parameters_for_report(self, report):
    """
        Publish telemetry parameters.

        When the VEN agent telemetry reporting parameters have been updated (by the␣
↪VTN),
        they are published with a topic that includes 'openadr/telemetry_parameters'.
        If a particular report has been updated, the reported parameters are for that␣
↪report.
```

(continues on next page)

(continued from previous page)

```
        Telemetry parameters JSON example:
        {
            "telemetry": {
                "baseline_power_kw": {
                    "r_id": "baseline_power",
                    "frequency": "30",
                    "report_type": "baseline",
                    "reading_type": "Mean",
                    "method_name": "get_baseline_power"
                }
                "current_power_kw": {
                    "r_id": "actual_power",
                    "frequency": "30",
                    "report_type": "reading",
                    "reading_type": "Mean",
                    "method_name": "get_current_power"
                }
                "manual_override": "False",
                "report_status": "active",
                "online": "False",
            }
        }

        The above example indicates that, for reporting purposes, telemetry values
        for baseline_power and actual_power should be updated -- via report_
↪telemetry() -- at
        least once every 30 seconds.

        Telemetry value definitions such as baseline_power and actual_power come from␣
↪the
        agent configuration.

    @param report: (EiReport) The report whose parameters should be published.
    """
```

RPC calls:

```
@RPC.export
def respond_to_event(self, event_id, opt_in_choice=None):
    """
        Respond to an event, opting in or opting out.

        If an event's status=unresponded, it is awaiting this call.
        When this RPC is received, the VENAgent sends an eventResponse to
        the VTN, indicating whether optIn or optOut has been chosen.
        If an event remains unresponded for a set period of time,
        it times out and automatically optsIn to the event.

        Since this call causes a change in the event's status, it triggers
        a PubSub call for the event update, as described above.

    @param event_id: (String) ID of an event.
    @param opt_in_choice: (String) 'OptIn' to opt into the event, anything else is␣
↪treated as 'OptOut'.
    """
```

```
@RPC.export
def get_events(self, event_id=None, in_progress_only=True, started_after=None, end_
↪time_before=None):
    """
        Return a list of events as a JSON string.

        Sample request:
            self.get_events(started_after=utils.get_aware_utc_now() -␣
↪timedelta(hours=1),
                            end_time_before=utils.get_aware_utc_now())

        Return a list of events.

        By default, return only event requests with status=active or␣
↪status=unresponded.

        If an event's status=active, a DR event is currently in progress.

    @param event_id: (String) Default None.
    @param in_progress_only: (Boolean) Default True.
    @param started_after: (DateTime) Default None.
    @param end_time_before: (DateTime) Default None.
    @return: (JSON) A list of events -- see 'PubSub: event update'.
    """
```

```
@RPC.export
def get_telemetry_parameters(self):
    """
        Return the VEN agent's current set of telemetry parameters.

    @return: (JSON) Current telemetry parameters -- see 'PubSub: telemetry parameters␣
↪update'.
    """
```

```
@RPC.export
def set_telemetry_status(self, online, manual_override):
    """
        Update the VEN agent's reporting status.

        Set these properties to either 'TRUE' or 'FALSE'.

    @param online: (Boolean) Whether the VEN agent's resource is online.
    @param manual_override: (Boolean) Whether resource control has been overridden.
    """
```

```
@RPC.export
def report_telemetry(self, telemetry):
    """
        Receive an update of the VENAgent's report metrics, and store them in the␣
↪agent's database.

        Examples of telemetry are:
        {
            'baseline_power_kw': '15.2',
            'current_power_kw': '371.1',
            'start_time': '2017-11-21T23:41:46.051405',
```

(continues on next page)

(continued from previous page)

```
            'end_time': '2017-11-21T23:42:45.951405'
        }

    @param telemetry_values: (JSON) Current value of each report metric, with␣
↪reporting-interval start/end.
    """
```

PubSub: Event Update

When an event is created/updated, the event is published with a topic that includes *openadr/event/{ven_id}*.

Event JSON structure:

```
{
    "event_id"      : String,
    "creation_time" : DateTime - UTC,
    "start_time"    : DateTime - UTC,
    "end_time"      : DateTime - UTC,
    "priority"      : Integer,     # Values: 0, 1, 2, 3. Usually expected to be 1.
    "signals"       : String,      # Values: json string describing one or more␣
↪signals.
    "status"        : String,      # Values: unresponded, far, near, active, completed,
↪ canceled.
    "opt_type"      : String       # Values: optIn, optOut, none.
}
```

If an event status is 'unresponded', the VEN is awaiting a decision on whether to *optIn* or *optOut*. The downstream agent that subscribes to this *PubSub* message should communicate that choice to the VEN by calling respond_to_event() (see below). The VEN then relays the choice to the VTN.

PubSub: Telemetry Parameters Update

When the VEN telemetry reporting parameters have been updated (by the VTN), they are published with a topic that includes *openadr/status/{ven_id}*.

These parameters include state information about the current report.

Telemetry parameters structure:

```
{
    'telemetry': '{
        "baseline_power_kw": {
            "r_id"            : "baseline_power",       # ID of the reporting metric
            "report_type"     : "baseline",             # Type of reporting metric, e.
↪g. baseline or reading
            "reading_type"    : "Direct Read",          # (per OpenADR telemetry_
↪usage report requirements)
            "units"           : "powerReal",            # (per OpenADR telemetry_
↪usage reoprt requirements)
            "method_name"     : "get_baseline_power",   # Name of the VEN agent␣
↪method that gets the metric
            "min_frequency"   : (Integer),              # Data capture frequency in␣
↪seconds (minimum)
            "max_frequency"   : (Integer)               # Data capture frequency in␣
↪seconds (maximum)
```

(continues on next page)

(continued from previous page)

```
        },
        "current_power_kw": {
            "r_id"              : "actual_power",         # ID of the reporting metric
            "report_type"       : "reading",              # Type of reporting metric, e.
↪g. baseline or reading
            "reading_type"      : "Direct Read",          # (per OpenADR telemetry_
↪usage report requirements)
            "units"             : "powerReal",            # (per OpenADR telemetry_
↪usage report requirements)
            "method_name"       : "get_current_power",    # Name of the VEN agent␣
↪method that gets the metric
            "min_frequency"     : (Integer),              # Data capture frequency in␣
↪seconds (minimum)
            "max_frequency"     : (Integer)               # Data capture frequency in␣
↪seconds (maximum)
        }
    }'
    'report parameters': '{
        "status"                : (String),               # active, inactive, completed,
↪ or cancelled
        "report_specifier_id" : "telemetry",              # ID of the report definition
        "report_request_id"     : (String),               # ID of the report request;␣
↪supplied by the VTN
        "request_id"            : (String),               # Request ID of the most␣
↪recent VTN report modification
        "interval_secs"         : (Integer),              # How often a report update␣
↪is sent to the VTN
        "granularity_secs"      : (Integer),              # How often a report update␣
↪is sent to the VTN
        "start_time"            : (DateTime - UTC),       # When the report started
        "end_time"              : (DateTime - UTC),       # When the report is␣
↪scheduled to end
        "last_report"           : (DateTime - UTC),       # When a report update was␣
↪last sent
        "created_on"            : (DateTime - UTC)        # When this set of␣
↪information was recorded in the VEN db
    }',
    'manual_override'           : (Boolean)               # VEN manual override status,␣
↪as supplied by Control Agent
    'online'                    : (Boolean)               # VEN online status, as␣
↪supplied by Control Agent
}
```

Telemetry value definitions such as *baseline_power_kw* and *current_power_kw* come from the VEN agent config.

OpenADR VEN Agent: Installation and Configuration

The VEN agent can be configured, built and launched using the VOLTTRON agent installation process described in http://volttron.readthedocs.io/en/develop/devguides/agent_development/Agent-Development.html#agent-development.

The VEN agent depends on some third-party libraries that are not in the standard VOLTTRON installation. They should be installed in the VOLTTRON virtual environment prior to building the agent:

```
(volttron) $ cd $VOLTTRON_ROOT/services/core/OpenADRVenAgent
(volttron) $ pip install -r requirements.txt
```

where `$VOLTTRON_ROOT` is the base directory of the cloned VOLTTRON code repository.

The VEN agent is designed to work in tandem with a "control agent," another VOLTTRON agent that uses VOLTTRON RPC calls to manage events and supply report data. A sample control agent has been provided in the *test/ControlAgentSim* subdirectory under OpenADRVenAgent.

The VEN agent maintains a persistent store of event and report data in `$VOLTTRON_HOME/data/openadr.sqlite`. Some care should be taken in managing the disk consumption of this data store. If no events or reports are active, it is safe to take down the VEN agent and delete the file; the persistent store will be reinitialized automatically on agent startup.

Configuration Parameters

The VEN agent's configuration file contains JSON that includes several parameters for configuring VTN server communications and other behavior. A sample configuration file, *openadrven.config*, has been provided in the agent directory.

The VEN agent supports the following configuration parameters:

Parameter	Example	Description
db_path	"$VOLTTRON_HOME/data/openadr.sqlite"	Pathname of the agent's sqlite database. Shell variables will be expanded if they are present in the pathname.
ven_id	"0"	The OpenADR ID of this virtual end node. Identifies this VEN to the VTN. If automated VEN registration is used, the ID is assigned by the VTN at that time. If the VEN is registered manually with the VTN (i.e., via configuration file settings), then a common VEN ID should be entered in this config file and in the VTN's site definition.
ven_name	"ven01"	Name of this virtual end node. This name is used during automated registration only, identiying the VEN before its VEN ID is known.
vtn_id	"vtn01"	OpenADR ID of the VTN with which this VEN communicates.
vtn_address	"http://openadr-vtn.ki-evi.com:8000"	URL and port number of the VTN.
send_registration	"False"	("True" or "False") If "True", the VEN sends a one-time automated registration request to the VTN to obtain the VEN ID. If automated registration will be used, the VEN should be run in this mode initially, then shut down and run with this parameter set to "False" thereafter.
security_level	"standard"	If 'high', the VTN and VEN use a third-party signing authority to sign and authenticate each request. The default setting is "standard": the XML payloads do not contain Signature elements.
poll_interval_secs	30	(integer) How often the VEN should send an OadrPoll request to the VTN. The poll interval cannot be more frequent than the VEN's 5-second process loop frequency.
log_xml	"False"	("True" or "False") Whether to write each inbound/outbound request's XML data to the agent's log.
opt_in_timeout_secs	1800	(integer) How long to wait before making a default optIn/optOut decision.
opt_in_default_decision	"optOut"	("True" or "False") Which optIn/optOut choice to make by default.
request_events_on_startup	"False"	("True" or "False") Whether to ask the VTN for a list of current events during VEN startup.
report_parameters	(see below)	A dictionary of definitions of reporting/telemetry parameters.

Reporting Configuration

The VEN's reporting configuration, specified as a dictionary in the agent configuration, defines each telemetry element (metric) that the VEN can report to the VTN, if requested. By default, it defines reports named "telemetry" and "telemetry_status", with a report configuration dictionary containing the following parameters:

"telemetry" report: parameters	Example	Description
report_name	"TELEMETRY_USAGE"	Friendly name of the report.
report_name_metadata	"METADATA_TELEMETRY_USAGE"	Friendly name of the report's metadata, when sent by the VEN's oadrRegisterReport request.
report_specifier_id	"telemetry"	Uniquely identifies the report's data set.
report_interval_secs_default	"300"	How often to send a reporting update to the VTN.
telemetry_parameters (baseline_power_kw): r_id	"baseline_power"	(baseline_power) Unique ID of the metric.
telemetry_parameters (baseline_power_kw): report_type	"baseline"	(baseline_power) The type of metric being reported.
telemetry_parameters (baseline_power_kw): reading_type	"Direct Read"	(baseline_power) How the metric was calculated.
telemetry_parameters (baseline_power_kw): units	"powerReal"	(baseline_power) The reading's data type.
telemetry_parameters (baseline_power_kw): method_name	"get_baseline_power"	(baseline_power) The VEN method to use when extracting the data for reporting.
telemetry_parameters (baseline_power_kw): min_frequency	30	(baseline_power) The metric's minimum sampling frequency.
telemetry_parameters (baseline_power_kw): max_frequency	60	(baseline_power) The metric's maximum sampling frequency.
telemetry_parameters (current_power_kw): r_id	"actual_power"	(current_power) Unique ID of the metric.
telemetry_parameters (current_power_kw): report_type	"reading"	(current_power) The type of metric being reported.
telemetry_parameters (current_power_kw): reading_type	"Direct Read"	(current_power) How the metric was calculated.
telemetry_parameters (current_power_kw): units	"powerReal"	(baseline_power) The reading's data type.
telemetry_parameters (current_power_kw): method_name	"get_current_power"	(current_power) The VEN method to use when extracting the data for reporting.
telemetry_parameters (current_power_kw): min_frequency	30	(current_power) The metric's minimum sampling frequency.
telemetry_parameters (current_power_kw): max_frequency	60	(current_power) The metric's maximum sampling frequency.

"telemetry_status" report: parameters	Example	Description
report_name	"TELEMETRY_STATUS"	Friendly name of the report.
report_name_metadata	"METADATA_TELEMETRY_STATUS"	Friendly name of the report's metadata, when sent by the VEN's oadrRegisterReport request.
report_specifier_id	"telemetry_status"	Uniquely identifies the report's data set.
report_interval_secs_default	"300"	How often to send a reporting update to the VTN.
telemetry_parameters (Status): r_id	"Status"	Unique ID of the metric.
telemetry_parameters (Status): report_type	"x-resourceStatus"	The type of metric being reported.
telemetry_parameters (Status): reading_type	"x-notApplicable"	How the metric was calculated.
telemetry_parameters (Status): units	""	The reading's data type.
telemetry_parameters (Status): method_name	""	The VEN method to use when extracting the data for reporting.
telemetry_parameters (Status): min_frequency	60	The metric's minimum sampling frequency.
telemetry_parameters (Status): max_frequency	120	The metric's maximum sampling frequency.

1.21.9 VOLTTRON Central Management Agent

Agent Introduction

The VOLTTRON Central Agent (VCM) is responsible for controlling multiple VOLTTRON instances through a single interfaces. The VOLTTRON instances can be either local or remote. VCM leverages an internal VOLTTRON web server providing a interface to our JSON-RPC based web API. Both the web api and the interface are served through the VCM agent. There is a *VOLTTRON Central Demo* that will allow you to quickly setup and see the current offerings of the interface. VOLTTRON Central will allow you to:

- See a list of platforms being managed.
- Add and remove platforms.
- Install, start and stop agents to the registered platforms.
- Create dynamic graphs from the historians based upon points.
- Execute functions on remote platforms.

Note: see *VCM JSON-RPC web API* for how the web interface works.

Instance Configuration

In order for any web agent to be enabled, there must be a port configured to serve the content. The easiest way to do this is to create a config file in the root of your VOLTTRON_HOME directory (to do this automatically see *VOLTTRON Config*.)

The following is an example of the configuration file

```
[volttron]
vip-addres=tcp://127.0.0.1:22916
bind-web-address=http://127.0.0.1:8080/vc/
```

Verify that the instance is serving properly by pointing your web browser to `http://127.0.0.1:8080/discovery/`

This is the required information for a VolttronCentralPlatform to be able to be managed.

VOLTTRON Central Manager Configuration

The following is the default configuration file for VOLTTRON Central:

```
{
    # The agentid is used during display on the VOLTTRON central platform
    # it does not need to be unique.
    "agentid": "volttron central",

    # Authentication for users is handled through a naive password algorithm
    # Note in the following example the user and password are both admin.

    # DO NOT USE IN PRODUCTION ENVIRONMENT!

    # import hashlib
    # hashlib.sha512(password).hexdigest() where password is the plain text password.
    "users" : {
        "reader" : {
            "password" :
↪"2d7349c51a3914cd6f5dc28e23c417ace074400d7c3e176bcf5da72fdbeb6ce7ed767ca00c6c1fb754b8df5114fc0b903
↪",
            "groups" : [
                "reader"
            ]
        },
        "writer" : {
            "password" :
↪"f7c31a682a838bbe0957cfa0bb060daff83c488fa5646eb541d334f241418af3611ff621b5a1b0d327f1ee80da25e0409
↪",
            "groups" : [
                "writer"
            ]
        },
        "admin" : {
            "password" :
↪"c7ad44cbad762a5da0a452f9e854fdc1e0e7a52a38015f23f3eab1d80b931dd472634dfac71cd34ebc35d16ab7fb8a90c
↪",
            "groups" : [
                "admin"
            ]
        },
        "dorothy" : {
            "password" :
↪"cf1b67402d648f51ef6ff8805736d588ca07cbf018a5fba404d28532d839a1c046bfcd31558dff658678b3112502f4da9
↪",
            "groups" : [
```

(continues on next page)

(continued from previous page)

```
                "reader, writer"
            ]
        }
    }
}
```

Agent Execution

To start VOLTTRON Central first make sure the *VOLTTRON instance is running*. Next create/choose the config file to use. Finally from an activated shell in the root of the VOLTTRON repository execute:

```
# Arguments are package to execute, config file to use, tag to use as reference
./scripts/core/pack_install.sh services/core/VolttronCentral services/core/
↪VolttronCentral/config vc

# Start the agent
vctl start --tag vc
```

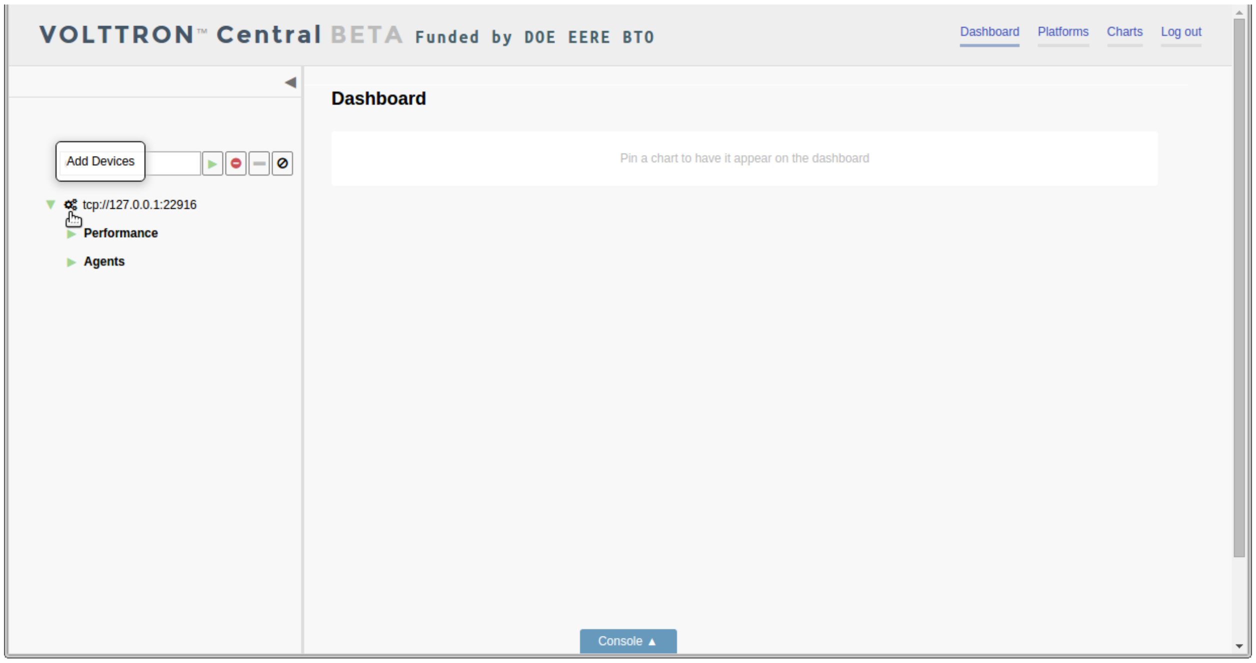
Device Configuration in VOLTTRON Central

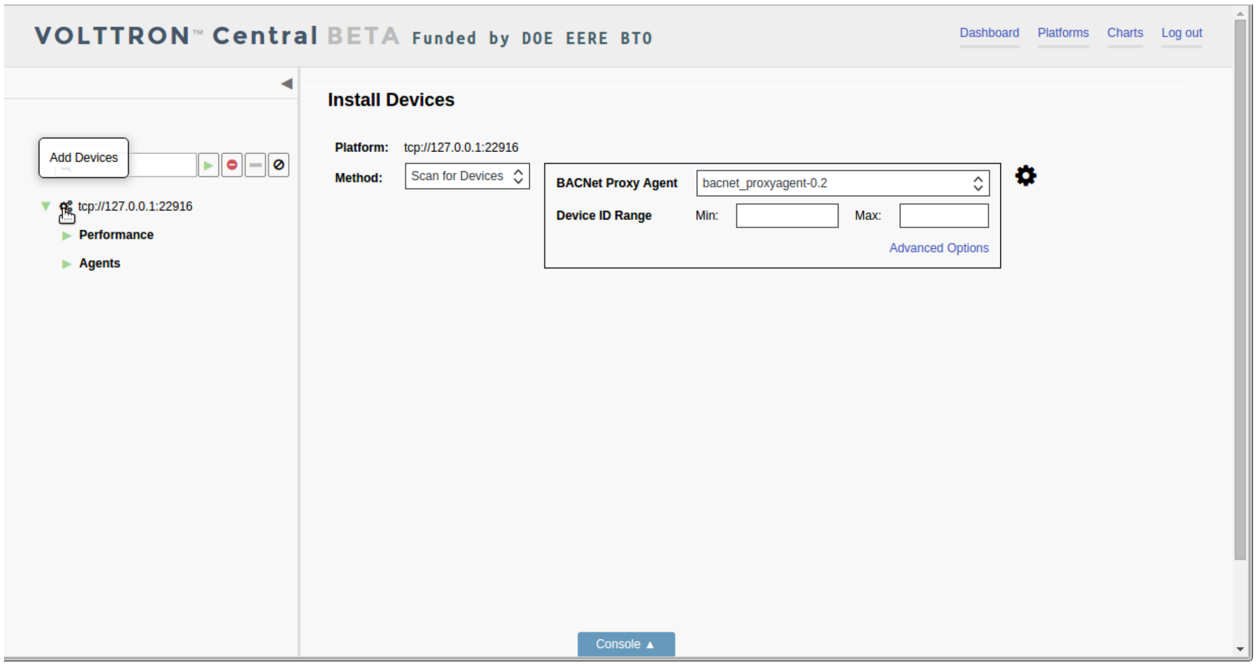
Devices in your network can be detected and configured through the VOLTTRON Central UI. The current version of VOLTTRON enables device detection and configuration for BACnet devices. The following sections describe the processes involved with performing scans to detect physical devices and get their points, and configuring them as virtual devices installed on VOLTTRON instances.

- Launching Device Configuration
- Scanning for Devices
- Scanning for Points
- Registry Configuration File
- Additional Attributes
- Quick Edit Features
- Keyboard Commands
- Registry Preview
- Registry Configuration Options
- Reloading Device Points
- Device Configuration Form
- Configuring Sub-devices
- Reconfiguring Devices
- Exporting Registry Configuration Files

Launching Device Configuration

To begin device configuration in VOLTTRON Central, extend the side panel on the left and find the cogs button next to the platform instance you want to add a device to. Click the cogs button to launch the device configuration feature.





Install Devices

Platform: tcp://127.0.0.1:22916

Method: Scan for Devices

BACNet Proxy Agent: bacnet_proxyagent-0.2

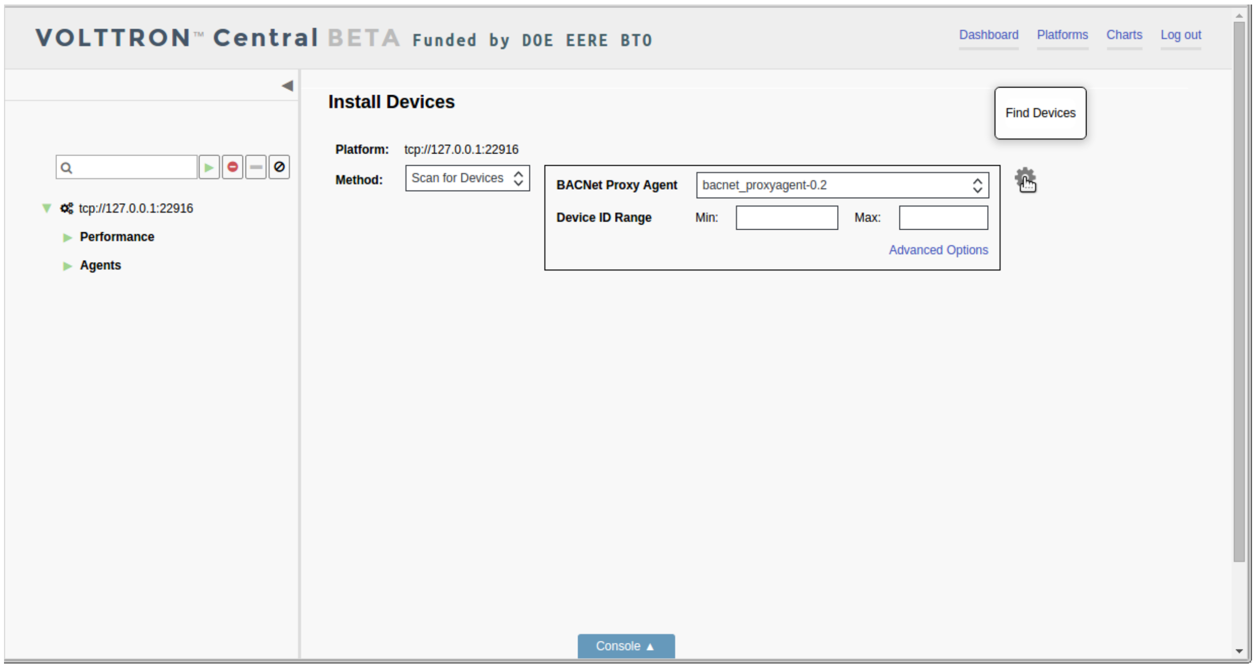
Device ID Range Min: Max:

Currently the only method of adding devices is to conduct a scan to detect BACnet devices. A BACnet Proxy Agent must be running in order to do the scan. If more than one BACnet Proxy is installed on the platform, choose the one that will be used for the scan.

The scan can be conducted using default settings that will search for all physical devices on the network. However, optional settings can be used to focus on specific devices or change the duration of the scan. Entering a range of device IDs will limit the scan to return only devices with IDs in that range. Advanced options include the ability to specify the IP address of a device to detect as well as the ability to change the duration of the scan from the default of five seconds.

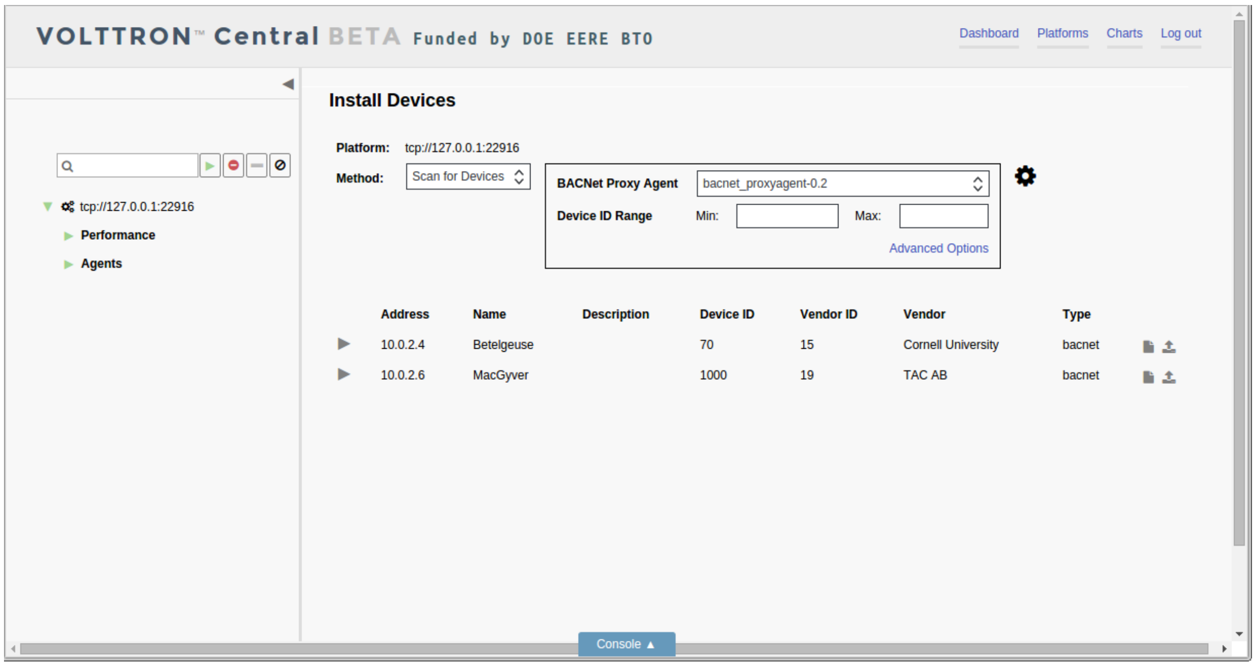
Scanning for Devices

To start the scan, click the large cog button to the right of the scan settings.



Platform: tcp://127.0.0.1:22916
Method: Scan for Devices
BACNet Proxy Agent: bacnet_proxyagent-0.2
Device ID Range Min: Max:

Devices that are detected will appear in the space below the scan settings. Scanning can be repeated at any time by clicking the large cog button again.

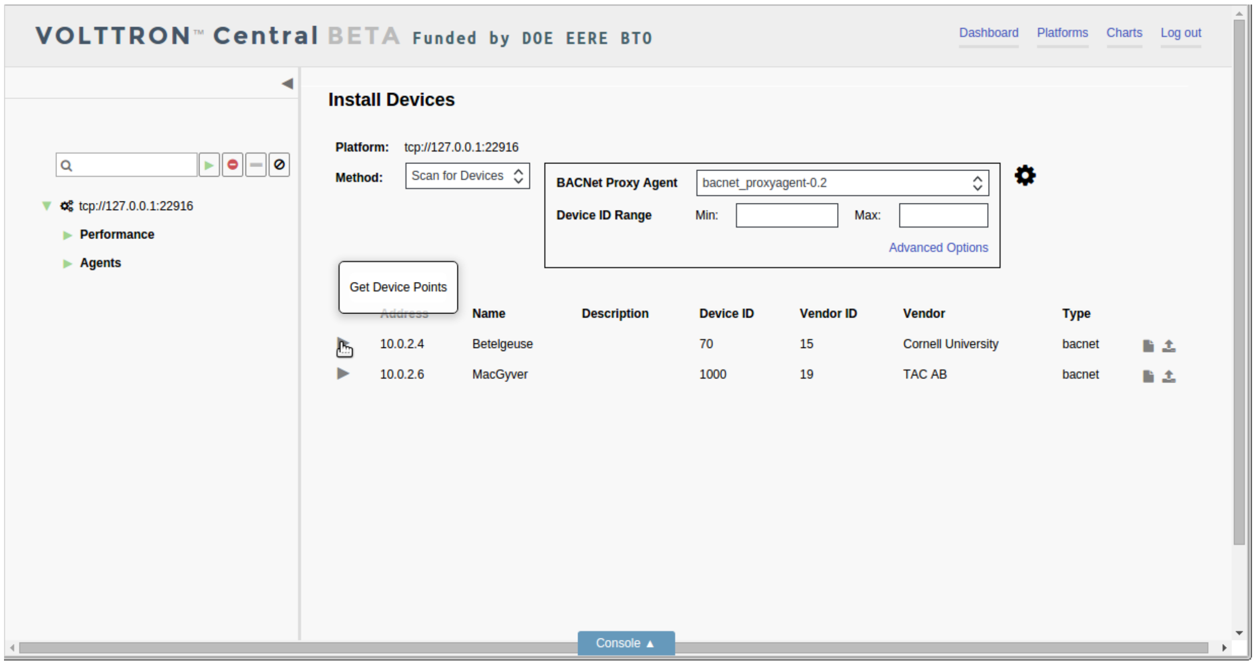


Platform: tcp://127.0.0.1:22916
Method: Scan for Devices
BACNet Proxy Agent: bacnet_proxyagent-0.2
Device ID Range Min: Max:

Address	Name	Description	Device ID	Vendor ID	Vendor	Type
10.0.2.4	Betelgeuse		70	15	Cornell University	bacnet
10.0.2.6	MacGyver		1000	19	TAC AB	bacnet

Scanning for Points

Another scan can be performed on each physical device to retrieve its available points. This scan is initiated by clicking the triangle next to the device in the list. The first time the arrow is clicked, it initiates the scan. After the points are retrieved, the arrow becomes a hide-and-show toggle button and won't re-initiate scanning the device.



Install Devices

Platform: tcp://127.0.0.1:22916

Method: Scan for Devices

BACNet Proxy Agent: bacnet_proxyagent-0.2

Device ID Range Min: Max:

Advanced Options

Address	Name	Description	Device ID	Vendor ID	Vendor	Type
10.0.2.4	Betelgeuse		70	15	Cornell University	bacnet
10.0.2.6	MacGyver		1000	19	TAC AB	bacnet

After the points have been retrieved once, the only way to scan the same device for points again is to relaunch the device configuration process from the start by clicking on the small cogs button next to the platform instance in the panel tree.

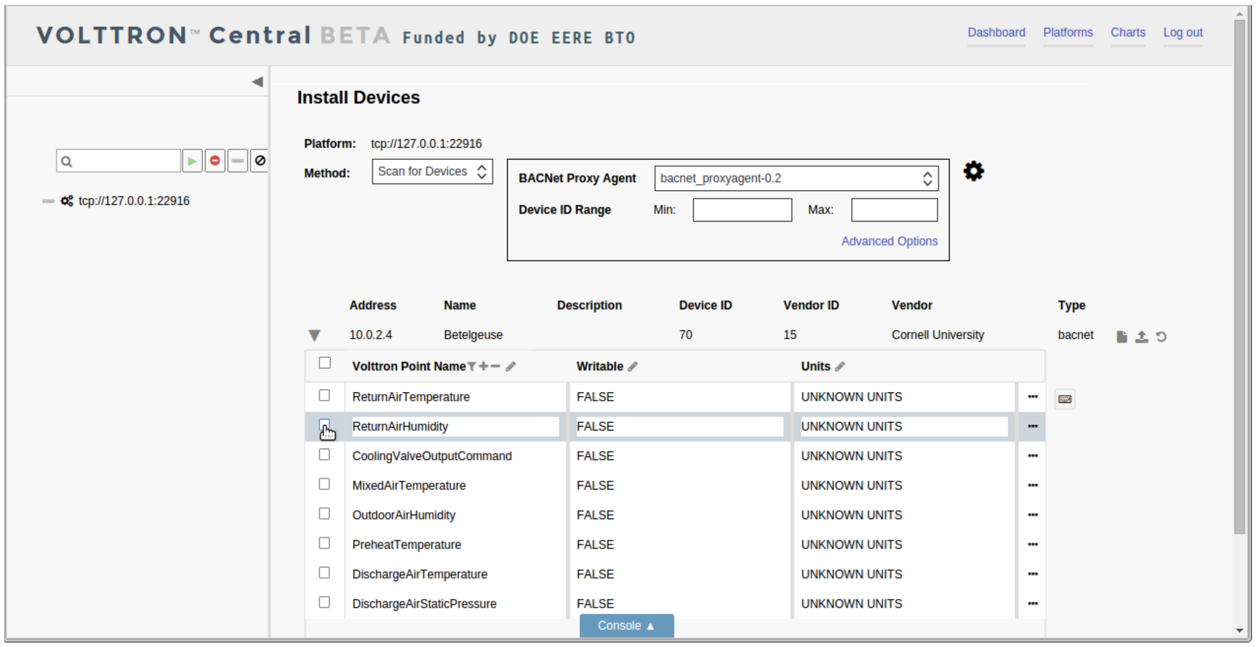
Registry Configuration File

The registry configuration determines which points on the physical device will be associated with the virtual device that uses that particular registry configuration. The registry configuration determines which points' data will be published to the message bus and recorded by the historian, and it determines how the data will be presented.

When all the points on the device have been retrieved, the points are loaded into the registry configuration editor. There, the points can be modified and selected to go into the registry configuration file for a device.

Each row in the registry configuration editor represents a point, and each cell in the row represents an attribute of the point.

Only points that have been selected will be included in the registry configuration file. To select a point, check the box next to the point in the editor.



Install Devices

Platform: tcp://127.0.0.1:22916

Method: Scan for Devices

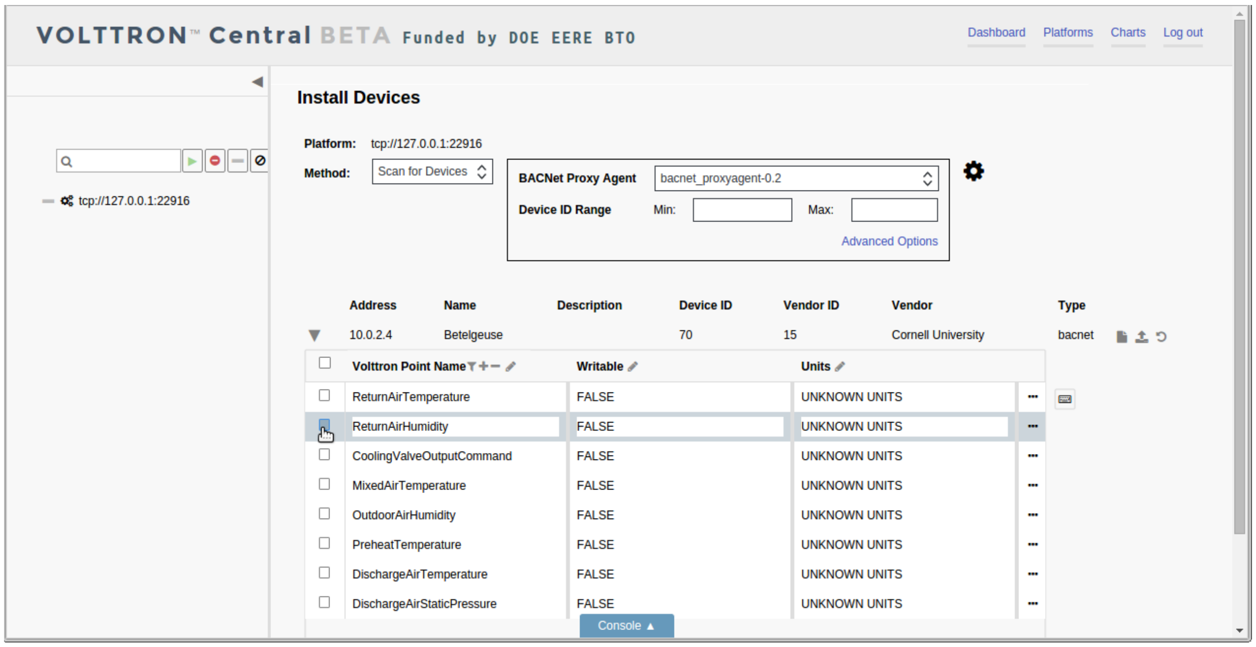
BACNet Proxy Agent: bacnet_proxyagent-0.2

Device ID Range Min: Max:

Advanced Options

Address	Name	Description	Device ID	Vendor ID	Vendor	Type
10.0.2.4	Betelgeuse		70	15	Cornell University	bacnet

Volttron Point Name	Writable	Units
ReturnAirTemperature	FALSE	UNKNOWN UNITS
ReturnAirHumidity	FALSE	UNKNOWN UNITS
CoolingValveOutputCommand	FALSE	UNKNOWN UNITS
MixedAirTemperature	FALSE	UNKNOWN UNITS
OutdoorAirHumidity	FALSE	UNKNOWN UNITS
PreheatTemperature	FALSE	UNKNOWN UNITS
DischargeAirTemperature	FALSE	UNKNOWN UNITS
DischargeAirStaticPressure	FALSE	UNKNOWN UNITS



Install Devices

Platform: tcp://127.0.0.1:22916

Method: Scan for Devices

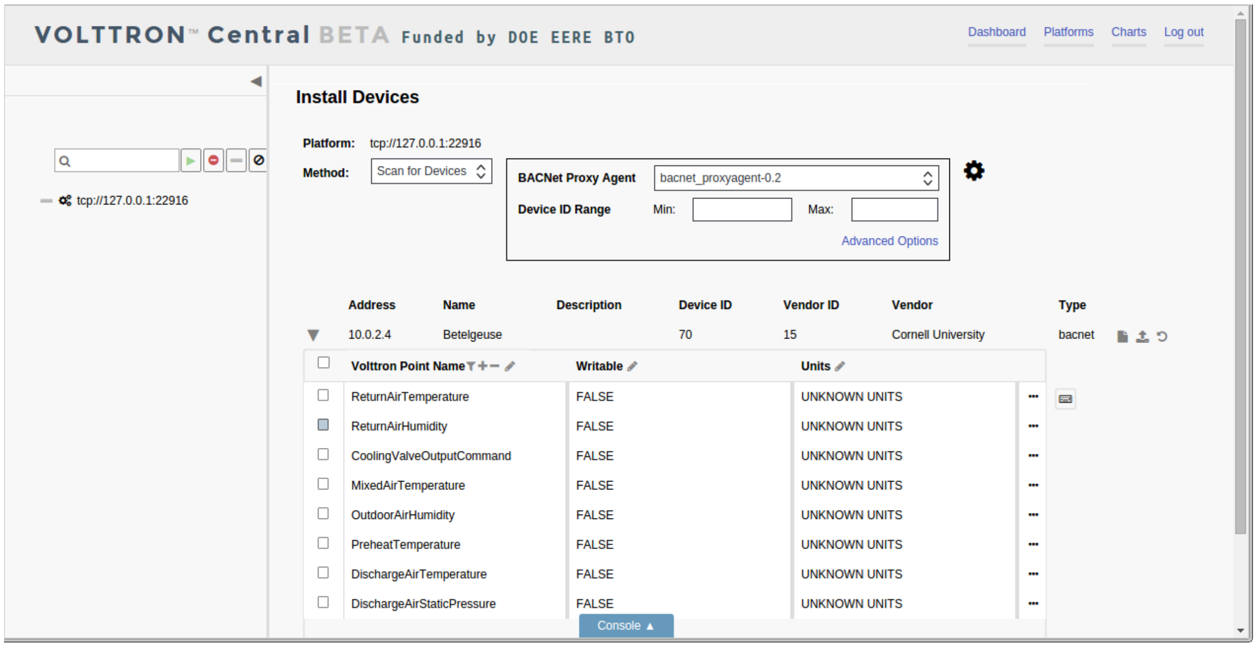
BACNet Proxy Agent: bacnet_proxyagent-0.2

Device ID Range Min: Max:

Advanced Options

Address	Name	Description	Device ID	Vendor ID	Vendor	Type
10.0.2.4	Betelgeuse		70	15	Cornell University	bacnet

Volttron Point Name	Writable	Units
ReturnAirTemperature	FALSE	UNKNOWN UNITS
ReturnAirHumidity	FALSE	UNKNOWN UNITS
CoolingValveOutputCommand	FALSE	UNKNOWN UNITS
MixedAirTemperature	FALSE	UNKNOWN UNITS
OutdoorAirHumidity	FALSE	UNKNOWN UNITS
PreheatTemperature	FALSE	UNKNOWN UNITS
DischargeAirTemperature	FALSE	UNKNOWN UNITS
DischargeAirStaticPressure	FALSE	UNKNOWN UNITS



Install Devices

Platform: tcp://127.0.0.1:22916

Method: Scan for Devices

BACNet Proxy Agent: bacnet_proxyagent-0.2

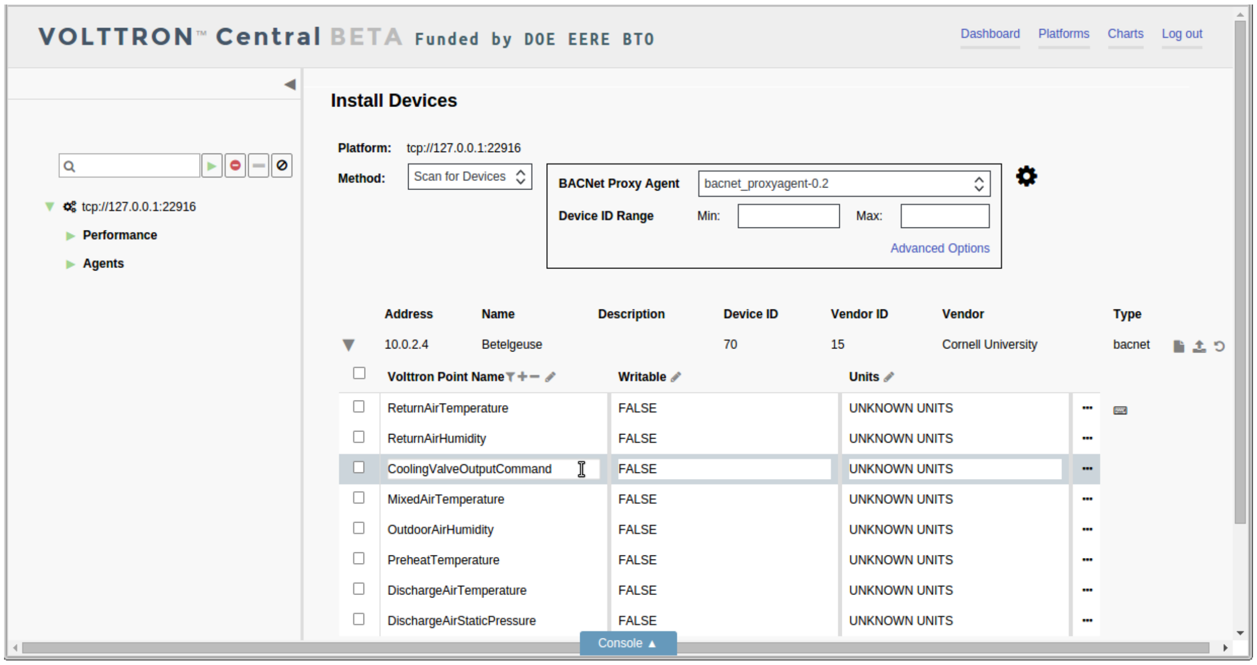
Device ID Range Min: Max:

Advanced Options

Address	Name	Description	Device ID	Vendor ID	Vendor	Type
10.0.2.4	Betelgeuse		70	15	Cornell University	bacnet

Volttron Point Name	Writable	Units
ReturnAirTemperature	FALSE	UNKNOWN UNITS
ReturnAirHumidity	FALSE	UNKNOWN UNITS
CoolingValveOutputCommand	FALSE	UNKNOWN UNITS
MixedAirTemperature	FALSE	UNKNOWN UNITS
OutdoorAirHumidity	FALSE	UNKNOWN UNITS
PreheatTemperature	FALSE	UNKNOWN UNITS
DischargeAirTemperature	FALSE	UNKNOWN UNITS
DischargeAirStaticPressure	FALSE	UNKNOWN UNITS

Type directly in a cell to change an attribute value for a point.



Install Devices

Platform: tcp://127.0.0.1:22916

Method: Scan for Devices

BACNet Proxy Agent: bacnet_proxyagent-0.2

Device ID Range Min: Max:

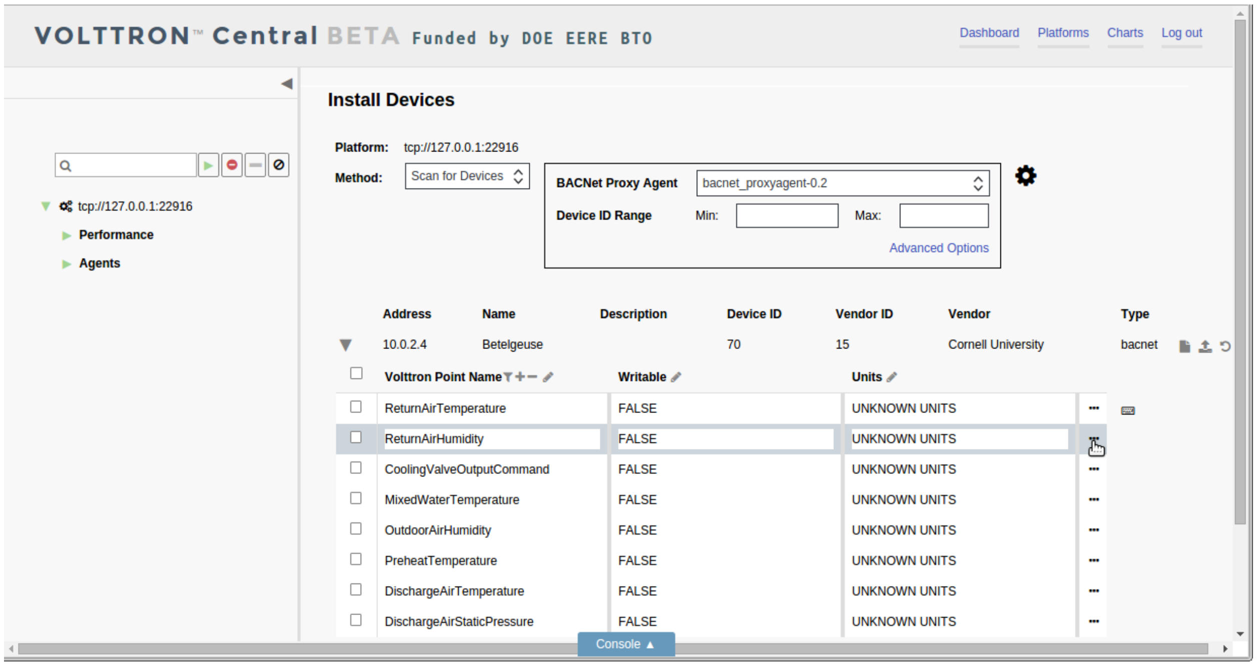
Advanced Options

Address	Name	Description	Device ID	Vendor ID	Vendor	Type
10.0.2.4	Betelgeuse		70	15	Cornell University	bacnet

Volttron Point Name	Writable	Units
ReturnAirTemperature	FALSE	UNKNOWN UNITS
ReturnAirHumidity	FALSE	UNKNOWN UNITS
CoolingValveOutputCommand	FALSE	UNKNOWN UNITS
MixedAirTemperature	FALSE	UNKNOWN UNITS
OutdoorAirHumidity	FALSE	UNKNOWN UNITS
PreheatTemperature	FALSE	UNKNOWN UNITS
DischargeAirTemperature	FALSE	UNKNOWN UNITS
DischargeAirStaticPressure	FALSE	UNKNOWN UNITS

Additional Attributes

The editor's default view shows the attributes that are most likely to be changed during configuration: the VOLTTRON point name, the writable setting, and the units. Other attributes are present but not shown in the default view. To see the entire set of attributes for a point, click the *Edit Point* button (the three dots) at the end of the point row.



Install Devices

Platform: tcp://127.0.0.1:22916

Method: Scan for Devices

BACNet Proxy Agent: bacnet_proxyagent-0.2

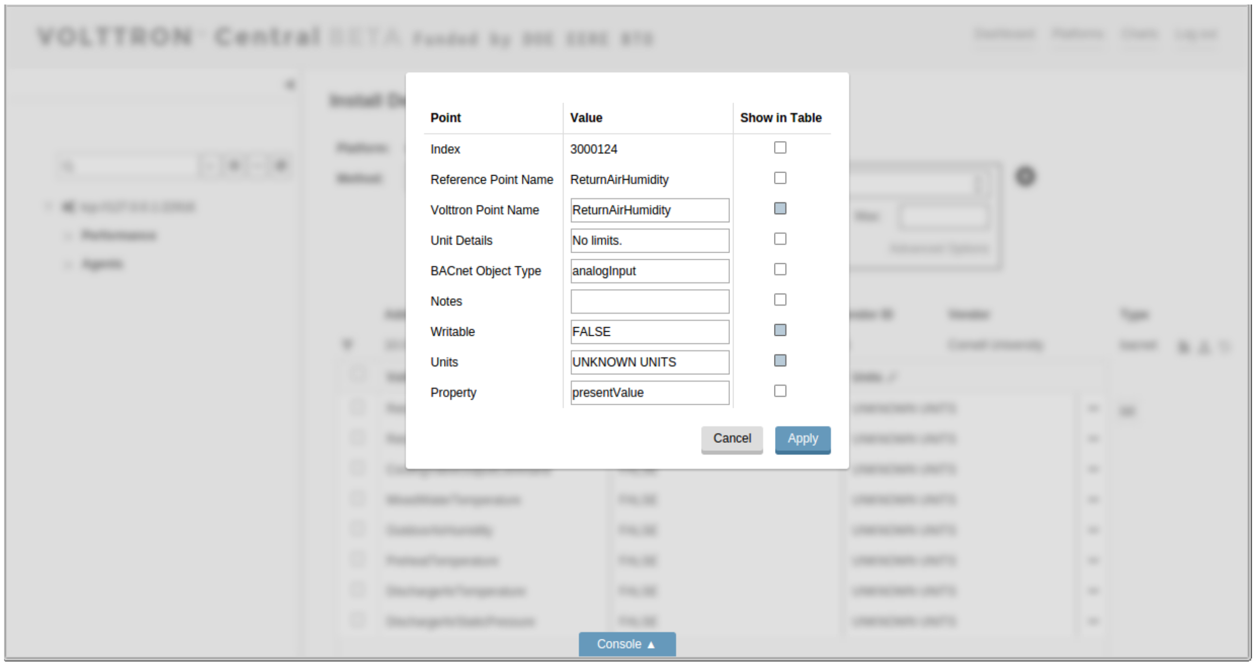
Device ID Range Min: Max:

Advanced Options

Address	Name	Description	Device ID	Vendor ID	Vendor	Type
10.0.2.4	Betelgeuse		70	15	Cornell University	bacnet

Volttron Point Name	Writable	Units
ReturnAirTemperature	FALSE	UNKNOWN UNITS
ReturnAirHumidity	FALSE	UNKNOWN UNITS
CoolingValveOutputCommand	FALSE	UNKNOWN UNITS
MixedWaterTemperature	FALSE	UNKNOWN UNITS
OutdoorAirHumidity	FALSE	UNKNOWN UNITS
PreheatTemperature	FALSE	UNKNOWN UNITS
DischargeAirTemperature	FALSE	UNKNOWN UNITS
DischargeAirStaticPressure	FALSE	UNKNOWN UNITS

In the window that opens, point attributes can be changed by typing in the fields and clicking the Apply button.



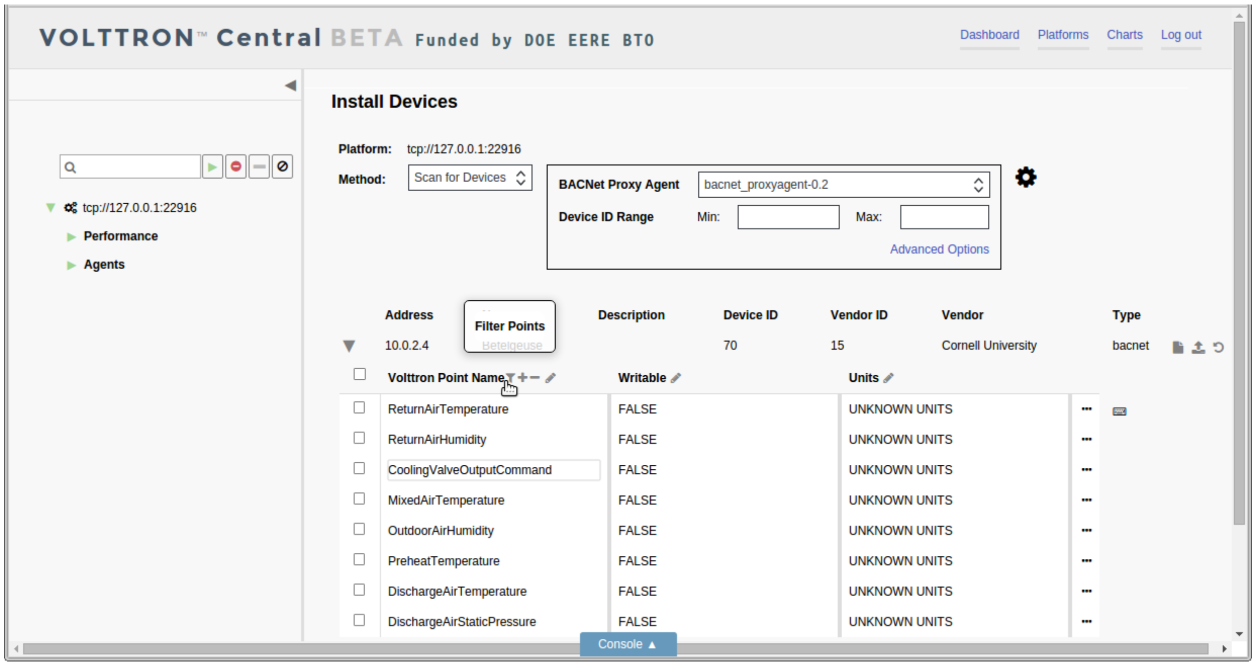
Point	Value	Show in Table
Index	3000124	☐
Reference Point Name	ReturnAirHumidity	☐
Volttron Point Name	ReturnAirHumidity	▨
Unit Details	No limits.	☐
BACnet Object Type	analogInput	☐
Notes		☐
Writable	FALSE	▨
Units	UNKNOWN UNITS	▨
Property	presentValue	☐

Checking or unchecking the *Show in Table* box for an attribute will add or remove it as a column in the registry configuration editor.

Quick Edit Features

Several quick-edit features are available in the registry configuration editor.

The list of points can be filtered based on values in the first column by clicking the filter button in the first column's header and entering a filter term.



Install Devices

Platform: tcp://127.0.0.1:22916

Method: Scan for Devices

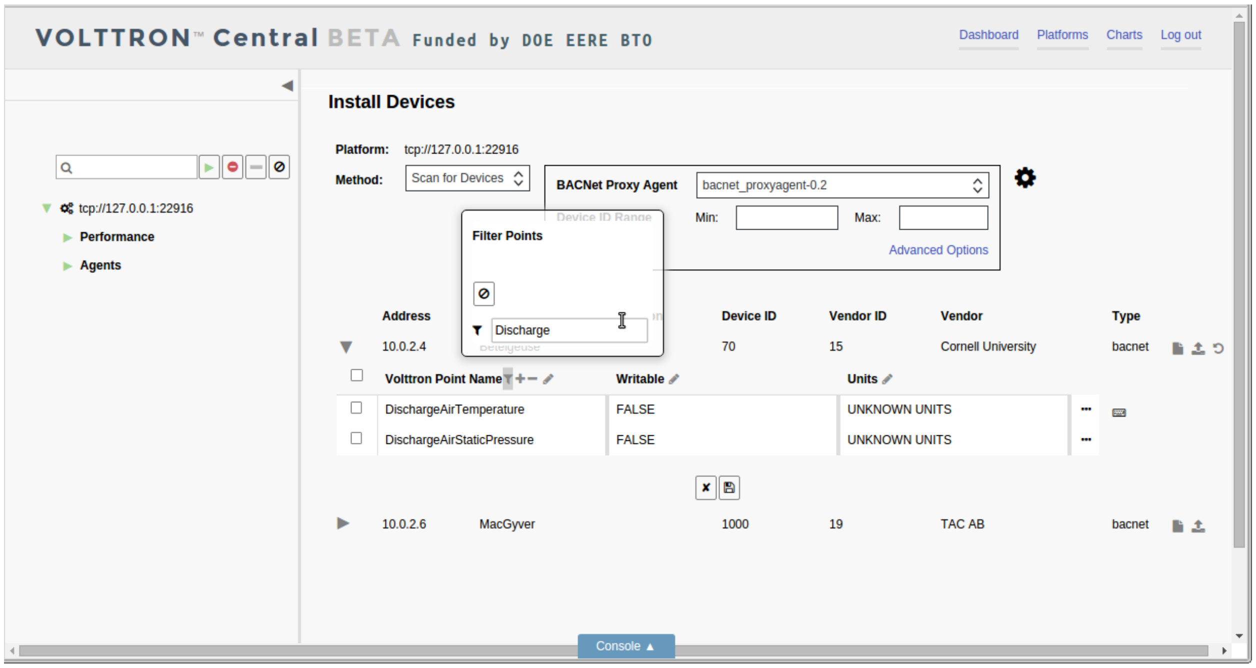
BACNet Proxy Agent: bacnet_proxyagent-0.2

Device ID Range Min: Max:

Advanced Options

Address	Name	Description	Device ID	Vendor ID	Vendor	Type
10.0.2.4	Betelgeuse		70	15	Cornell University	bacnet

Volttron Point Name	Writable	Units
ReturnAirTemperature	FALSE	UNKNOWN UNITS
ReturnAirHumidity	FALSE	UNKNOWN UNITS
CoolingValveOutputCommand	FALSE	UNKNOWN UNITS
MixedAirTemperature	FALSE	UNKNOWN UNITS
OutdoorAirHumidity	FALSE	UNKNOWN UNITS
PreheatTemperature	FALSE	UNKNOWN UNITS
DischargeAirTemperature	FALSE	UNKNOWN UNITS
DischargeAirStaticPressure	FALSE	UNKNOWN UNITS



Install Devices

Platform: tcp://127.0.0.1:22916

Method: Scan for Devices

BACNet Proxy Agent: bacnet_proxyagent-0.2

Device ID Range Min: Max:

Advanced Options

Filter Points: Discharge

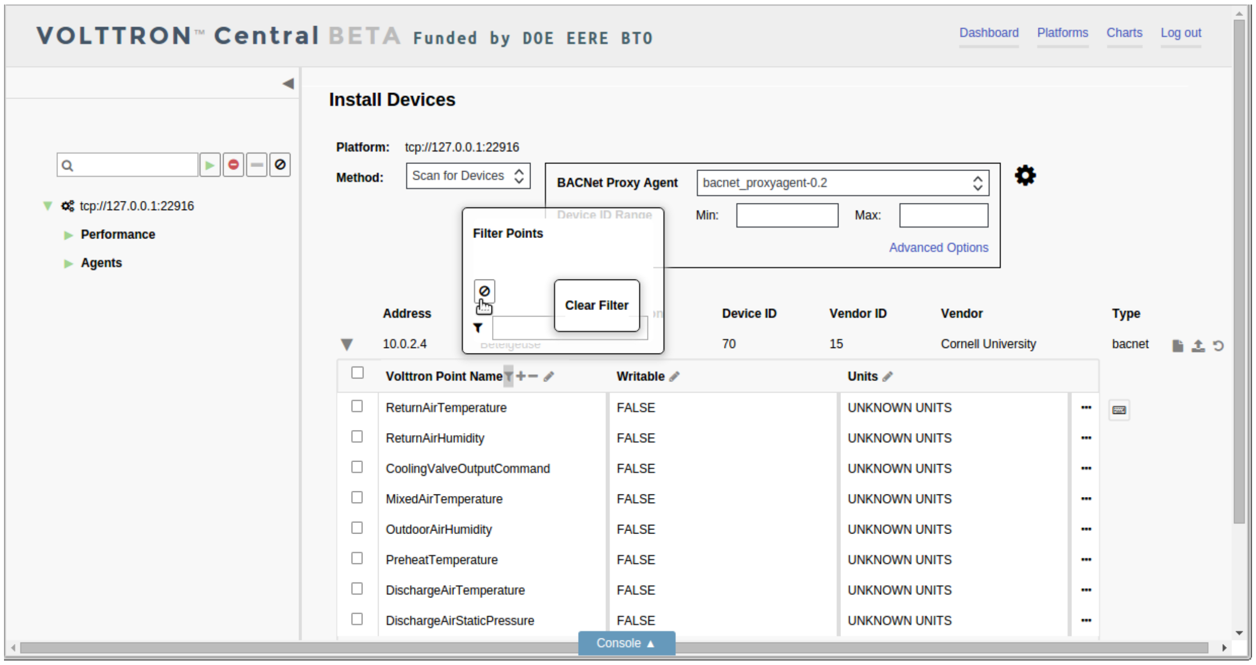
Address	Name	Description	Device ID	Vendor ID	Vendor	Type
10.0.2.4	Betelgeuse		70	15	Cornell University	bacnet

Volttron Point Name	Writable	Units
DischargeAirTemperature	FALSE	UNKNOWN UNITS
DischargeAirStaticPressure	FALSE	UNKNOWN UNITS

Address	Name	Description	Device ID	Vendor ID	Vendor	Type
10.0.2.6	MacGyver		1000	19	TAC AB	bacnet

The filter feature allows points to be edited, selected, or deselected more quickly by narrowing down potentially large lists of points. However, the filter doesn't select points, and if the registry configuration is saved while a filter is applied, any selected points not included in the filter will still be included in the registry file.

To clear the filter, click on the *Clear Filter* button in the filter popup.



Install Devices

Platform: tcp://127.0.0.1:22916

Method: Scan for Devices

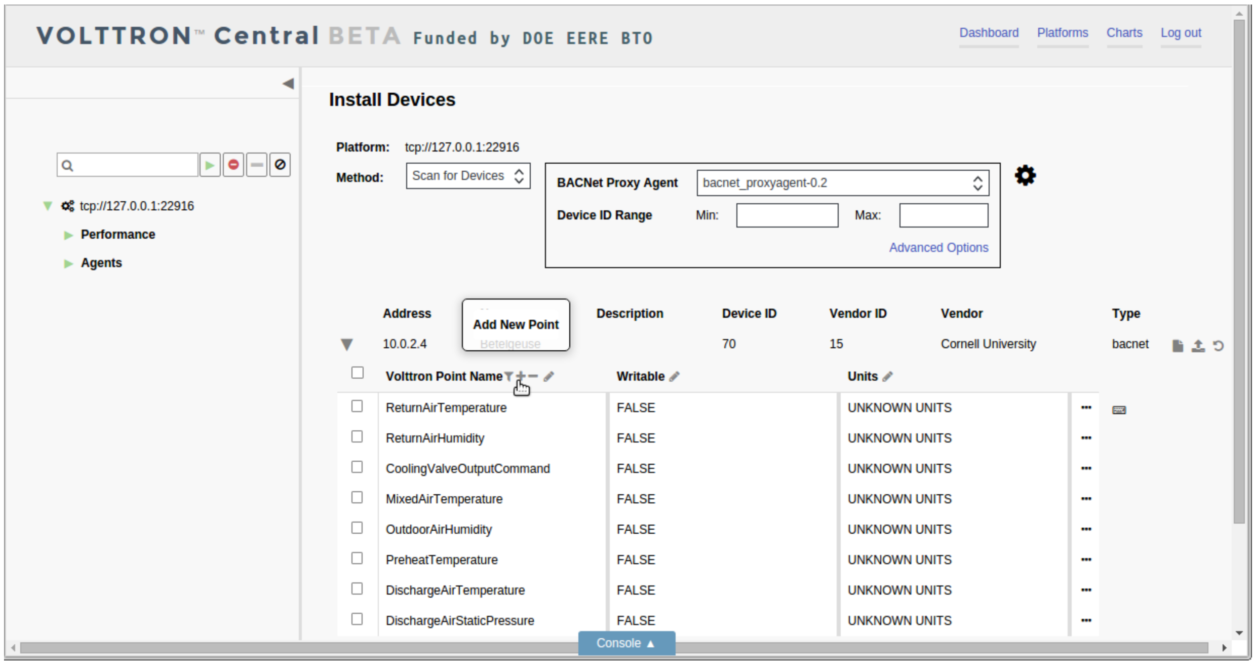
BACNet Proxy Agent: bacnet_proxyagent-0.2

Device ID Range Min: Max:

Address	Device ID	Vendor ID	Vendor	Type
10.0.2.4	70	15	Cornell University	bacnet

Volttron Point Name	Writable	Units
ReturnAirTemperature	FALSE	UNKNOWN UNITS
ReturnAirHumidity	FALSE	UNKNOWN UNITS
CoolingValveOutputCommand	FALSE	UNKNOWN UNITS
MixedAirTemperature	FALSE	UNKNOWN UNITS
OutdoorAirHumidity	FALSE	UNKNOWN UNITS
PreheatTemperature	FALSE	UNKNOWN UNITS
DischargeAirTemperature	FALSE	UNKNOWN UNITS
DischargeAirStaticPressure	FALSE	UNKNOWN UNITS

To add a new point to the points listed in the registry configuration editor, click on the *Add Point* button in the header of the first column.



Install Devices

Platform: tcp://127.0.0.1:22916

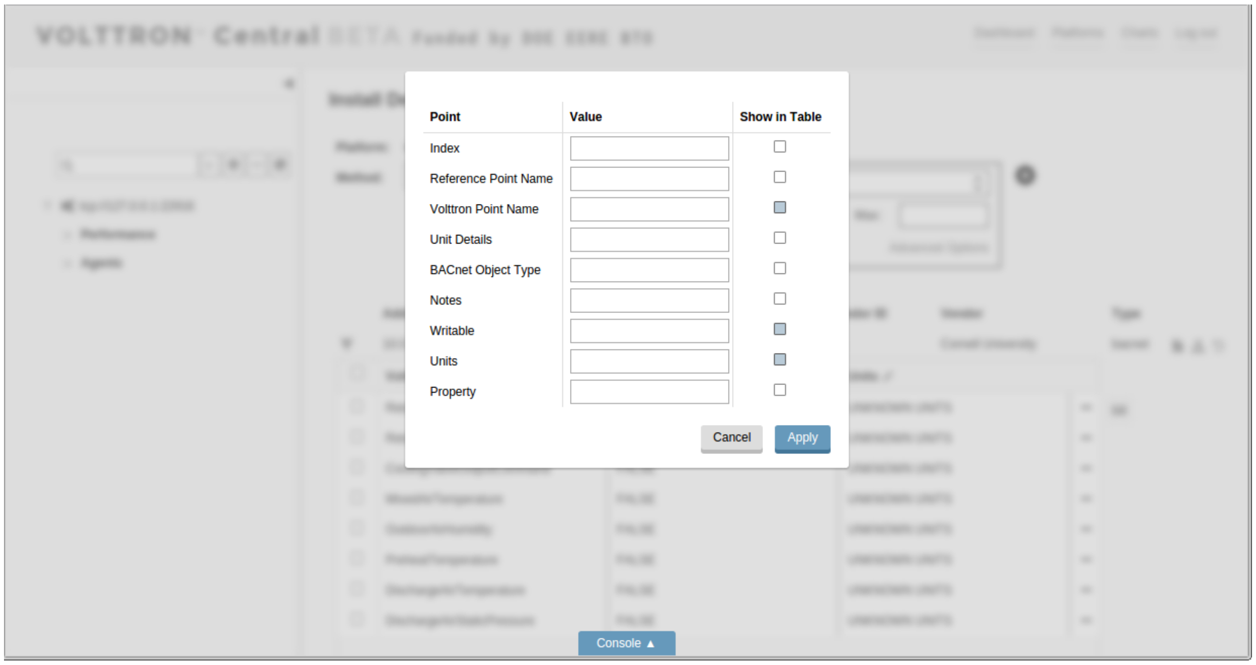
Method: Scan for Devices

BACNet Proxy Agent: bacnet_proxyagent-0.2

Device ID Range Min: Max:

Address	Description	Device ID	Vendor ID	Vendor	Type
10.0.2.4	Betelgeuse	70	15	Cornell University	bacnet

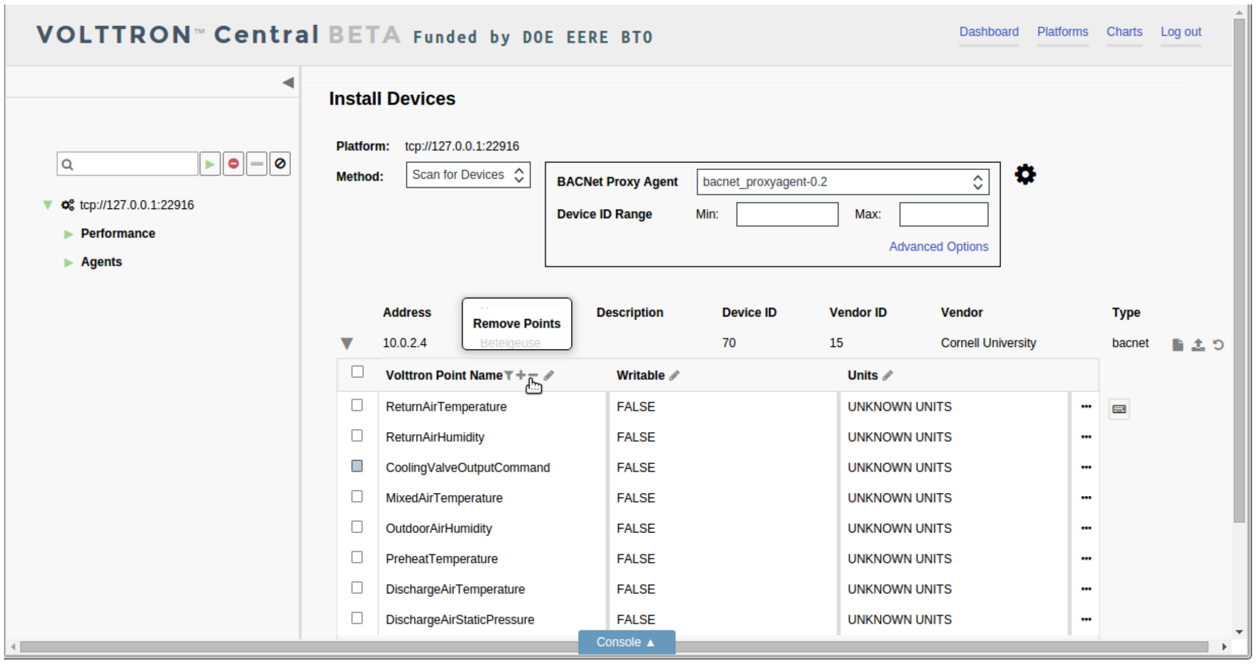
Volttron Point Name	Writable	Units
ReturnAirTemperature	FALSE	UNKNOWN UNITS
ReturnAirHumidity	FALSE	UNKNOWN UNITS
CoolingValveOutputCommand	FALSE	UNKNOWN UNITS
MixedAirTemperature	FALSE	UNKNOWN UNITS
OutdoorAirHumidity	FALSE	UNKNOWN UNITS
PreheatTemperature	FALSE	UNKNOWN UNITS
DischargeAirTemperature	FALSE	UNKNOWN UNITS
DischargeAirStaticPressure	FALSE	UNKNOWN UNITS



Point	Value	Show in Table
Index		☐
Reference Point Name		☐
Volttron Point Name		▨
Unit Details		☐
BACnet Object Type		☐
Notes		☐
Writable		▨
Units		▨
Property		☐

Provide attribute values, and click the *Apply* button to add the new point, which will be appended to the bottom of the list.

To remove points from the list, select the points and click the *Remove Points* button in the header of the first column.



Install Devices

Platform: tcp://127.0.0.1:22916

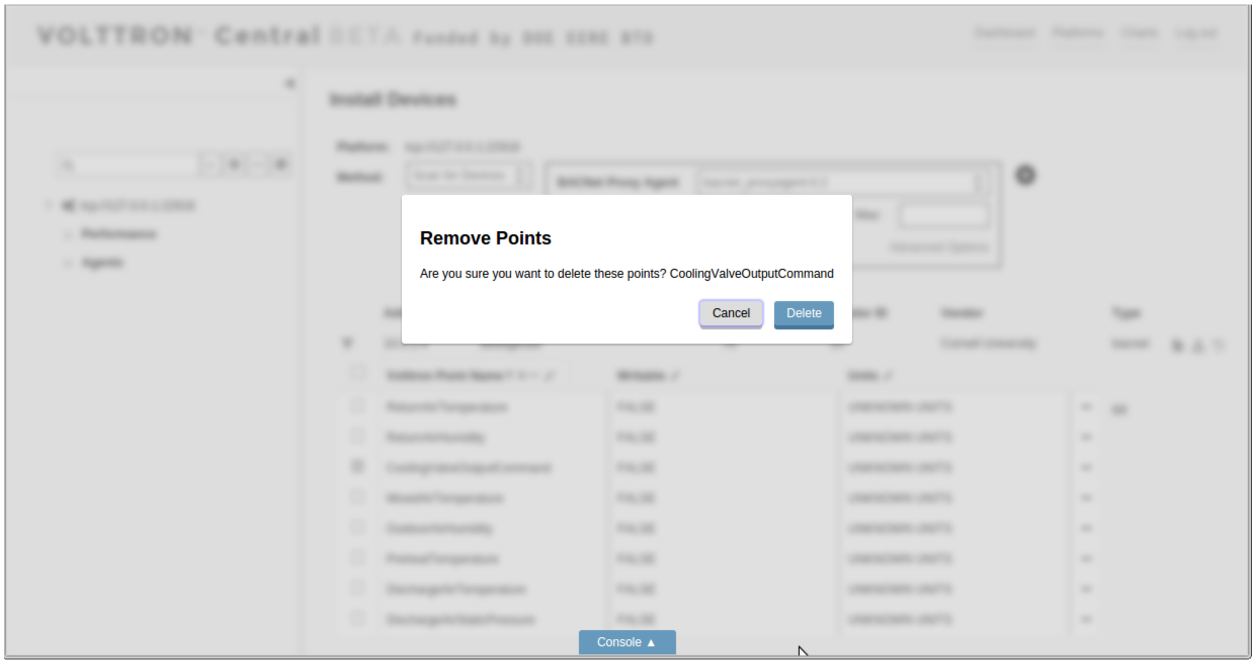
Method: Scan for Devices

BACNet Proxy Agent: bacnet_proxyagent-0.2

Device ID Range Min: Max:

Address	Description	Device ID	Vendor ID	Vendor	Type
10.0.2.4		70	15	Cornell University	bacnet

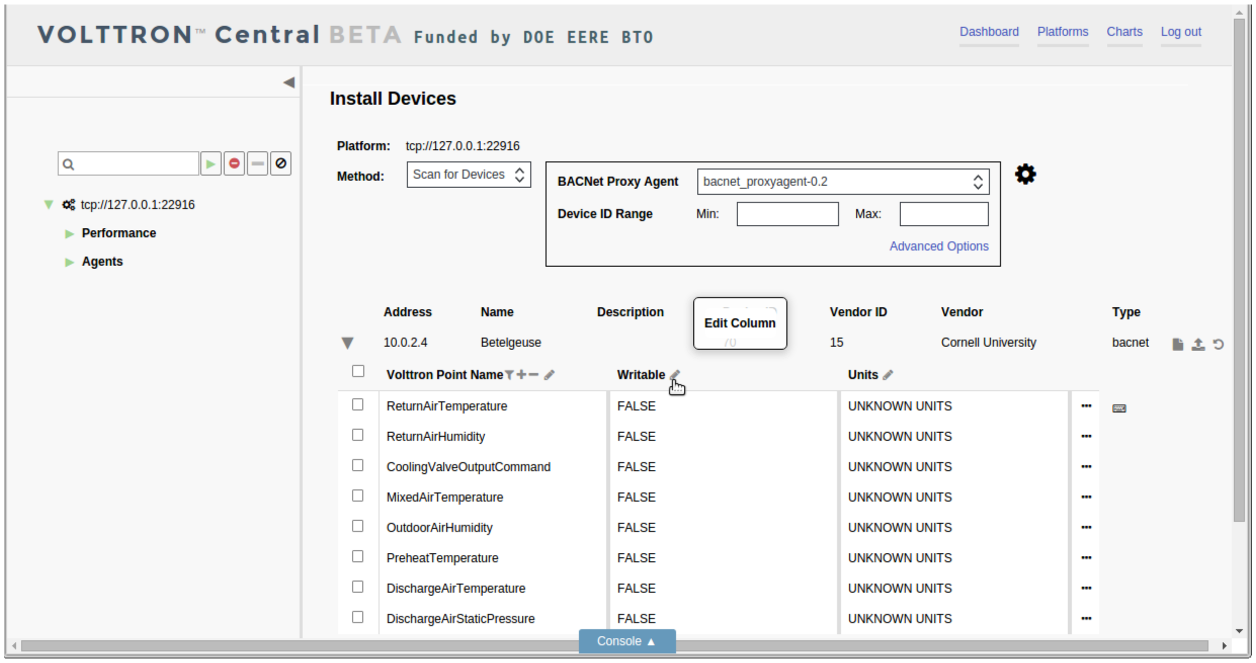
	Volttron Point Name	Writable	Units
☐	ReturnAirTemperature	FALSE	UNKNOWN UNITS
☐	ReturnAirHumidity	FALSE	UNKNOWN UNITS
▨	CoolingValveOutputCommand	FALSE	UNKNOWN UNITS
☐	MixedAirTemperature	FALSE	UNKNOWN UNITS
☐	OutdoorAirHumidity	FALSE	UNKNOWN UNITS
☐	PreheatTemperature	FALSE	UNKNOWN UNITS
☐	DischargeAirTemperature	FALSE	UNKNOWN UNITS
☐	DischargeAirStaticPressure	FALSE	UNKNOWN UNITS



Remove Points

Are you sure you want to delete these points? CoolingValveOutputCommand

Each column has an *Edit Column* button in its header.



Install Devices

Platform: tcp://127.0.0.1:22916

Method: Scan for Devices

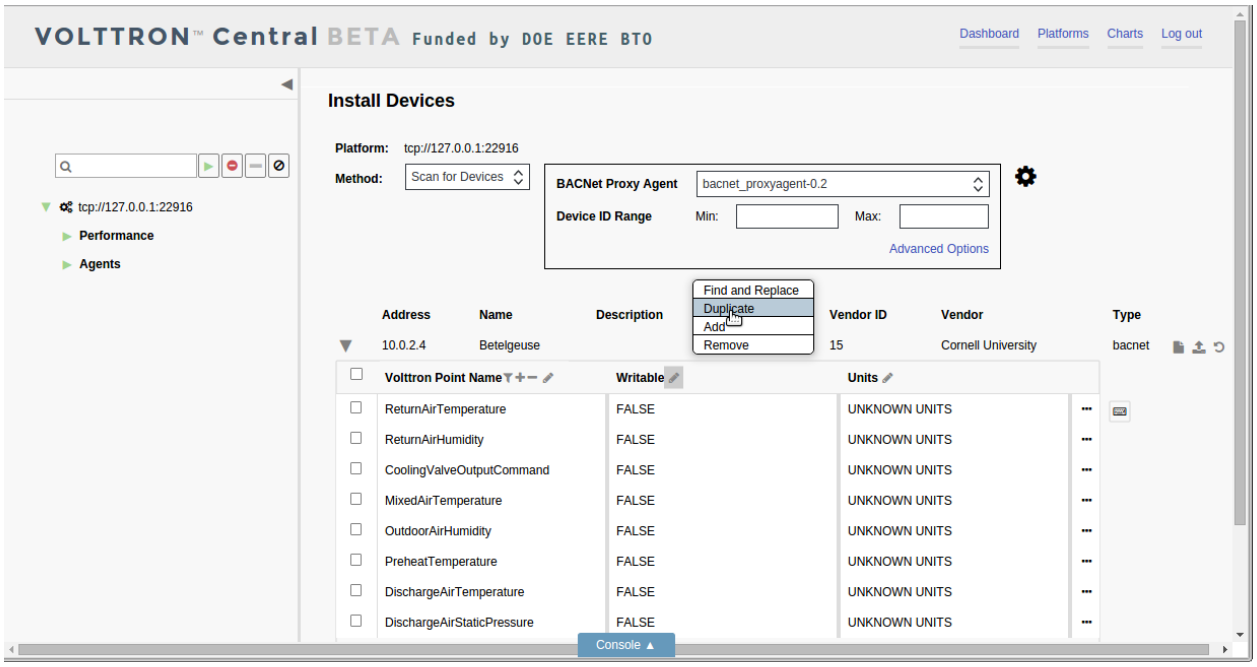
BACNet Proxy Agent: bacnet_proxyagent-0.2

Device ID Range Min: Max:

Address	Name	Description	Vendor ID	Vendor	Type
10.0.2.4	Betelgeuse		15	Cornell University	bacnet

Volttron Point Name	Writable	Units
ReturnAirTemperature	FALSE	UNKNOWN UNITS
ReturnAirHumidity	FALSE	UNKNOWN UNITS
CoolingValveOutputCommand	FALSE	UNKNOWN UNITS
MixedAirTemperature	FALSE	UNKNOWN UNITS
OutdoorAirHumidity	FALSE	UNKNOWN UNITS
PreheatTemperature	FALSE	UNKNOWN UNITS
DischargeAirTemperature	FALSE	UNKNOWN UNITS
DischargeAirStaticPressure	FALSE	UNKNOWN UNITS

Click on the button to display a popup menu of operations to perform on the column. The options include inserting a blank new column, duplicating an existing column, removing a column, or searching for a value within a column.



Install Devices

Platform: tcp://127.0.0.1:22916

Method: Scan for Devices

BACNet Proxy Agent: bacnet_proxyagent-0.2

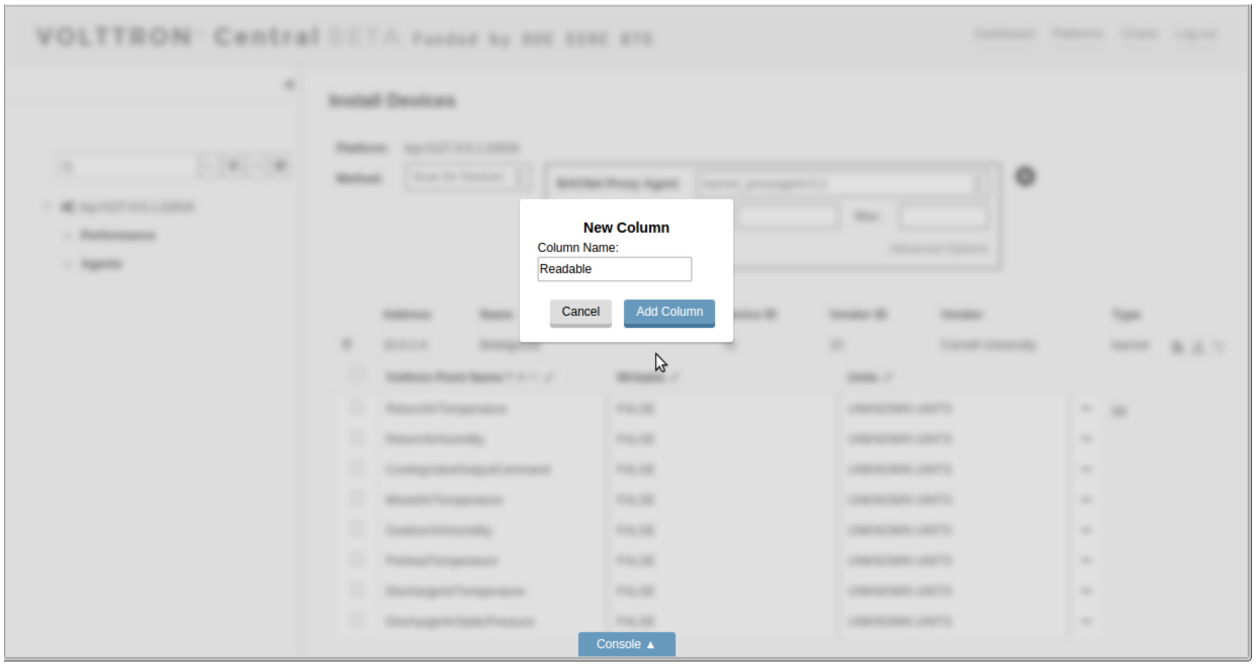
Device ID Range Min: Max:

Advanced Options

Address	Name	Description	Vendor ID	Vendor	Type
10.0.2.4	Betelgeuse		15	Cornell University	bacnet

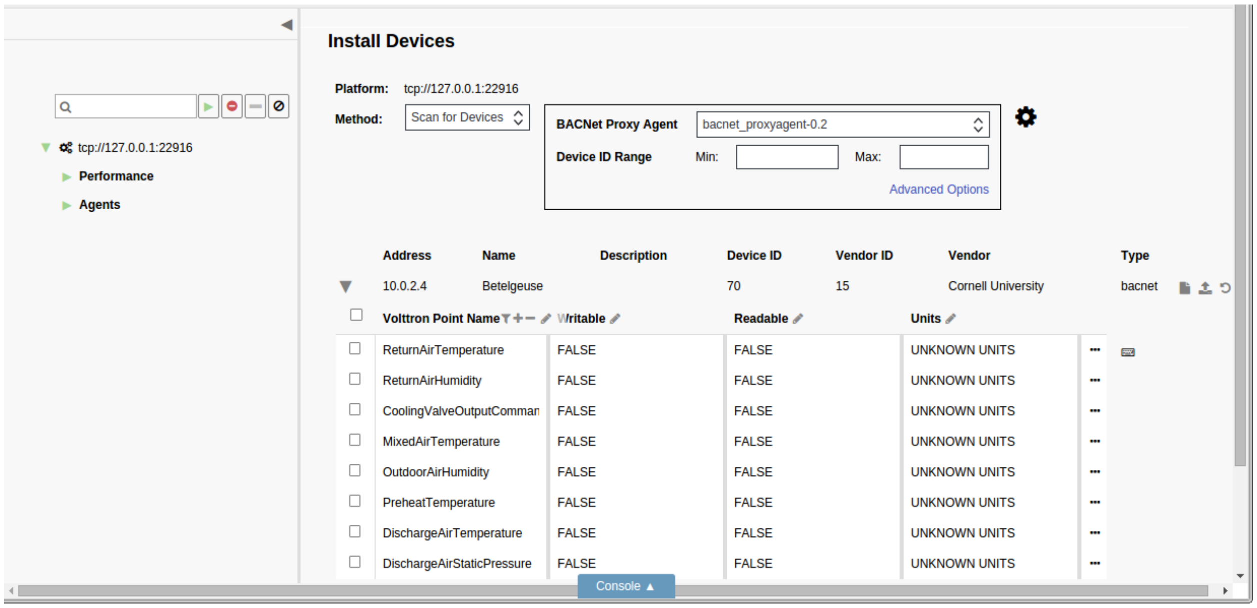
Volttron Point Name	Writable	Units
ReturnAirTemperature	FALSE	UNKNOWN UNITS
ReturnAirHumidity	FALSE	UNKNOWN UNITS
CoolingValveOutputCommand	FALSE	UNKNOWN UNITS
MixedAirTemperature	FALSE	UNKNOWN UNITS
OutdoorAirHumidity	FALSE	UNKNOWN UNITS
PreheatTemperature	FALSE	UNKNOWN UNITS
DischargeAirTemperature	FALSE	UNKNOWN UNITS
DischargeAirStaticPressure	FALSE	UNKNOWN UNITS

A duplicate or new column has to be given a unique name.



New Column

Column Name: Readable



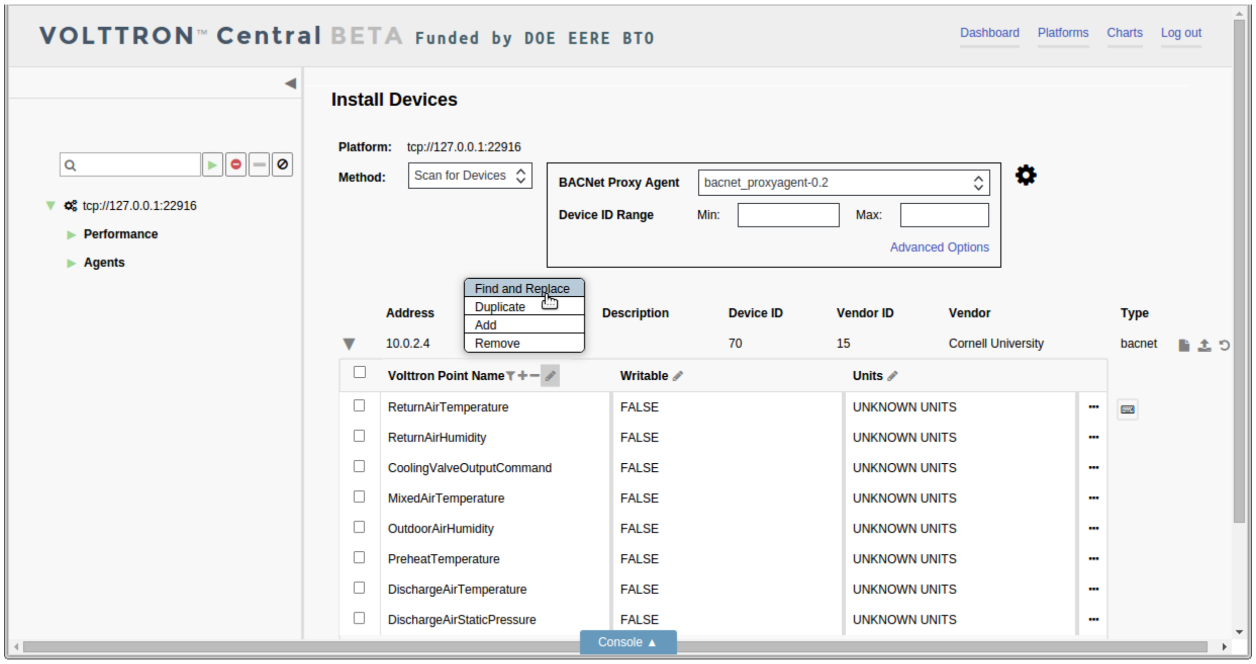
Install Devices

Platform: tcp://127.0.0.1:22916
Method: Scan for Devices
BACNet Proxy Agent: bacnet_proxyagent-0.2
Device ID Range Min: Max:

Address	Name	Description	Device ID	Vendor ID	Vendor	Type
10.0.2.4	Betelgeuse		70	15	Cornell University	bacnet

Volttron Point Name	Writable	Readable	Units
ReturnAirTemperature	FALSE	FALSE	UNKNOWN UNITS
ReturnAirHumidity	FALSE	FALSE	UNKNOWN UNITS
CoolingValveOutputComman	FALSE	FALSE	UNKNOWN UNITS
MixedAirTemperature	FALSE	FALSE	UNKNOWN UNITS
OutdoorAirHumidity	FALSE	FALSE	UNKNOWN UNITS
PreheatTemperature	FALSE	FALSE	UNKNOWN UNITS
DischargeAirTemperature	FALSE	FALSE	UNKNOWN UNITS
DischargeAirStaticPressure	FALSE	FALSE	UNKNOWN UNITS

To search for values in a column, choose the *Find and Replace* option in the popup menu.



VOLTTRON™ Central BETA Funded by DOE EERE BTO

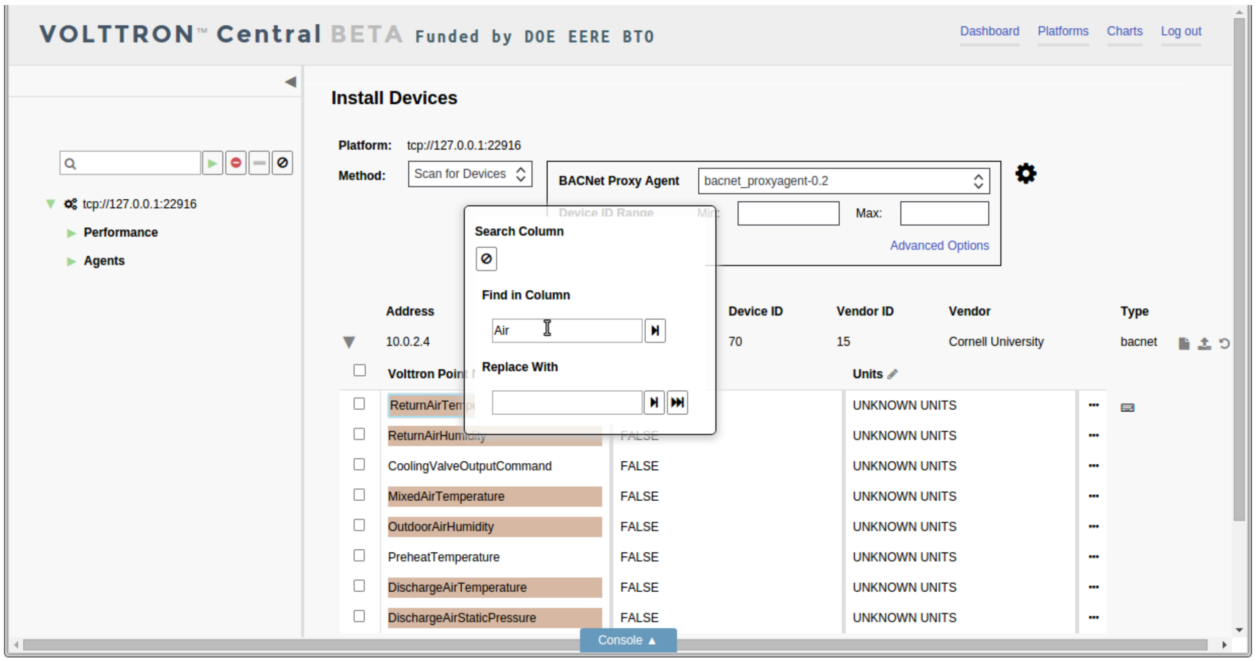
Install Devices

Platform: tcp://127.0.0.1:22916
Method: Scan for Devices
BACNet Proxy Agent: bacnet_proxyagent-0.2
Device ID Range Min: Max:

Address	Description	Device ID	Vendor ID	Vendor	Type
10.0.2.4		70	15	Cornell University	bacnet

Volttron Point Name	Writable	Units
ReturnAirTemperature	FALSE	UNKNOWN UNITS
ReturnAirHumidity	FALSE	UNKNOWN UNITS
CoolingValveOutputCommand	FALSE	UNKNOWN UNITS
MixedAirTemperature	FALSE	UNKNOWN UNITS
OutdoorAirHumidity	FALSE	UNKNOWN UNITS
PreheatTemperature	FALSE	UNKNOWN UNITS
DischargeAirTemperature	FALSE	UNKNOWN UNITS
DischargeAirStaticPressure	FALSE	UNKNOWN UNITS

Type the term to search for, and click the *Find Next* button to highlight all the matched fields in the column.



Install Devices

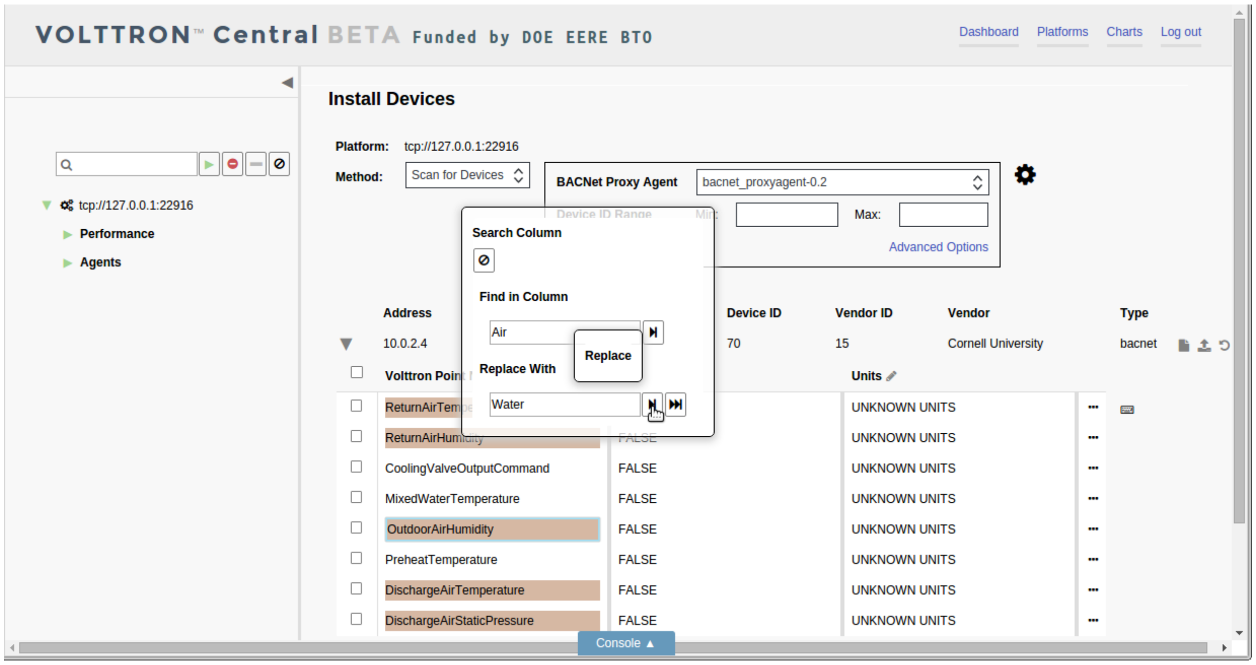
Platform: tcp://127.0.0.1:22916
Method: Scan for Devices
BACNet Proxy Agent: bacnet_proxyagent-0.2
Find in Column: Air

Address	Device ID	Vendor ID	Vendor	Type
10.0.2.4	70	15	Cornell University	bacnet

Volttron Point		Units
ReturnAirTemp		UNKNOWN UNITS
ReturnAirHumidity	FALSE	UNKNOWN UNITS
CoolingValveOutputCommand	FALSE	UNKNOWN UNITS
MixedAirTemperature	FALSE	UNKNOWN UNITS
OutdoorAirHumidity	FALSE	UNKNOWN UNITS
PreheatTemperature	FALSE	UNKNOWN UNITS
DischargeAirTemperature	FALSE	UNKNOWN UNITS
DischargeAirStaticPressure	FALSE	UNKNOWN UNITS

Click the *Find Next* button again to advance the focus down the list of matched terms.

To quickly replace the matched term in the cell with focus, type a replacement term, and click on the *Replace* button.



Install Devices

Platform: tcp://127.0.0.1:22916
Method: Scan for Devices
BACNet Proxy Agent: bacnet_proxyagent-0.2
Find in Column: Air
Replace With: Water

Address	Device ID	Vendor ID	Vendor	Type
10.0.2.4	70	15	Cornell University	bacnet

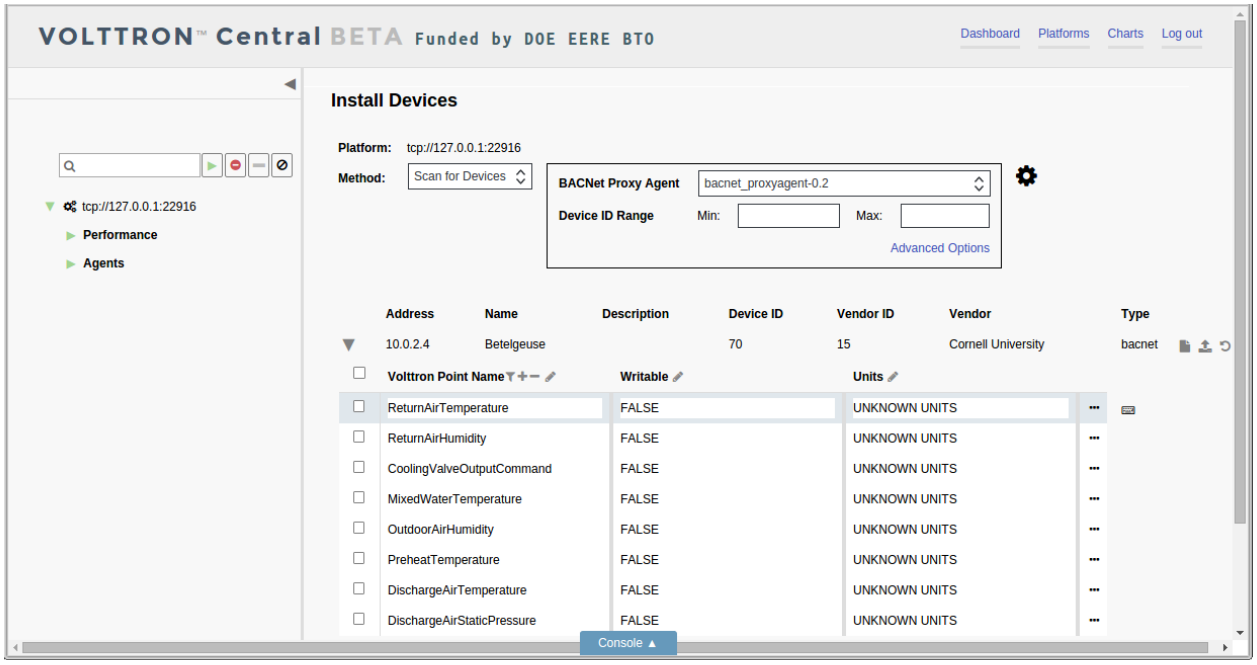
Volttron Point		Units
ReturnAirTemp		UNKNOWN UNITS
ReturnAirHumidity	FALSE	UNKNOWN UNITS
CoolingValveOutputCommand	FALSE	UNKNOWN UNITS
MixedWaterTemperature	FALSE	UNKNOWN UNITS
OutdoorAirHumidity	FALSE	UNKNOWN UNITS
PreheatTemperature	FALSE	UNKNOWN UNITS
DischargeAirTemperature	FALSE	UNKNOWN UNITS
DischargeAirStaticPressure	FALSE	UNKNOWN UNITS

To replace all the matched terms in the column, click on the *Replace All* button. Click the *Clear Search* button to end the search.

Keyboard Commands

Some keyboard commands are available to expedite the selection or de-selection of points. To initiate use of the keyboard commands, strike the *Control* key on the keyboard. For keyboard commands to be activated, the registry configuration editor has to have focus, which comes from interacting with it. But the commands won't be activated if the cursor is in a type-able field.

If the keyboard commands have been successfully activated, a faint highlight will appear over the first row in the registry configuration editor.



Install Devices

Platform: tcp://127.0.0.1:22916

Method: Scan for Devices

BACNet Proxy Agent: bacnet_proxyagent-0.2

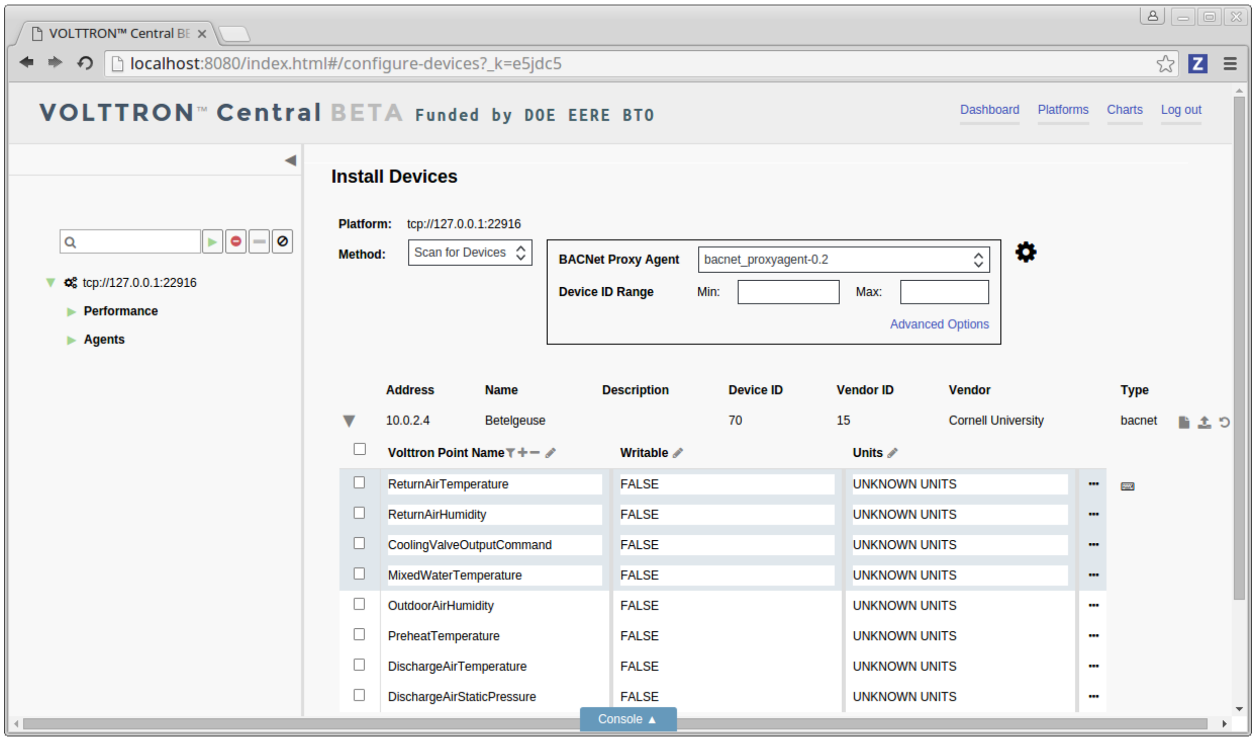
Device ID Range Min: Max:

Address	Name	Description	Device ID	Vendor ID	Vendor	Type
10.0.2.4	Betelgeuse		70	15	Cornell University	bacnet

Volttron Point Name	Writable	Units
ReturnAirTemperature	FALSE	UNKNOWN UNITS
ReturnAirHumidity	FALSE	UNKNOWN UNITS
CoolingValveOutputCommand	FALSE	UNKNOWN UNITS
MixedWaterTemperature	FALSE	UNKNOWN UNITS
OutdoorAirHumidity	FALSE	UNKNOWN UNITS
PreheatTemperature	FALSE	UNKNOWN UNITS
DischargeAirTemperature	FALSE	UNKNOWN UNITS
DischargeAirStaticPressure	FALSE	UNKNOWN UNITS

Keyboard commands are deactivated when the mouse cursor moves over the configuration editor. If unintentional deactivation occurs, strike the *Control* key again to reactivate the commands.

With keyboard commands activated, the highlighted row can be advanced up or down by striking the *up* or *down arrow* on the keyboard. A group of rows can be highlighted by striking the up or down arrow while holding down the *Shift* key.



Install Devices

Platform: tcp://127.0.0.1:22916

Method: Scan for Devices

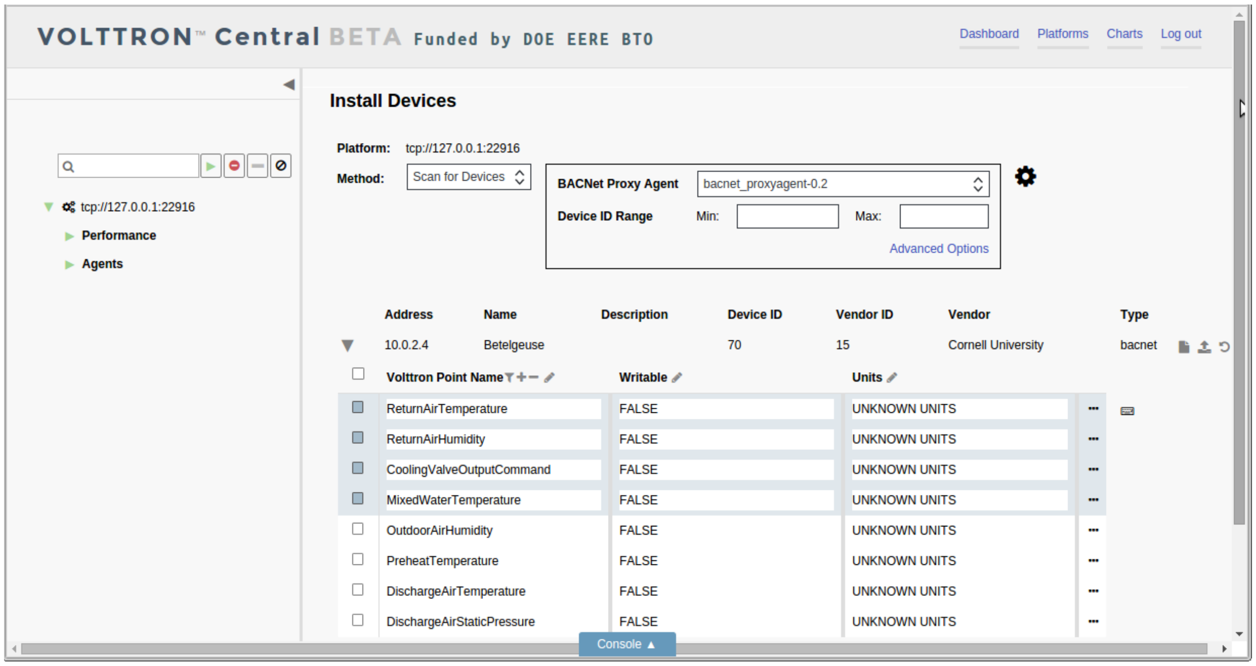
BACNet Proxy Agent: bacnet_proxyagent-0.2

Device ID Range Min: Max:

Address	Name	Description	Device ID	Vendor ID	Vendor	Type
10.0.2.4	Betelgeuse		70	15	Cornell University	bacnet

Volttron Point Name	Writable	Units
ReturnAirTemperature	FALSE	UNKNOWN UNITS
ReturnAirHumidity	FALSE	UNKNOWN UNITS
CoolingValveOutputCommand	FALSE	UNKNOWN UNITS
MixedWaterTemperature	FALSE	UNKNOWN UNITS
OutdoorAirHumidity	FALSE	UNKNOWN UNITS
PreheatTemperature	FALSE	UNKNOWN UNITS
DischargeAirTemperature	FALSE	UNKNOWN UNITS
DischargeAirStaticPressure	FALSE	UNKNOWN UNITS

To select the highlighted rows, strike the *Enter* key.



Install Devices

Platform: tcp://127.0.0.1:22916

Method: Scan for Devices

BACNet Proxy Agent: bacnet_proxyagent-0.2

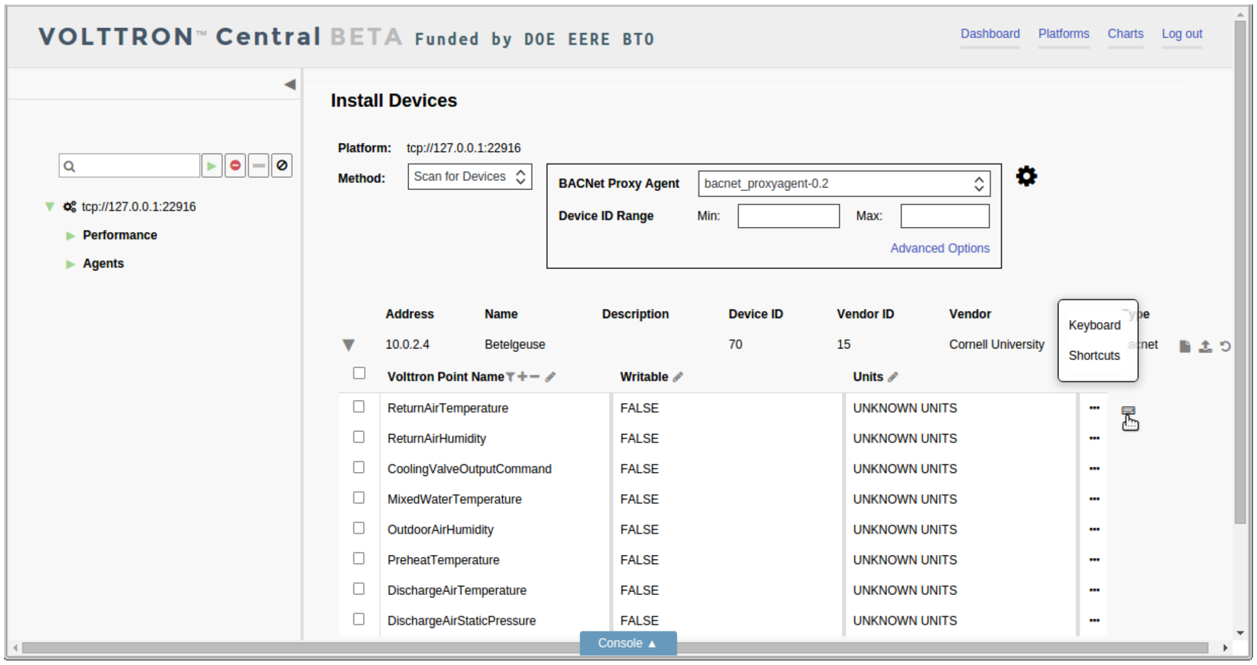
Device ID Range Min: Max:

Address	Name	Description	Device ID	Vendor ID	Vendor	Type
10.0.2.4	Betelgeuse		70	15	Cornell University	bacnet

Volttron Point Name	Writable	Units
ReturnAirTemperature	FALSE	UNKNOWN UNITS
ReturnAirHumidity	FALSE	UNKNOWN UNITS
CoolingValveOutputCommand	FALSE	UNKNOWN UNITS
MixedWaterTemperature	FALSE	UNKNOWN UNITS
OutdoorAirHumidity	FALSE	UNKNOWN UNITS
PreheatTemperature	FALSE	UNKNOWN UNITS
DischargeAirTemperature	FALSE	UNKNOWN UNITS
DischargeAirStaticPressure	FALSE	UNKNOWN UNITS

Striking the *Enter* key with rows highlighted will also deselect any rows that were already selected.

Click on the *Keyboard Shortcuts* button to show a popup list of the available keyboard commands.



Install Devices

Platform: tcp://127.0.0.1:22916

Method: Scan for Devices

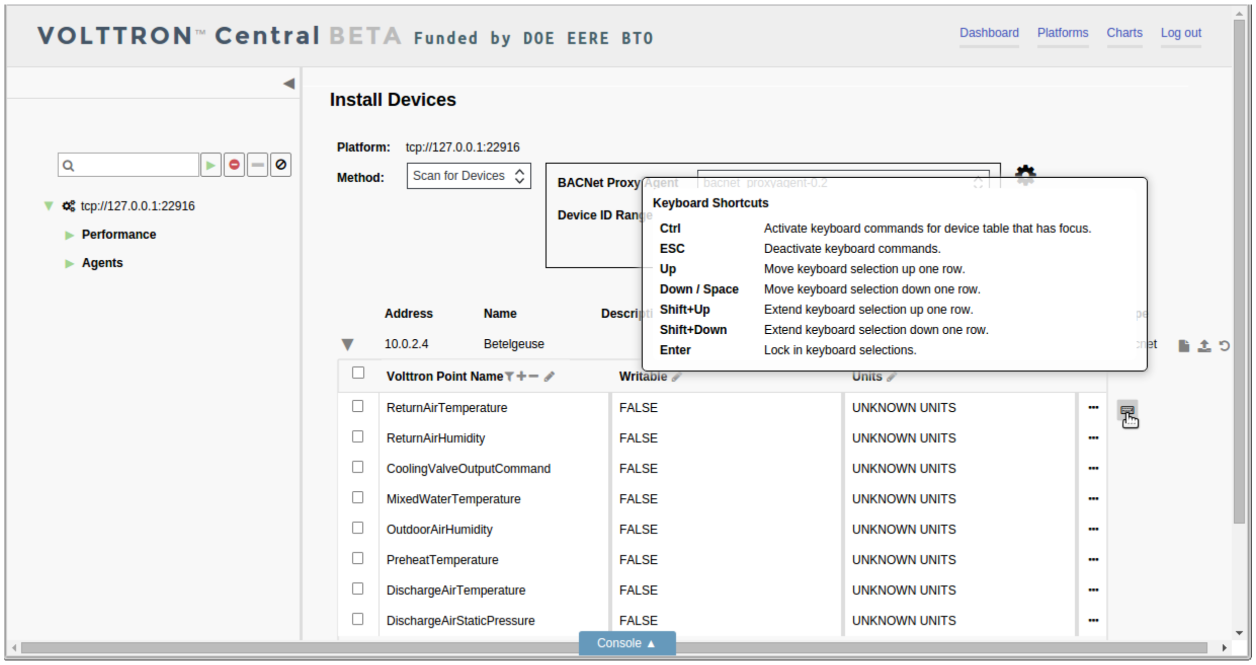
BACNet Proxy Agent: bacnet_proxyagent-0.2

Device ID Range Min: Max:

Advanced Options

Address	Name	Description	Device ID	Vendor ID	Vendor
10.0.2.4	Betelgeuse		70	15	Cornell University

Volttron Point Name	Writable	Units
ReturnAirTemperature	FALSE	UNKNOWN UNITS
ReturnAirHumidity	FALSE	UNKNOWN UNITS
CoolingValveOutputCommand	FALSE	UNKNOWN UNITS
MixedWaterTemperature	FALSE	UNKNOWN UNITS
OutdoorAirHumidity	FALSE	UNKNOWN UNITS
PreheatTemperature	FALSE	UNKNOWN UNITS
DischargeAirTemperature	FALSE	UNKNOWN UNITS
DischargeAirStaticPressure	FALSE	UNKNOWN UNITS



Install Devices

Platform: tcp://127.0.0.1:22916

Method: Scan for Devices

Keyboard Shortcuts

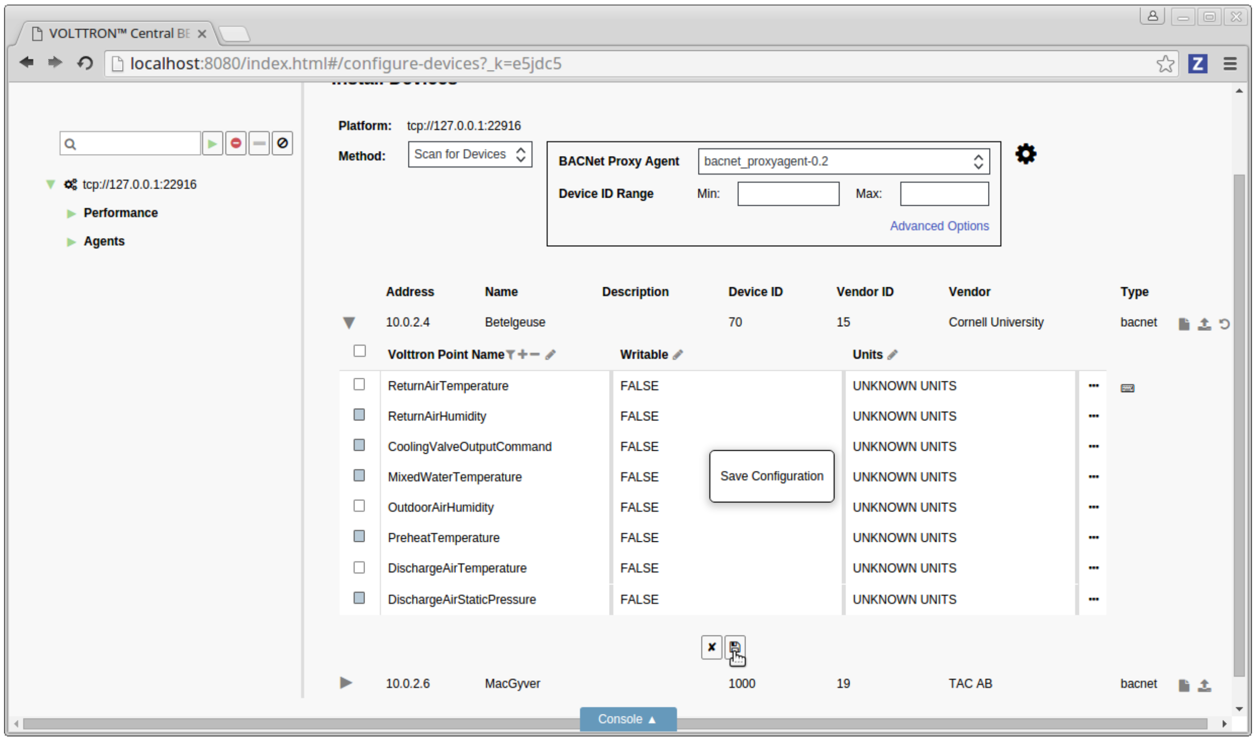
Ctrl	Activate keyboard commands for device table that has focus.
ESC	Deactivate keyboard commands.
Up	Move keyboard selection up one row.
Down / Space	Move keyboard selection down one row.
Shift+Up	Extend keyboard selection up one row.
Shift+Down	Extend keyboard selection down one row.
Enter	Lock in keyboard selections.

Address	Name
10.0.2.4	Betelgeuse

Volttron Point Name	Writable	Units
ReturnAirTemperature	FALSE	UNKNOWN UNITS
ReturnAirHumidity	FALSE	UNKNOWN UNITS
CoolingValveOutputCommand	FALSE	UNKNOWN UNITS
MixedWaterTemperature	FALSE	UNKNOWN UNITS
OutdoorAirHumidity	FALSE	UNKNOWN UNITS
PreheatTemperature	FALSE	UNKNOWN UNITS
DischargeAirTemperature	FALSE	UNKNOWN UNITS
DischargeAirStaticPressure	FALSE	UNKNOWN UNITS

Registry Preview

To save the registry configuration, click the *Save* button at the bottom of the registry configuration editor.



VOLTTRON™ Central B

localhost:8080/index.html#/configure-devices?_k=e5jdc5

tcp://127.0.0.1:22916

Performance

Agents

Platform: tcp://127.0.0.1:22916

Method: Scan for Devices

BACNet Proxy Agent: bacnet_proxyagent-0.2

Device ID Range Min: Max:

Advanced Options

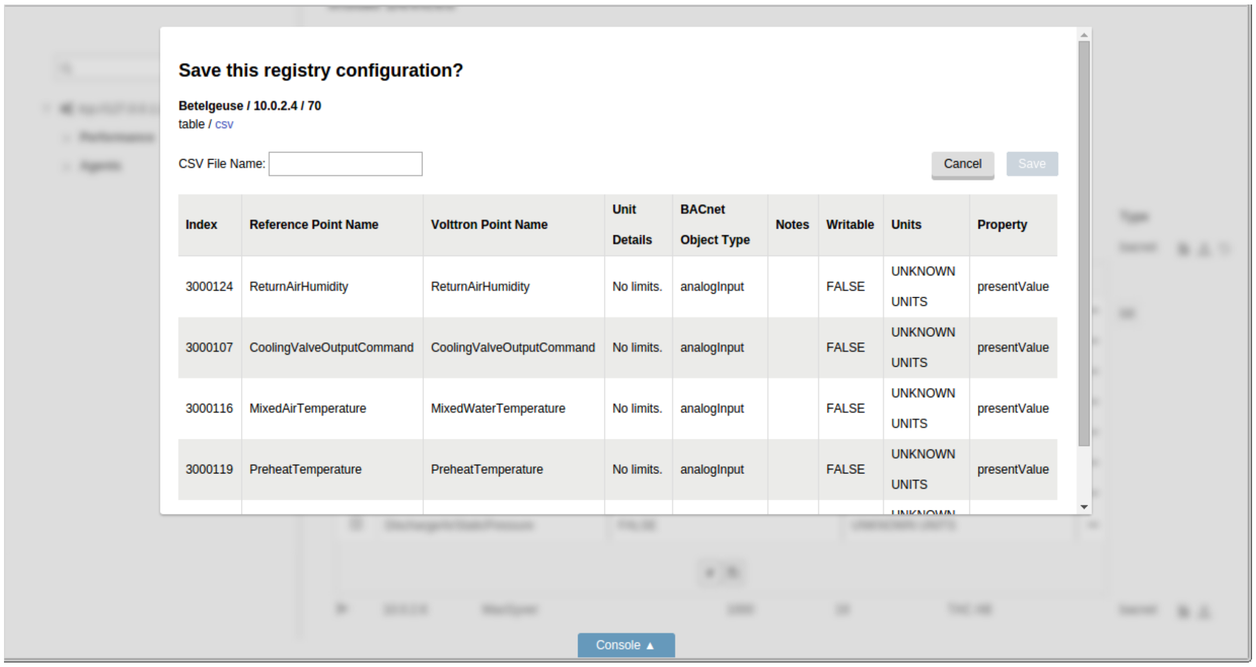
	Address	Name	Description	Device ID	Vendor ID	Vendor	Type
▼	10.0.2.4	Betelgeuse		70	15	Cornell University	bacnet
▶	10.0.2.6	MacGyver		1000	19	TAC AB	bacnet

	Volttron Point Name	Writable	Units
☐	ReturnAirTemperature	FALSE	UNKNOWN UNITS
▨	ReturnAirHumidity	FALSE	UNKNOWN UNITS
▨	CoolingValveOutputCommand	FALSE	UNKNOWN UNITS
▨	MixedWaterTemperature	FALSE	UNKNOWN UNITS
☐	OutdoorAirHumidity	FALSE	UNKNOWN UNITS
▨	PreheatTemperature	FALSE	UNKNOWN UNITS
☐	DischargeAirTemperature	FALSE	UNKNOWN UNITS
▨	DischargeAirStaticPressure	FALSE	UNKNOWN UNITS

Save Configuration

Console ▲

A preview will appear to let you confirm that the configuration is what you intended.



Save this registry configuration?

Betelgeuse / 10.0.2.4 / 70

table / csv

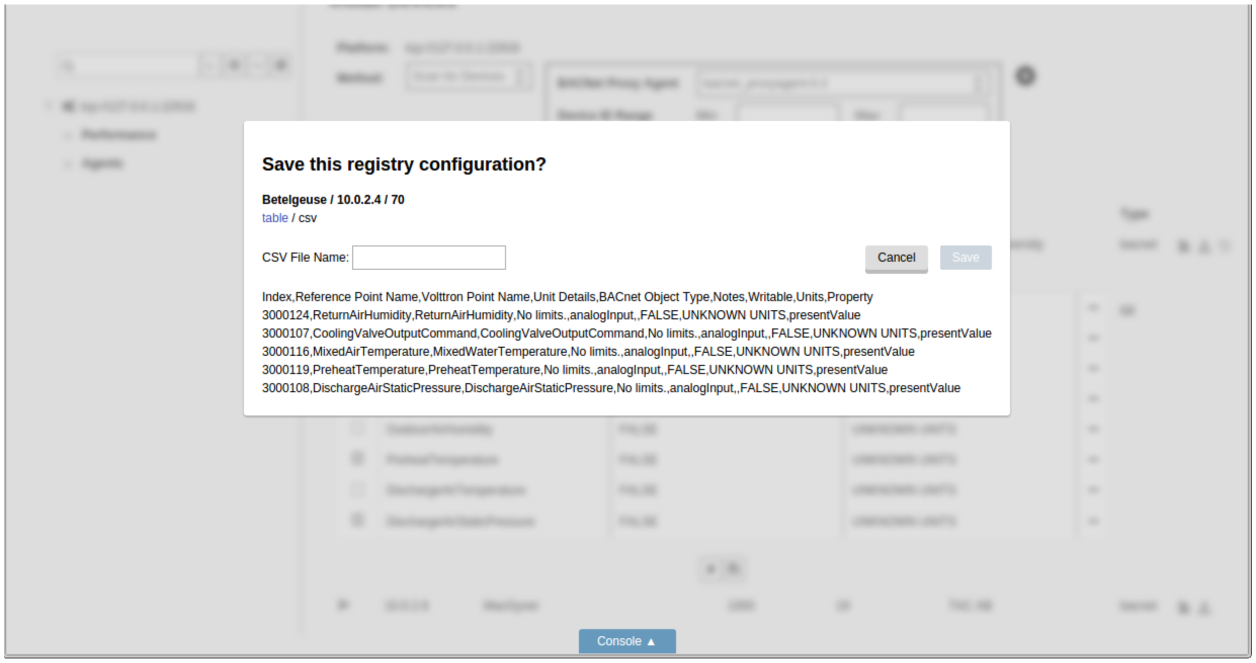
CSV File Name:

Cancel Save

Index	Reference Point Name	Volttron Point Name	Unit Details	BACnet Object Type	Notes	Writable	Units	Property
3000124	ReturnAirHumidity	ReturnAirHumidity	No limits.	analogInput		FALSE	UNKNOWN UNITS	presentValue
3000107	CoolingValveOutputCommand	CoolingValveOutputCommand	No limits.	analogInput		FALSE	UNKNOWN UNITS	presentValue
3000116	MixedAirTemperature	MixedWaterTemperature	No limits.	analogInput		FALSE	UNKNOWN UNITS	presentValue
3000119	PreheatTemperature	PreheatTemperature	No limits.	analogInput		FALSE	UNKNOWN UNITS	presentValue

Console ▲

The configuration also can be inspected in the comma-separated format of the actual registry configuration file.



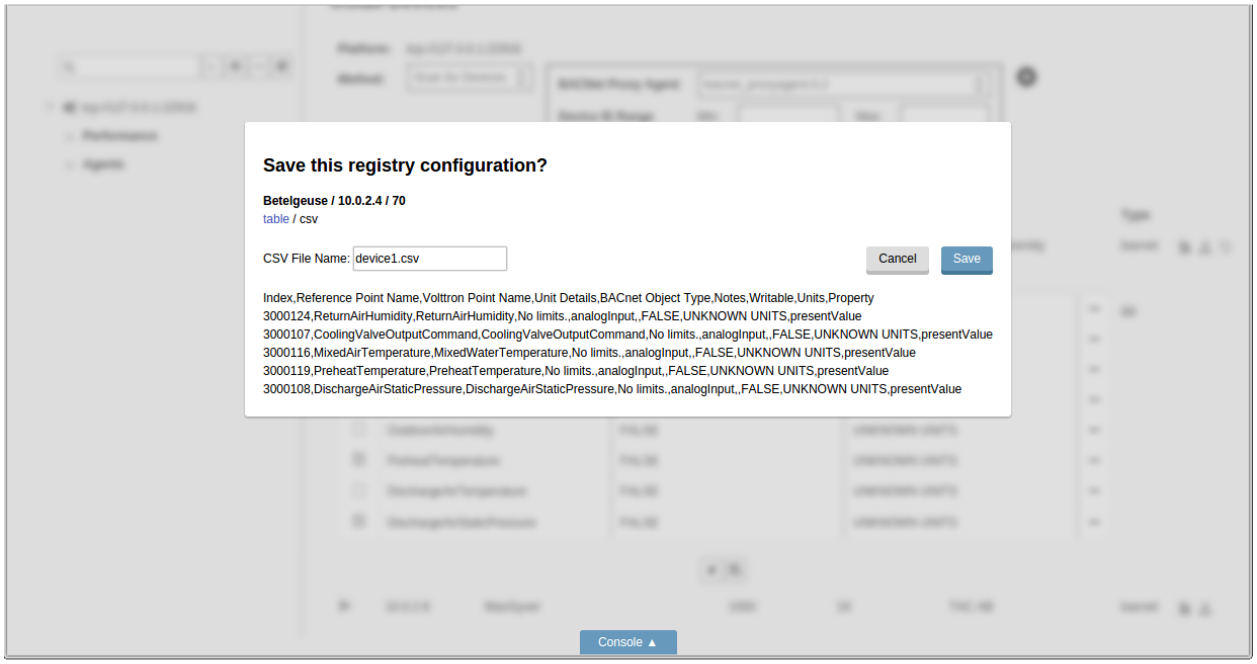
Save this registry configuration?

Betelgeuse / 10.0.2.4 / 70
table / csv

CSV File Name:

```
Index,Reference Point Name,Volttron Point Name,Unit Details,BACnet Object Type,Notes,Writable,Units,Property
3000124,ReturnAirHumidity,ReturnAirHumidity,No limits.,analogInput,,FALSE,UNKNOWN UNITS,presentValue
3000107,CoolingValveOutputCommand,CoolingValveOutputCommand,No limits.,analogInput,,FALSE,UNKNOWN UNITS,presentValue
3000116,MixedAirTemperature,MixedWaterTemperature,No limits.,analogInput,,FALSE,UNKNOWN UNITS,presentValue
3000119,PreheatTemperature,PreheatTemperature,No limits.,analogInput,,FALSE,UNKNOWN UNITS,presentValue
3000108,DischargeAirStaticPressure,DischargeAirStaticPressure,No limits.,analogInput,,FALSE,UNKNOWN UNITS,presentValue
```

Provide a name for the registry configuration file, and click the *Save* button to save the file.

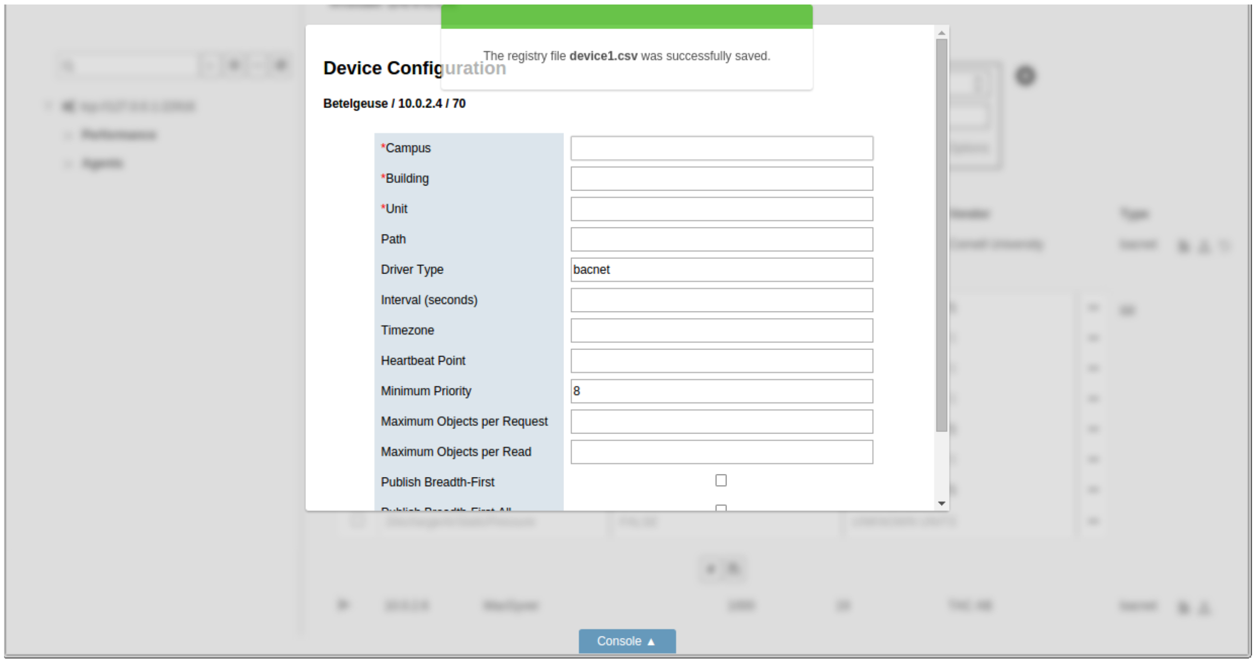


Save this registry configuration?

Betelgeuse / 10.0.2.4 / 70
table / csv

CSV File Name: device1.csv

```
Index,Reference Point Name,Volttron Point Name,Unit Details,BACnet Object Type,Notes,Writable,Units,Property
3000124,ReturnAirHumidity,ReturnAirHumidity,No limits.,analogInput,,FALSE,UNKNOWN UNITS,presentValue
3000107,CoolingValveOutputCommand,CoolingValveOutputCommand,No limits.,analogInput,,FALSE,UNKNOWN UNITS,presentValue
3000116,MixedAirTemperature,MixedWaterTemperature,No limits.,analogInput,,FALSE,UNKNOWN UNITS,presentValue
3000119,PreheatTemperature,PreheatTemperature,No limits.,analogInput,,FALSE,UNKNOWN UNITS,presentValue
3000108,DischargeAirStaticPressure,DischargeAirStaticPressure,No limits.,analogInput,,FALSE,UNKNOWN UNITS,presentValue
```



The registry file **device1.csv** was successfully saved.

Device Configuration

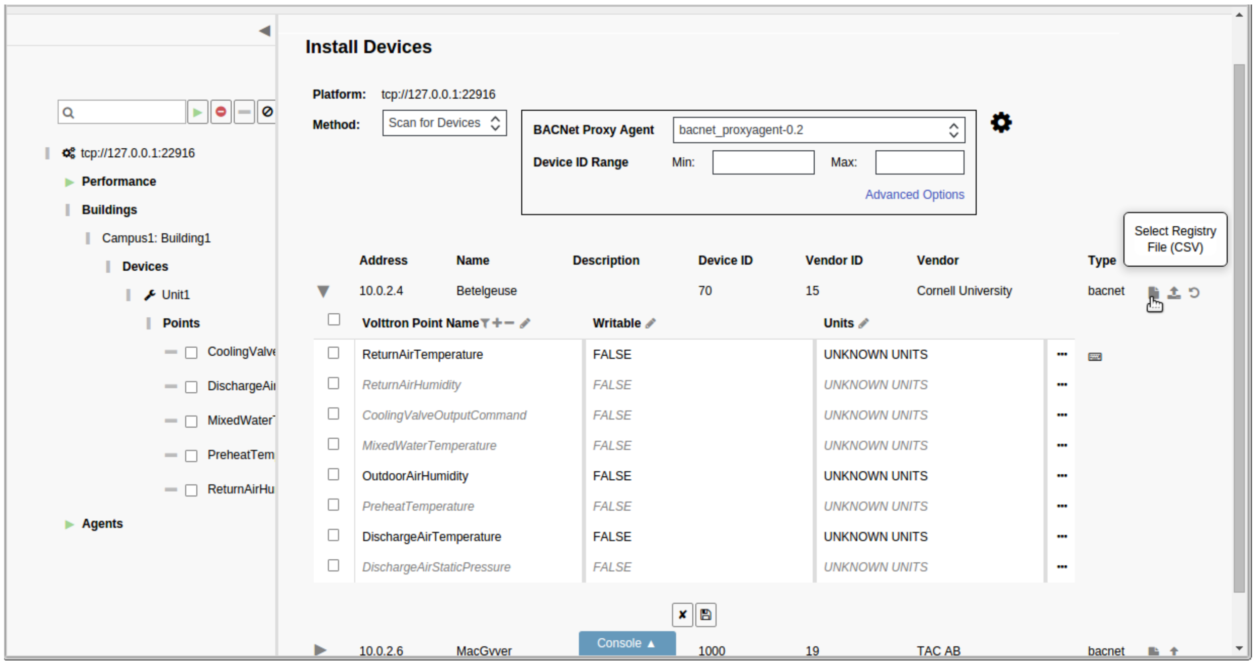
Betelgeuse / 10.0.2.4 / 70

*Campus	
*Building	
*Unit	
Path	
Driver Type	bacnet
Interval (seconds)	
Timezone	
Heartbeat Point	
Minimum Priority	8
Maximum Objects per Request	
Maximum Objects per Read	
Publish Breadth-First	☐

Registry Configuration Options

Different subsets of configured points can be saved from the same physical device and used to create separate registry files for multiple virtual devices and sub-devices. Likewise, a single registry file can be reused by multiple virtual devices and sub-devices.

To reuse a previously saved registry file, click on the *Select Registry File (CSV)* button at the end of the physical device's listing.



Install Devices

Platform: tcp://127.0.0.1:22916

Method: Scan for Devices

BACNet Proxy Agent bacnet_proxyagent-0.2

Device ID Range Min: Max:

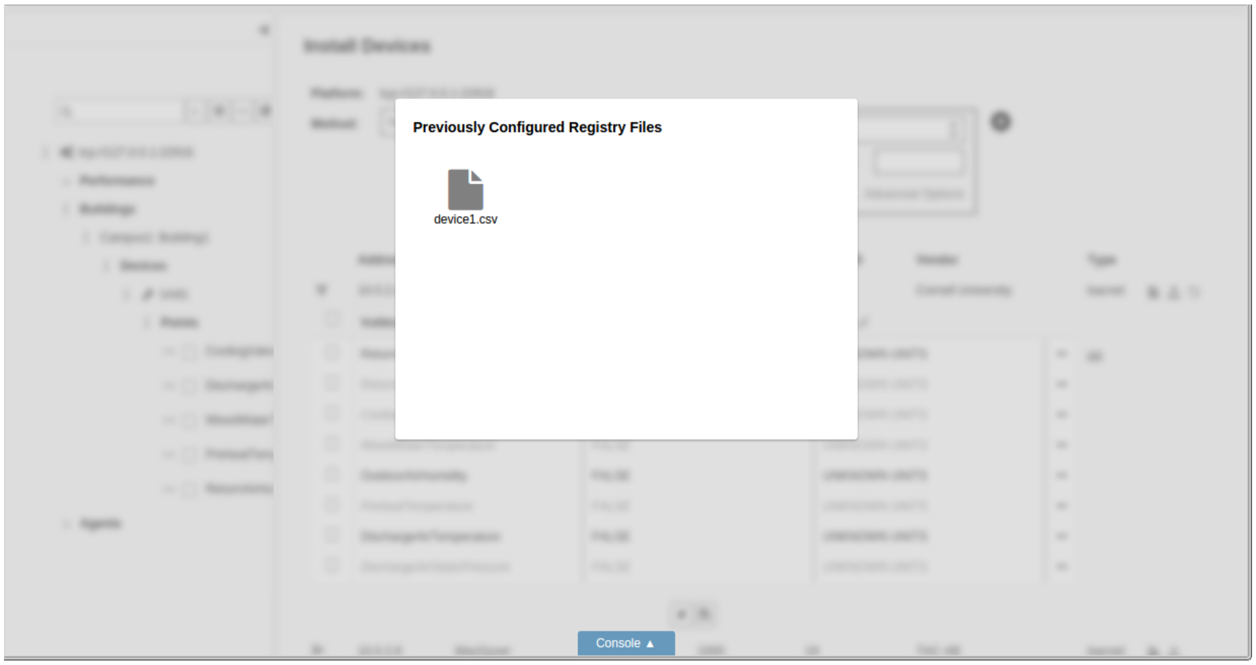
Advanced Options

Select Registry File (CSV)

Address	Name	Description	Device ID	Vendor ID	Vendor	Type
10.0.2.4	Betelgeuse		70	15	Cornell University	bacnet
10.0.2.6	MacGyver		1000	19	TAC AB	bacnet

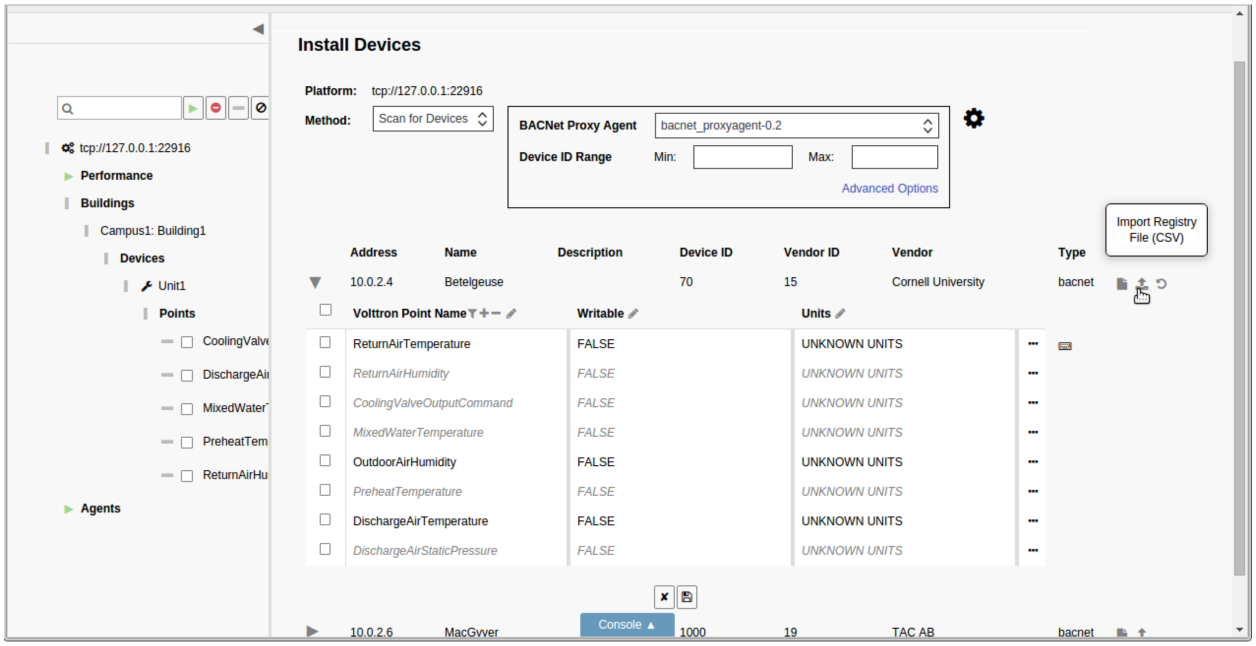
Volttron Point Name	Writable	Units
ReturnAirTemperature	FALSE	UNKNOWN UNITS
ReturnAirHumidity	*FALSE*	*UNKNOWN UNITS*
CoolingValveOutputCommand	*FALSE*	*UNKNOWN UNITS*
MixedWaterTemperature	*FALSE*	*UNKNOWN UNITS*
OutdoorAirHumidity	FALSE	UNKNOWN UNITS
PreheatTemperature	*FALSE*	*UNKNOWN UNITS*
DischargeAirTemperature	FALSE	UNKNOWN UNITS
DischargeAirStaticPressure	*FALSE*	*UNKNOWN UNITS*

The *Previously Configured Registry Files* window will appear, and a file can be selected to load it into the registry configuration editor.


Previously Configured Registry Files

device1.csv

Another option is to import a registry configuration file from the computer running the VOLTTRON Central web application, if one has been saved to local storage connected to the computer. To import a registry configuration file from local storage, click on the *Import Registry File (CSV)* button at the end of the physical device's listing, and use the file selector window to locate and load the file.


Install Devices

Platform: tcp://127.0.0.1:22916

Method: Scan for Devices

BACNet Proxy Agent: bacnet_proxyagent-0.2

Device ID Range Min: Max:

Advanced Options

Import Registry File (CSV)

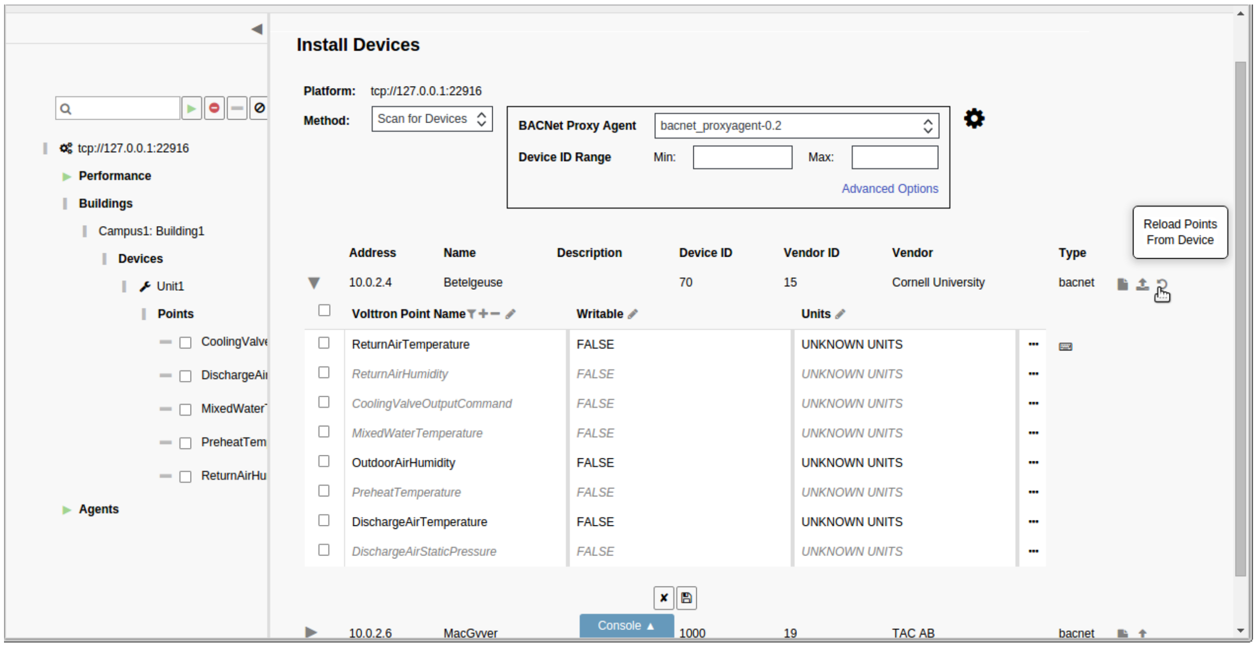
Address	Name	Description	Device ID	Vendor ID	Vendor	Type
10.0.2.4	Betelgeuse		70	15	Cornell University	bacnet

Volttron Point Name	Writable	Units
ReturnAirTemperature	FALSE	UNKNOWN UNITS
ReturnAirHumidity	FALSE	UNKNOWN UNITS
CoolingValveOutputCommand	FALSE	UNKNOWN UNITS
MixedWaterTemperature	FALSE	UNKNOWN UNITS
OutdoorAirHumidity	FALSE	UNKNOWN UNITS
PreheatTemperature	FALSE	UNKNOWN UNITS
DischargeAirTemperature	FALSE	UNKNOWN UNITS
DischargeAirStaticPressure	FALSE	UNKNOWN UNITS

Address	Name	Description	Device ID	Vendor ID	Vendor	Type
10.0.2.6	MacGyver		1000	19	TAC AB	bacnet

Reloading Device Points

Once a physical device has been scanned, the original points from the scan can be reloaded at any point during device configuration by clicking on the *Reload Points From Device* button at the end of the device's listing.


Install Devices

Platform: tcp://127.0.0.1:22916

Method: Scan for Devices

BACNet Proxy Agent: bacnet_proxyagent-0.2

Device ID Range Min: Max:

Advanced Options

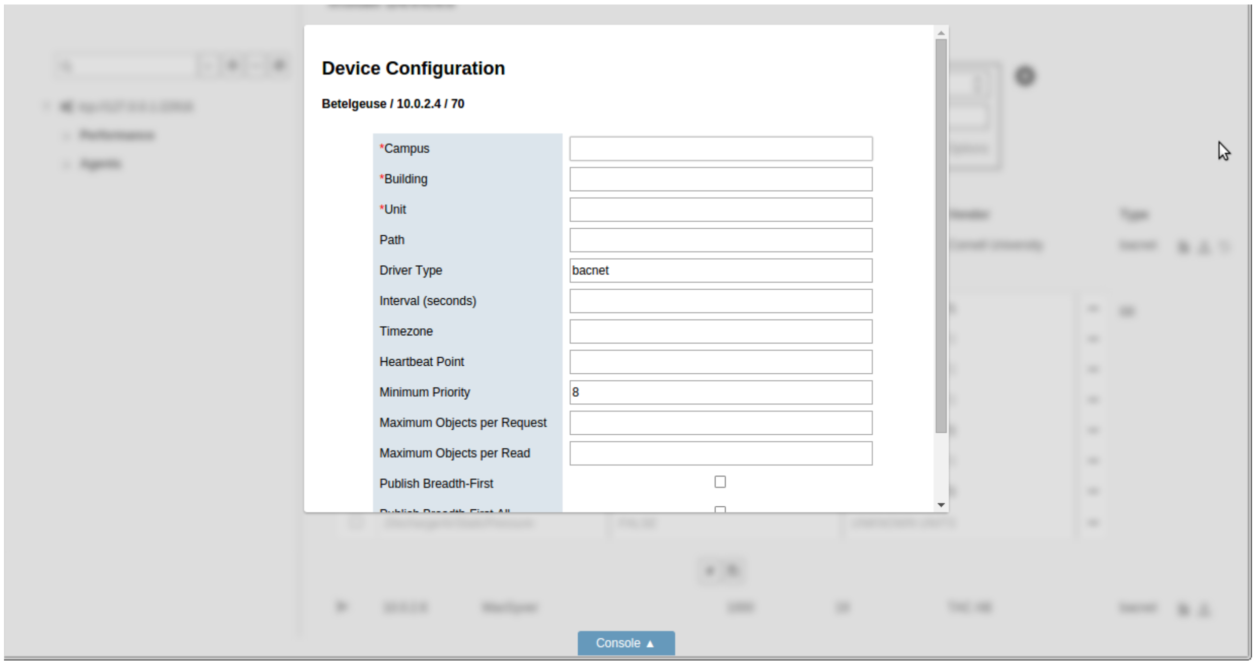
Reload Points From Device

Address	Name	Description	Device ID	Vendor ID	Vendor	Type
10.0.2.4	Betelgeuse		70	15	Cornell University	bacnet
10.0.2.6	MacGyver		1000	19	TAC AB	bacnet

Volttron Point Name	Writable	Units
ReturnAirTemperature	FALSE	UNKNOWN UNITS
ReturnAirHumidity	FALSE	UNKNOWN UNITS
CoolingValveOutputCommand	FALSE	UNKNOWN UNITS
MixedWaterTemperature	FALSE	UNKNOWN UNITS
OutdoorAirHumidity	FALSE	UNKNOWN UNITS
PreheatTemperature	FALSE	UNKNOWN UNITS
DischargeAirTemperature	FALSE	UNKNOWN UNITS
DischargeAirStaticPressure	FALSE	UNKNOWN UNITS

Device Configuration Form

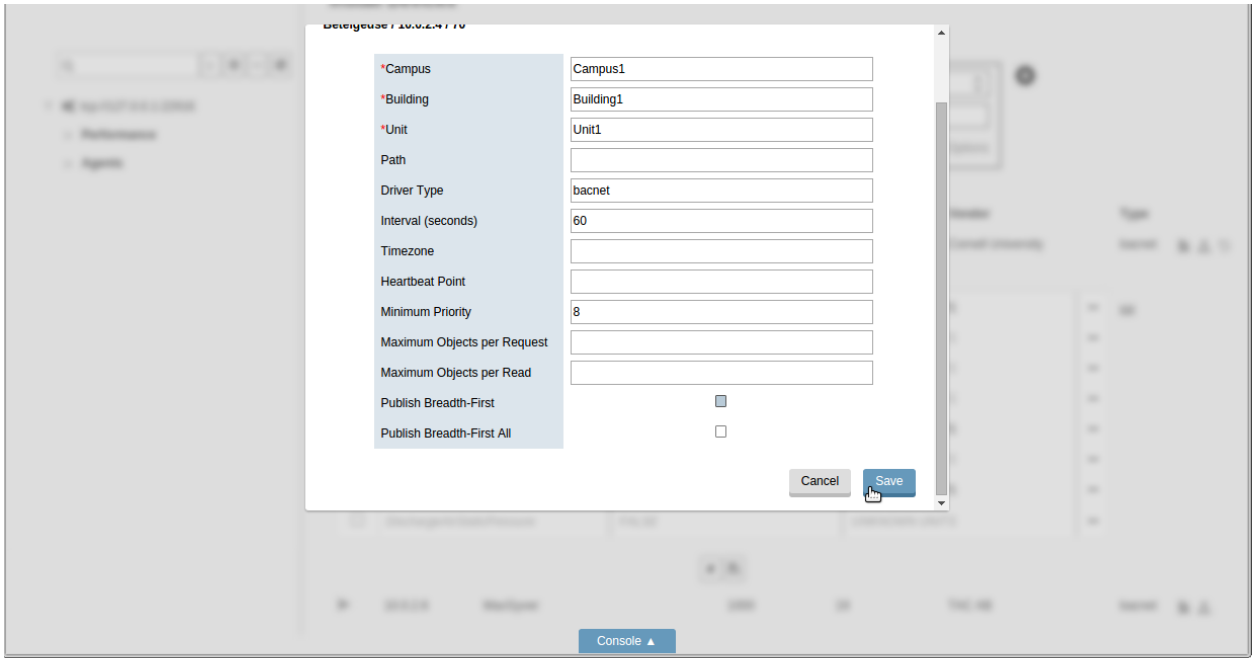
After the registry configuration file has been saved, the device configuration form appears. Creating the device configuration results in the virtual device being installed in the platform and determines the device's position in the side panel tree. It also contains some settings that determine how data is collected from the device.


Device Configuration

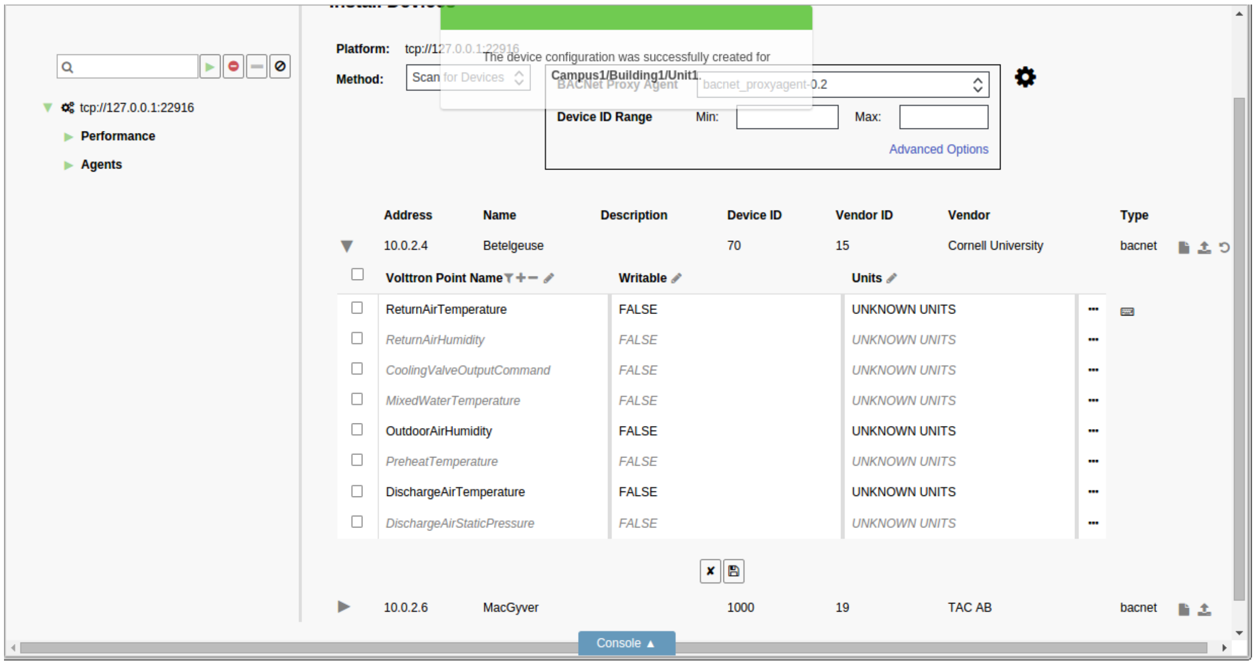
Betelgeuse / 10.0.2.4 / 70

*Campus	
*Building	
*Unit	
Path	
Driver Type	bacnet
Interval (seconds)	
Timezone	
Heartbeat Point	
Minimum Priority	8
Maximum Objects per Request	
Maximum Objects per Read	
Publish Breadth-First	☐

After the device configuration settings have been entered, click the *Save* button to save the configuration and add the device to the platform.



*Campus	Campus1
*Building	Building1
*Unit	Unit1
Path	
Driver Type	bacnet
Interval (seconds)	60
Timezone	
Heartbeat Point	
Minimum Priority	8
Maximum Objects per Request	
Maximum Objects per Read	
Publish Breadth-First	☐
Publish Breadth-First All	☐



The device configuration was successfully created for Campus1/Building1/Unit1.

Platform: tcp://127.0.0.1:22916

Method: Scan for Devices

BACNet Proxy Agent: bacnet_proxyagent-0.2

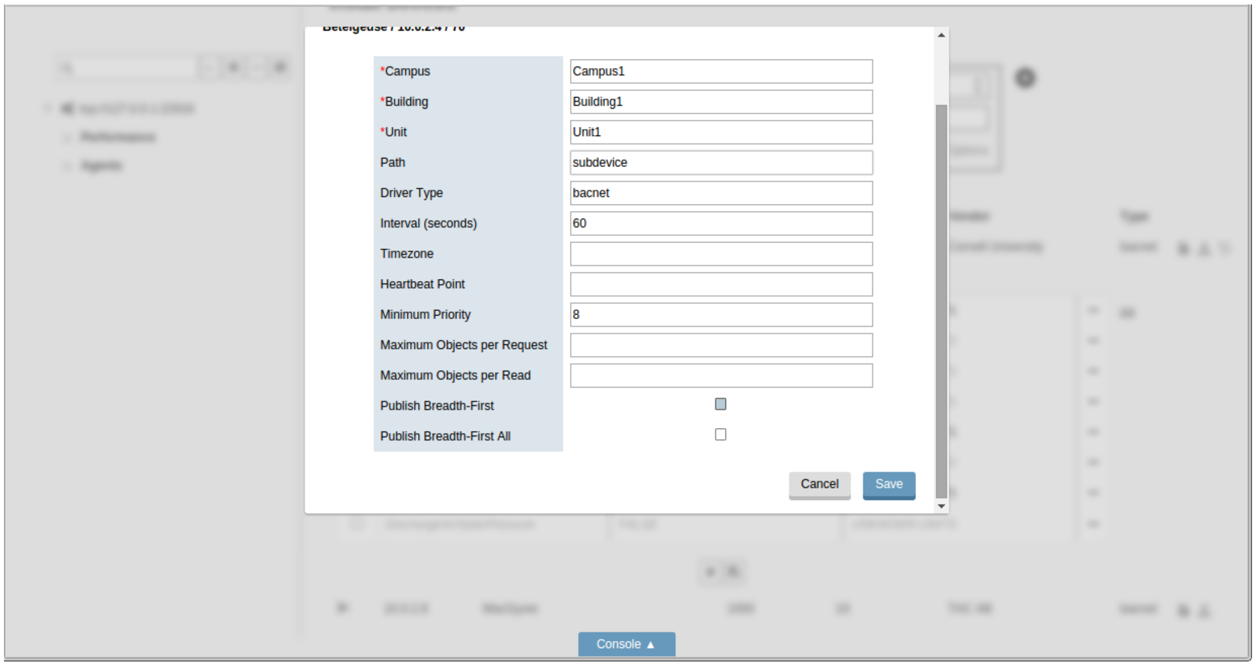
Device ID Range Min: Max:

Address	Name	Description	Device ID	Vendor ID	Vendor	Type
10.0.2.4	Betelgeuse		70	15	Cornell University	bacnet
10.0.2.6	MacGyver		1000	19	TAC AB	bacnet

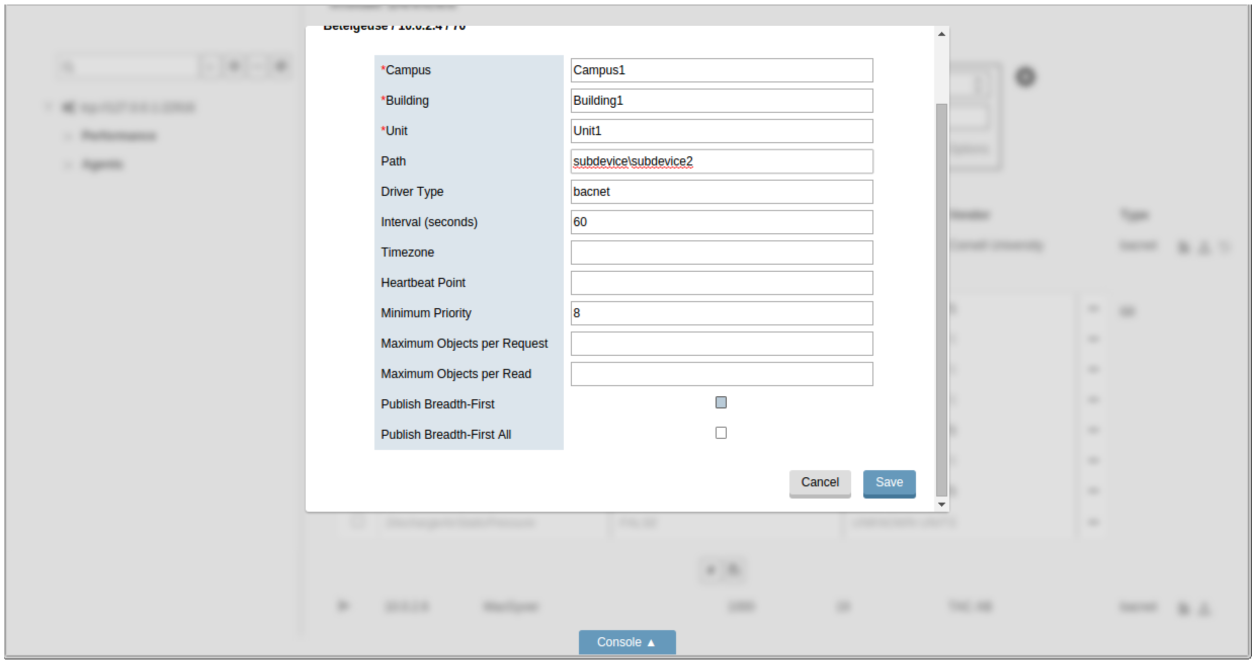
Volttron Point Name	Writable	Units
ReturnAirTemperature	FALSE	UNKNOWN UNITS
ReturnAirHumidity	FALSE	UNKNOWN UNITS
CoolingValveOutputCommand	FALSE	UNKNOWN UNITS
MixedWaterTemperature	FALSE	UNKNOWN UNITS
OutdoorAirHumidity	FALSE	UNKNOWN UNITS
PreheatTemperature	FALSE	UNKNOWN UNITS
DischargeAirTemperature	FALSE	UNKNOWN UNITS
DischargeAirStaticPressure	FALSE	UNKNOWN UNITS

Configuring Sub-devices

After a device has been configured, sub-devices can be configured by pointing to their position in the `Path` attribute of the device configuration form. But a sub-device can't be configured until its parent device has been configured first.

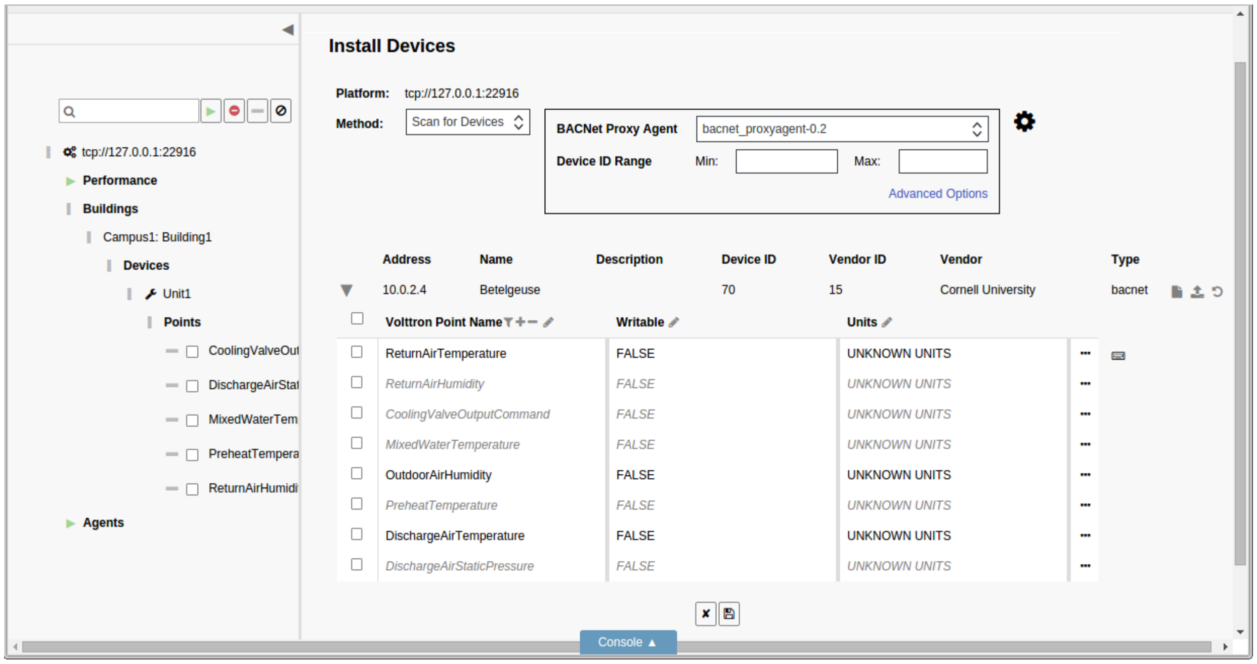


*Campus	Campus1
*Building	Building1
*Unit	Unit1
Path	subdevice
Driver Type	bacnet
Interval (seconds)	60
Timezone	
Heartbeat Point	
Minimum Priority	8
Maximum Objects per Request	
Maximum Objects per Read	
Publish Breadth-First	☐
Publish Breadth-First All	☐



*Campus	Campus1
*Building	Building1
*Unit	Unit1
Path	subdevice\subdevice2
Driver Type	bacnet
Interval (seconds)	60
Timezone	
Heartbeat Point	
Minimum Priority	8
Maximum Objects per Request	
Maximum Objects per Read	
Publish Breadth-First	☐
Publish Breadth-First All	☐

As devices are configured, they’re inserted into position in the side panel tree, along with their configured points.

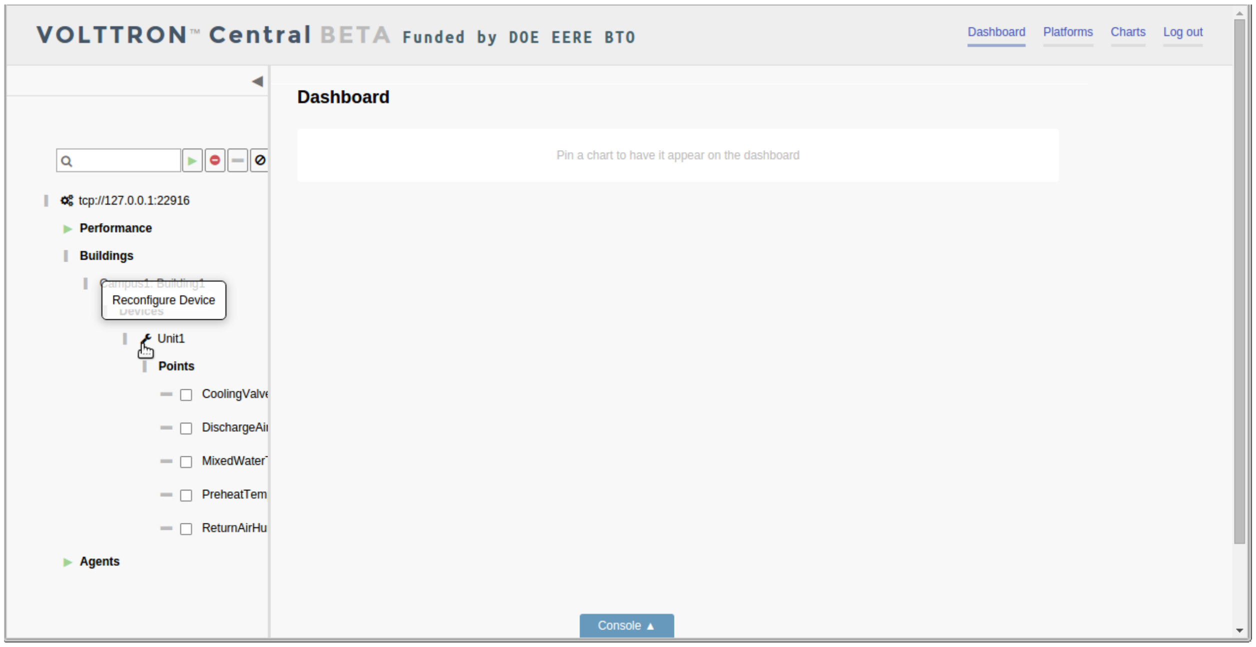


Address	Name	Description	Device ID	Vendor ID	Vendor	Type
10.0.2.4	Betelgeuse		70	15	Cornell University	bacnet

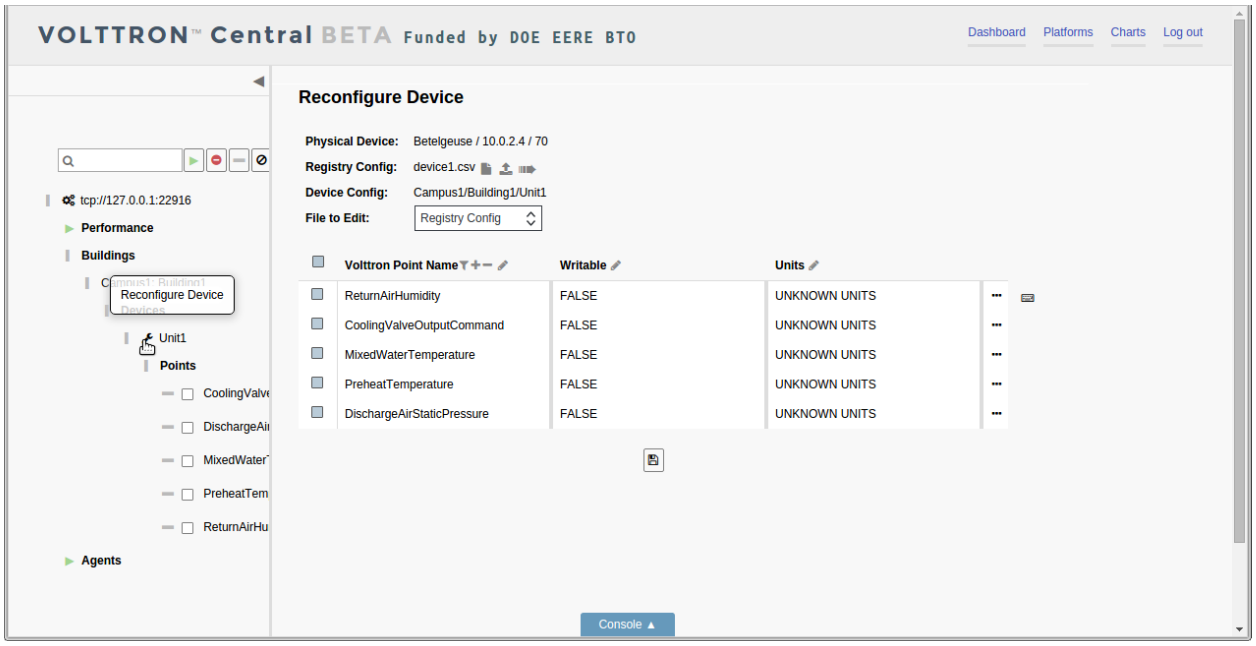
Volttron Point Name	Writable	Units
ReturnAirTemperature	FALSE	UNKNOWN UNITS
ReturnAirHumidity	FALSE	UNKNOWN UNITS
CoolingValveOutputCommand	FALSE	UNKNOWN UNITS
MixedWaterTemperature	FALSE	UNKNOWN UNITS
OutdoorAirHumidity	FALSE	UNKNOWN UNITS
PreheatTemperature	FALSE	UNKNOWN UNITS
DischargeAirTemperature	FALSE	UNKNOWN UNITS
DischargeAirStaticPressure	FALSE	UNKNOWN UNITS

Reconfiguring Devices

A device that's been added to a VOLTTRON instance can be reconfigured by changing its registry configuration or its device configuration. To launch reconfiguration, click on the wrench button next to the device in the side panel tree.



Reconfiguration reloads the registry configuration editor and the device configuration form for the virtual device. The editor and the form work the same way in reconfiguration as during initial device configuration.



Reconfigure Device

Physical Device: Betelgeuse / 10.0.2.4 / 70
Registry Config: device1.csv
Device Config: Campus1/Building1/Unit1
File to Edit: Registry Config

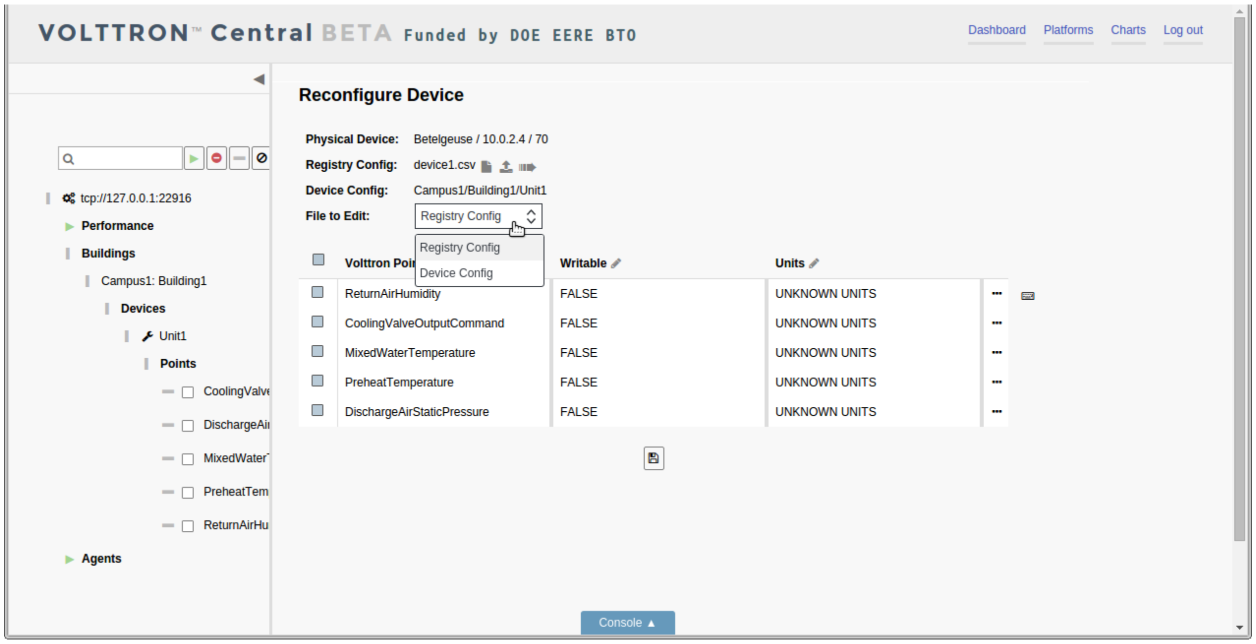
Volttron Point Name	Writable	Units
ReturnAirHumidity	FALSE	UNKNOWN UNITS
CoolingValveOutputCommand	FALSE	UNKNOWN UNITS
MixedWaterTemperature	FALSE	UNKNOWN UNITS
PreheatTemperature	FALSE	UNKNOWN UNITS
DischargeAirStaticPressure	FALSE	UNKNOWN UNITS

The reconfiguration view shows the name, address, and ID of the physical device that the virtual device was configured from. It also shows the name of the registry configuration file associated with the virtual device as well as its configured path.

A different registry configuration file can be associated with the device by clicking on the *Select Registry File (CSV)* button or the *Import Registry File (CSV)* button.

The registry configuration can be edited by making changes to the configuration in the editor and clicking the *Save* button.

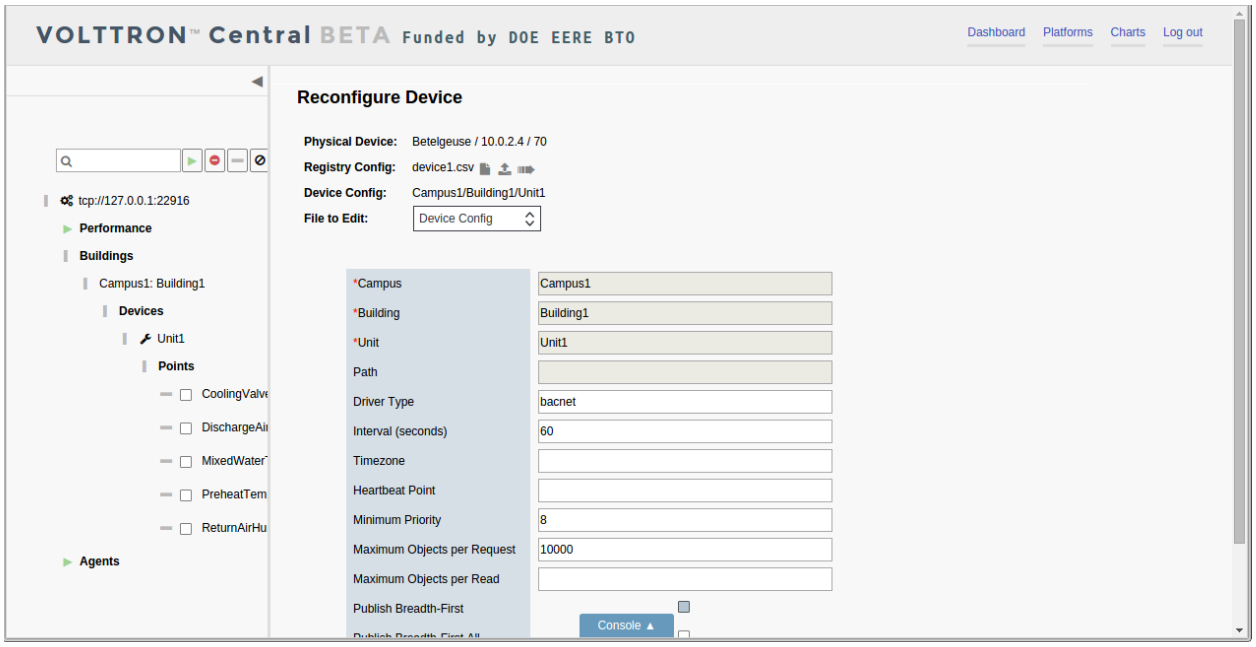
To make changes to the device configuration form, click on the *File to Edit* selector and choose *Device Config*.



Reconfigure Device

Physical Device: Betelgeuse / 10.0.2.4 / 70
Registry Config: device1.csv
Device Config: Campus1/Building1/Unit1
File to Edit: Registry Config

Volttron Poi	Writable	Units
ReturnAirHumidity	FALSE	UNKNOWN UNITS
CoolingValveOutputCommand	FALSE	UNKNOWN UNITS
MixedWaterTemperature	FALSE	UNKNOWN UNITS
PreheatTemperature	FALSE	UNKNOWN UNITS
DischargeAirStaticPressure	FALSE	UNKNOWN UNITS



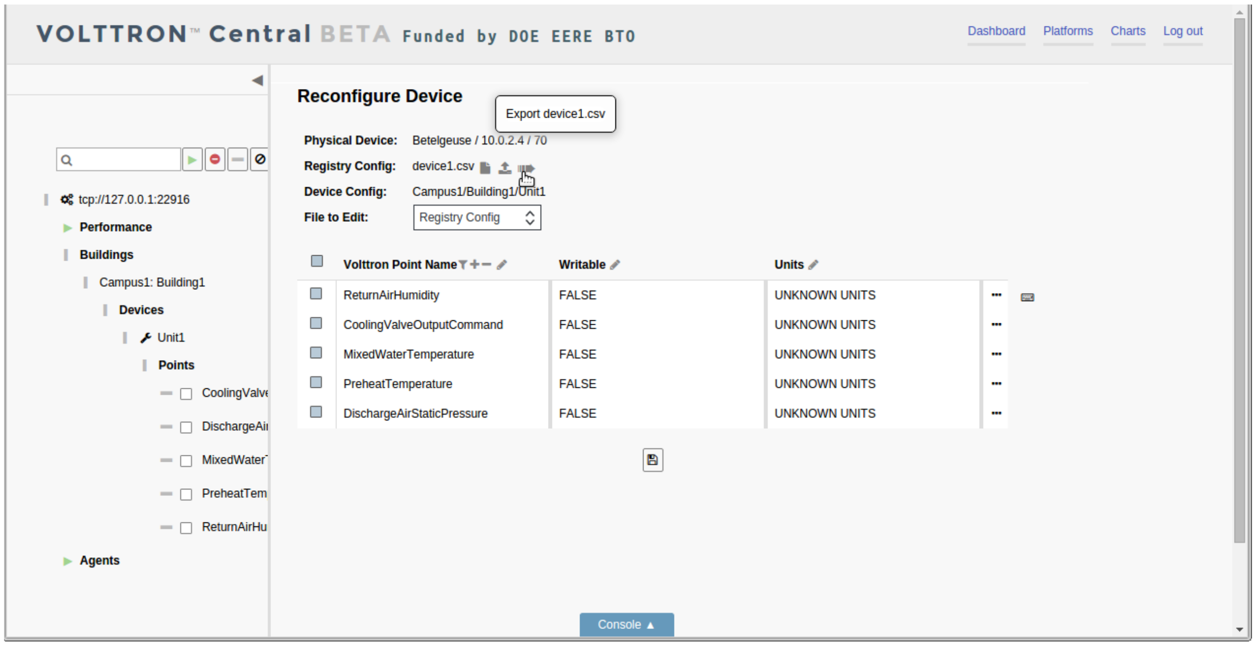
Reconfigure Device

Physical Device: Betelgeuse / 10.0.2.4 / 70
Registry Config: device1.csv
Device Config: Campus1/Building1/Unit1
File to Edit: Device Config

*Campus	Campus1
*Building	Building1
*Unit	Unit1
Path	
Driver Type	bacnet
Interval (seconds)	60
Timezone	
Heartbeat Point	
Minimum Priority	8
Maximum Objects per Request	10000
Maximum Objects per Read	
Publish Breadth-First	

Exporting Registry Configuration Files

The registry configuration file associated with a virtual device can be exported from the web browser to the computer's local storage by clicking on the *File Export* Button in the device reconfiguration view.



Reconfigure Device

Physical Device: Betelgeuse / 10.0.2.4 / 70
Registry Config: device1.csv
Device Config: Campus1/Building1/Unit1
File to Edit: Registry Config

Volttron Point Name	Writable	Units
ReturnAirHumidity	FALSE	UNKNOWN UNITS
CoolingValveOutputCommand	FALSE	UNKNOWN UNITS
MixedWaterTemperature	FALSE	UNKNOWN UNITS
PreheatTemperature	FALSE	UNKNOWN UNITS
DischargeAirStaticPressure	FALSE	UNKNOWN UNITS

VOLTTRON Central Platform Agent

The Platform Agent allows communication from a VOLTTRON Central instance. Each VOLTTRON instance that is to be controlled through the VOLTTRON Central agent should have one and only one Platform Agent. The Platform Agent must have the VIP identity of *platform.agent* which is specified by default by VOLTTRON *known identities*.

Configuration

The minimal configuration (and most likely the only used) for a Platform Agent is as follows:

```
{
    # Agent id is used in the display on volttron central.
    "agentid": "Platform 1",
}
```

VOLTTRON Central Web Services Api Documentation

VOLTTRON Central (VC) is meant to be the hub of communication within a cluster of VOLTTRON instances. VC exposes a JSON-RPC 2.0 based API that allows a user to control multiple instances of VOLTTRON.

Why JSON-RPC

SOAP messaging is unfriendly to many developers, especially those wanting to make calls in a browser from AJAX environment. We have therefore have implemented a JSON-RPC API capability to VC, as a more JSON/JavaScript friendly mechanism.

How the API is Implemented

- All calls are made through a POST to */vc/jsonrpc*
- All calls (not including the call to authenticate) will include an authorization token (a json-rpc extension).

JSON-RPC Request Payload

All posted JSON payloads will look like the following block:

```
{
    "jsonrpc": "2.0",
    "method": "method_to_invoke",
    "params": {
        "param1name": "param1value",
        "param2name": "param2value"
    },
    "id": "unique_message_id",
    "authorization": "server_authorization_token"
}
```

As an alternative, the params can be an array as illustrated by the following:

```
{
    "jsonrpc": "2.0",
    "method": "method_to_invoke",
    "params": [
        "param1value",
        "param2value"
    ],
    "id": "unique_message_id",
```

(continues on next page)

(continued from previous page)

```
    "authorization": "server_authorization_token"
}
```

For full documentation of the *Request* object please see section 4 of the JSON-RPC 2.0 specification.

JSON-RPC Response Payload

All responses shall have either an either an error response or a result response. The result key shown below can be a single instance of a JSON type, an array or a JSON object.

A result response will have the following format:

```
{
    "jsonrpc": "2.0",
    "result": "method_results",
    "id": "sent_in_unique_message_id"
}
```

An error response will have the following format:

```
{
    "jsonrpc": "2.0",
    "error": {
        "code": "standard_code_or_extended_code",
        "message": "error message"
    }
    "id": "sent_in_unique_message_id_or_null"
}
```

For full documentation of the Response object please see section 5 of the JSON-RPC 2.0 specification.

JSON-RPC Data Objects

Table 5: Platform

Key	Type	Value
uuid	string	A unique identifier for the platform.
name	string	A user defined string for the platform.
status	Status	A status object for the platform.

Table 6: PlatformDetails

Key	Type	Value
uuid	string	A unique identifier for the platform.
name	string	A user defined string for the platform.
status	Status	A status object for the platform.

Table 7: Agent

Key	Type	Value
uuid	string	A unique identifier for the agent.
name	string	Defaults to the agentid of the installed agent
tag	string	A shortcut that can be used for referencing the agent
priority	int	If this is set the agent will autostart on the instance.
process_id	int	The process id or null if not running.
status	string	A status string made by the status rpc call, on an agent.

Table 8: DiscoveryRegistryEntry

Key	Type	Value
name		
discovery_address		

Table 9: AdvancedRegistratyEntry_TODO

Key	Type	Value
name		
vip_address		

Table 10: Agent_TODO

Key	Type	Value
uuid	string	A unique identifier for the platform.
name	string	A user defined string for the platform.
status	Status	A status object for the platform.

Table 11: Building_TODO

Key	Type	Value
uuid	string	A unique identifier for the platform.
name	string	A user defined string for the platform.
status	Status	A status object for the platform.

Table 12: Device_TODO

Key	Type	Value
uuid	string	A unique identifier for the platform.
name	string	A user defined string for the platform.
status	Status	A status object for the platform.

Table 13: Status

Key	Type	Value
status	string	A value of GOOD, BAD, UNKNOWN, SUCCESS, FAIL
context	string	Provides context about what the status means (optional)

JSON-RPC API Methods

Table 14: Methods

method	parameters	returns
get_authentication	(username, password)	authentication token

Messages

Retrieve Authorization Token

```
# POST /vc/jsonrpc
{
    "jsonrpc": "2.0",
    "method": "get_authorization",
    "params": {
        "username": "dorothy",
        "password": "toto123"
    },
    "id": "someID"
}
```

Response Success

```
# 200 OK
{
    "jsonrpc": "2.0",
    "result": "somAuthorizationToken",
    "id": "someID"
}
```

Failure

```
HTTP Status Code 401
```

Register a VOLTTRON Platform Instance (Using Discovery)

```
# POST /vc/jsonrpc
{
    "jsonrpc": "2.0",
    "method": "register_instance",
    "params": {
        "discovery_address": "http://127.0.0.2:8080",
        "display_name": "foo" # Optional
    }
    "authorization": "someAuthorizationToken",
    "id": "someID"
}
```

Success

```
# 200 OK
{
    "jsonrpc": "2.0",
```

(continues on next page)

(continued from previous page)

```
    "result": {
        "status": {
            "code": "SUCCESS"
            "context": "Registered instance foo" # or the uri if not␣
↪specified.
        }
    },
    "id": "someID"
}
```

TODO: Request Registration of an External Platform

```
# POST /vc/jsonrpc
{
    "jsonrpc": "2.0",
    "method": "register_platform",
    "params": {
        "uri": "127.0.0.2:8080?serverkey=...&publickey=...&secretkey=..."
    }
    "authorization": "someAuthorizationToken",
    "id": #
}
```

Unregister a Volttron Platform Instance

```
# POST /vc/jsonrpc
{
    "jsonrpc": "2.0",
    "method": "unregister_platform",
    "params": {
        "platform_uuid": "somePlatformUuid",
    }
    "authorization": "someAuthorizationToken",
    "id": "someID"
}
```

Retrieve Managed Instances

```
#POST /vc/jsonrpc
{
    "jsonrpc": "2.0",
    "method": "list_platforms",
    "authorization": "someAuthorizationToken",
    "id": #
}
```

Response Success

```
200 OK
{
    "jsonrpc": "2.0",
    "result": [
        {
            "name": "platform1",
            "uuid": "abcd1234-ef56-ab78-cd90-efabcd123456",
            "health": {
```

(continues on next page)

(continued from previous page)

```
                "status": "GOOD",
                "context": null,
                "last_updated": "2016-04-27T19:47:05.184997+00:00"
            }
        },
        {
            "name": "platform2",
            "uuid": "0987fedc-65ba-43fe-21dc-098765bafedc",
            "health": {
                "status": "BAD",
                "context": "Expected 9 agents running, but only 5 are",
                "last_updated": "2016-04-27T19:47:05.184997+00:00",
            }

        },
        {
            "name": "platform3",
            "uuid": "0000aaaa-1111-bbbb-2222-cccc3333dddd",
            "health": {
                "status": "GOOD",
                "context": "Currently scraping 20 devices",
                "last_updated": "2016-04-27T19:47:05.184997+00:00",
            }
        }
    ],
    "id": #
}
```

TODO: change response Retrieve Installed Agents From platform1

```
# POST /vc/jsonrpc
{
    "jsonrpc": "2.0",
    "method": "platforms.uuid.abcd1234-ef56-ab78-cd90-efabcd123456.list_agents",
    "authorization": "someAuthorizationToken",
    "id": #
}
```

Response Success

```
200 OK
{
    "jsonrpc": "2.0",
    "result": [
        {
            "name": "HelloAgent",
            "identity": "helloagent-0.0_1",
            "uuid": "a1b2c3d4-e5f6-a7b8-c9d0-e1f2a3b4c5d6",
            "process_id": 3142,
            "error_code": null,
            "is_running": true,
            "permissions": {
                "can_start": true,
                "can_stop": true,
                "can_restart": true,
                "can_remove": true
            }
```

(continues on next page)

(continued from previous page)

```
            "health": {
               "status": "GOOD",
               "context": null
            }
        },
        {
            "name": "Historian",
            "identity": "sqlhistorianagent-3.5.0_1",
            "uuid": "a1b2c3d4-e5f6-a7b8-c9d0-e1f2a3b4c5d6",
            "process_id": 3143,
            "error_code": null,
            "is_running": true,
            "permissions": {
               "can_start": true,
               "can_stop": true,
               "can_restart": true,
               "can_remove": true
            }

            "health": {
               "status": "BAD",
               "context": "No publish in last 5 minutes"
            }
        },
        {
           "name": "VolltronCentralPlatform",
           "identity": "platform.agent",
           "uuid": "a1b2c3d4-e5f6-a7b8-c9d0-e1f2a3b4c5d6",
           "process_id": 3144,
           "error_code": null,
           "is_running": true,
           "permissions": {
              "can_start": false,
              "can_stop": false,
              "can_restart": true,
              "can_remove": false
           }
           "health": {
              "status": "BAD",
              "context": "One agent has reported bad status"
           }
       },
       {
            "name": "StoppedAgent-0.1",
            "identity": "stoppedagent-0.1_1",
            "uuid": "a1b2c3d4-e5f6-a7b8-c9d0-e1f2a3b4c5d6",
            "process_id": null,
            "error_code": 0,
            "is_running": false,s
            "health": {
               "status": "UNKNOWN",
               "context": "Error code -15"
            }
           "permissions": {
              "can_start": true,
              "can_stop": false,
              "can_restart": true,
```

(continues on next page)

(continued from previous page)

```
                "can_remove": true
            }
        }
    ],
    "id": #
}
```

TODO: Start An Agent

```
# POST /vc/jsonrpc
{
    "jsonrpc": "2.0",
    "method": "platforms.uuid.0987fedc-65ba-43fe-21dc-098765bafedc.start_agent",
    "params": ["a1b2c3d4-e5f6-a7b8-c9d0-e1f2a3b4c5d6"],
    "authorization": "someAuthorizationToken",
    "id": #
}
```

Response Success

```
200 OK
{
    "jsonrpc": "2.0",
    "result": {
        "process_id": 1000,
        "return_code": null
    },
    "id": #
}
```

TODO: Stop An Agent

```
# POST /vc/jsonrpc
{
    "jsonrpc": "2.0",
    "method": "platforms.uuid.0987fedc-65ba-43fe-21dc-098765bafedc.stop_agent",
    "params": ["a1b2c3d4-e5f6-a7b8-c9d0-e1f2a3b4c5d6"],
    "authorization": "someAuthorizationToken",
    "id": #
}
```

Response Success

```
200 OK
{
    "jsonrpc": "2.0",
    "result": {
        "process_id": 1000,
        "return_code": 0
    },
    "id": #
}
```

TODO: Remove An Agent

```
# POST /vc/jsonrpc
{
```

(continues on next page)

(continued from previous page)

```
    "jsonrpc": "2.0",
    "method": "platforms.uuid.0987fedc-65ba-43fe-21dc-098765bafedc.remove_agent",
    "params": ["a1b2c3d4-e5f6-a7b8-c9d0-e1f2a3b4c5d6"],
    "authorization": "someAuthorizationToken",
    "id": #
}
```

Response Success

```
200 OK
{
    "jsonrpc": "2.0",
    "result": {
        "process_id": 1000,
        "return_code": 0
    },
    "id": #
}
```

TODO: Retrieve Running Agents

```
# POST /vc/jsonrpc
{
    "jsonrpc": "2.0",
    "method": "platforms.uuid.0987fedc-65ba-43fe-21dc-098765bafedc.status_agents",
    "authorization": "someAuthorizationToken",
    "id": #
}
```

Response Success

```
200 OK
{
    "jsonrpc": "2.0",
    "result": [
        {
            "name": "RunningAgent",
            "uuid": "a1b2c3d4-e5f6-a7b8-c9d0-e1f2a3b4c5d6"
            "process_id": 1234,
            "return_code": null
        },
        {
            "name": "StoppedAgent",
            "uuid": "a1b2c3d4-e5f6-a7b8-c9d0-e1f2a3b4c5d6"
            "process_id": 1000,
            "return_code": 0
        }
    ],
    "id": #
}
```

TODO: currently getting 500 error Retrieve An Agent's RPC Methods

```
# POST /vc/jsonrpc
{
    "jsonrpc": "2.0",
```

(continues on next page)

(continued from previous page)

```
    "method": "platforms.uuid.0987fedc-65ba-43fe-21dc-098765bafedc.agents.
↪uuid.a1b2c3d4-e5f6-a7b8-c9d0-e1f2a3b4c5d6.inspect",
    "authorization": "someAuthorizationToken",
    "id": #
}
```

Response Success

```
200 OK
{
    "jsonrpc": "2.0",
    "result": [
        {
            "method": "sayHello",
            "params": {
                "name": "string"
            }
        }
    ],
    "id": #
}
```

TODO: Perform Agent Action

```
# POST /vc/jsonrpc
{
    "jsonrpc": "2.0",
    "method": "platforms.uuid.0987fedc-65ba-43fe-21dc-098765bafedc.agents.uuid.
↪a1b2c3d4-e5f6-a7b8-c9d0-e1f2a3b4c5d6.methods.say_hello",
    "params": {
        "name": "Dorothy"
    },
    "authorization": "someAuthorizationToken",
    "id": #
}
```

Success Response

```
200 OK
{
    "jsonrpc": "2.0",
    "result": "Hello, Dorothy!",
    "id": #
}
```

TODO: Install Agent

```
# POST /vc/jsonrpc
{
    "jsonrpc": "2.0",
    "method": "platforms.uuid.0987fedc-65ba-43fe-21dc-098765bafedc.install",
    "params": {
        "files": [
            {
                "file_name": "helloagent-0.1-py2-none-any.whl",
                "file": "data:application/octet-stream;base64,..."
```

(continues on next page)

(continued from previous page)

```
                },
                {
                    "file_name": "some-non-wheel-file.txt",
                    "file": "data:application/octet-stream;base64,..."
                },
                ...
            ],
        }
    "authorization": "someAuthorizationToken",
    "id": #
}
```

Success Response

```
200 OK
{
    "jsonrpc": "2.0",
    "result": {
        [
            {
                "uuid": "a1b2c3d4-e5f6-a7b8-c9d0-e1f2a3b4c5d6"
            },
            {
                "error": "Some error message"
            },
            ...
        ]
    },
    "id": #
}
```

1.22 Operations

Operations agents assist with the operations of the platform systems and provide alerts for various platform and environmental conditions. For details on each, please refer to the corresponding documents.

1.22.1 Emailer Agent

Emailer agent is responsible for sending emails for an instance. It has been written so that any agent on the instance can send emails through it via the "send_email" method or through the pubsub message bus using the topic "platform/send_email".

By default any alerts will be sent through this agent. In addition all emails will be published to the "record/sent_email" topic for a historian to be able to capture that data.

Configuration

A typical configuration for this agent is as follows. We need to specify the SMTP server address, email address of the sender, email addresses of all the recipients and minimum time for duplicate emails based upon the key.

```
{
    "smtp-address": "smtp.foo.com",
    "from-address": "billy@foo.com",
    "to-addresses": ["ann@foo.com", "bob@gmail.com"],
    "allow-frequency-minutes": 10
}
```

Finally package, install and start the agent. For more details, see *Agent Creation Walk-through*

1.22.2 Failover Agent

Introduction

The failover agent provides a generic high availability option to VOLTTRON. When the **primary** platform becomes inactive the **secondary** platform will start an installed agent.

Standard Failover

There are two behavior patterns implemented in the agent. In the default configuration, the secondary instance will ask Volttron Central to verify that the primary instance is down. This helps to avoid a split brain scenario. If neither Volttron Central nor the other failover instance is reachable then the failover agent will stop the agent it is managing. These states are shown in the tables below.

Primary Behavior

	VC Up	VC Down
Secondary Up	start	start
Secondary Down	start	stop

Secondary Behavior

	VC Up	VC Down
Primary Up	stop	stop
Primary Down	Verify with VC before starting	stop

Simple Failover

There is also a *simple* configuration available that does not involve coordination with Volttron Central. The secondary agent will start its managed agent if believes the primary to be inactive. The simple primary always has its managed agent started.

Configuration

Failover behavior is set in the failover agent's configuration file. Example primary and secondary configuration files are shown below.

```
{                                               |     {
    "agent_id": "primary",                      |         "agent_id": "secondary",
    "simple_behavior": true,                    |         "simple_behavior": true,
                                                |
```

(continues on next page)

(continued from previous page)

```
    "remote_vip": "tcp://127.0.0.1:8001",    |         "remote_vip": "tcp://127.0.0.
↪1:8000",
    "remote_serverkey": "",                  |         "remote_serverkey": "",
                                             |
    "agent_vip_identity": "platform.driver",|         "agent_vip_identity": "platform.
↪driver",
                                             |
    "heartbeat_period": 10,                  |         "heartbeat_period": 10,
                                             |
    "timeout": 120                           |         "timeout": 120
}                                            |     }
```

- **agent_id** - primary **or** secondary
- **simple_behavior** - Switch to turn on or off simple behavior. Both instances should match.
- **remote_vip** - Address where *remote_id* can be reached.
- **remote_serverkey** - The public key of the platform where *remote_id* lives.
- **agent_vip_identity** - The vip identity of the agent that we want to manage.
- **heartbeat_period** - Send a message to *remote_id* with this period. Measured in seconds.
- **timeout** - Consider a platform inactive if a heartbeat has not been received for *timeout* seconds.

1.22.3 File Watch Publisher Agent

Introduction

FileWatchPublisher agent watches files for changes and publishes those changes per line on the corresponding topics. Files and topics should be provided in the configuration.

Configuration

A simple configuration for FileWatchPublisher with two files to monitor is as follows:

```
{
    "files": [
        {
            "file": "/var/log/syslog",
            "topic": "platform/syslog"
        },
        {
            "file": "/home/volttron/tempfile.txt",
            "topic": "temp/filepublisher"
        }
    ]
}
```

Using this example configuration, FileWatchPublisher will watch syslog and tempFile.txt files and publish the changes per line on their respective topics.

1.22.4 Message Debugging

VOLTTRON agent messages are routed over the VOLTTRON message bus. The Message Debugger Agent provides enhanced examination of this message stream's contents as an aid to debugging and troubleshooting agents and drivers.

This feature is implemented to provide visibility into the ZeroMQ message bus. The RabbitMQ message bus includes methods for message debugging by default in the RabbitMQ management UI.

When enabled, the Message Debugger Agent captures and records each message as it is routed. A second process, Message Viewer, provides a user interface that optimizes and filters the resulting data stream, either in real time or retrospectively, and displays its contents.

The Message Viewer can convey information about high-level interactions among VOLTTRON agents, representing the message data as conversations that can be filtered and/or expanded. A simple RPC call involving 4 individual message send/receive segments can be displayed as a single row, which can then be expanded to drill down into the message details. This results in a higher-level, easier-to-obtain view of message bus activity than might be gleaned by using grep on verbose log files.

Pub/Sub interactions can be summarized by topic, including counts of messages published during a given capture period by sender, receiver and topic.

Another view displays the most-recently-published message, or message exchange, that satisfies the current filter criteria, continuously updated as new messages are routed.

Enabling the Message Debugger

In order to use the Message Debugger, two steps are required:

- VOLTTRON must have been started with a `--msgdebug` command line option.
- The Message Debugger Agent must be running.

When VOLTTRON has been started with `--msgdebug`, its Router publishes each message to an IPC socket for which the Message Debugger Agent is a subscriber. This is kept disabled by default because it consumes a significant quantity of CPU and memory resources, potentially affecting VOLTTRON timing and performance. So as a general rule, the `--msgdebug` option should be employed during development/debugging only, and should not be left enabled in a production environment.

Example of starting VOLTTRON with the `--msgdebug` command line option:

```
(volttron) ./start-volttron ``--msgdebug``
```

If VOLTTRON is running in this mode, the stream of routed messages is available to a subscribing Message Debugger Agent. It can be started from volttron-ctl in the same fashion as other agents, for example:

```
(volttron) $ vctl status
   AGENT                        IDENTITY                  TAG                      STATUS
fd listeneragent-3.2            listener                  listener
08 messagedebuggeragent-0.1     platform.messagedebugger platform.messagedebugger
e1 vcplatformagent-3.5.4        platform.agent            vcp
47 volttroncentralagent-3.5.5 volttron.central          vc

(volttron) $ vctl start 08
Starting 089c53f0-f225-4608-aecb-3e86e0df30eb messagedebuggeragent-0.1

(volttron) $ vctl status
   AGENT                        IDENTITY                  TAG                      STATUS
fd listeneragent-3.2            listener                  listener
```

(continues on next page)

(continued from previous page)

```
08 messagedebuggeragent-0.1   platform.messagedebugger platform.messagedebugger␣
↪running [43498]
e1 vcplatformagent-3.5.4      platform.agent           vcp
47 volttroncentralagent-3.5.5 volttron.central         vc
```

See *Agent Creation Walk-through* for further details on installing and starting agents from *vctl*.

Once the Message Debugger Agent is running, it begins capturing message data and writing it to a SQLite database.

Message Viewer

The Message Viewer is a separate process that interacts with the Message Debugger Agent primarily via VOLTTRON RPC calls. These calls allow it to request and report on filtered sets of message data.

Since the Agent's RPC methods are available for use by any VOLTTRON agent, the Message Viewer is really just one example of a Message Debugger information consumer. Other viewers could be created to satisfy a variety of specific debugging needs. For example, a viewer could support browser-based message debugging with a graphical user interface, or a viewer could transform message data into PCAP format for consumption by WireShark.

The Message Viewer in *services/ops/MessageDebuggerAgent/messageviewer/viewer.py* implements a command-line UI, subclassing Python's `Cmd` class. Most of the command-line options that it displays result in a MessageDebugger-Agent RPC request. The Message Viewer formats and displays the results.

In Linux, the Message Viewer can be started as follows, and displays the following menu:

```
(volttron) $ cd services/ops/MessageDebuggerAgent/messageviewer
(volttron) $ python viewer.py
Welcome to the MessageViewer command line. Supported commands include:
     display_message_stream
     display_messages
     display_exchanges
     display_exchange_details
     display_session_details_by_agent <session_id>
     display_session_details_by_topic <session_id>

     list_sessions
     set_verbosity <level>
     list_filters
     set_filter <filter_name> <value>
     clear_filters
     clear_filter <filter_name>

     start_streaming
     stop_streaming
     start_session
     stop_session
     delete_session <session_id>
     delete_database

     help
     quit
Please enter a command.
Viewer>
```

Command-Line Help

The Message Viewer offers two help levels. Simply typing `help` gives a list of available commands. If a command name is provided as an argument, advice is offered on how to use that command:

```
Viewer> help

Documented commands (type help <topic>):
========================================
clear_filter              display_messages                  set_filter
clear_filters             display_session_details_by_agent  set_verbosity
delete_database           display_session_details_by_topic  start_session
delete_session            help                              start_streaming
display_exchange_details  list_filters                      stop_session
display_exchanges         list_sessions                     stop_streaming
display_message_stream    quit

Viewer> help set_filter

            Set a filter to a value; syntax is: set_filter <filter_name> <value>

            Some recognized filters include:
            . freq <n>: Use a single-line display, refreshing every <n> seconds (<n>␣
↪can be floating point)
            . session_id <n>: Display Messages and Exchanges for the indicated␣
↪debugging session ID only
            . results_only <n>: Display Messages and Exchanges only if they have a␣
↪result
            . sender <agent_name>
            . recipient <agent_name>
            . device <device_name>
            . point <point_name>
            . topic <topic_name>: Matches all topics that start with the supplied
↪<topic_name>
            . starttime <YYYY-MM-DD HH:MM:SS>: Matches rows with timestamps after the␣
↪supplied time
            . endtime <YYYY-MM-DD HH:MM:SS>: Matches rows with timestamps before the␣
↪supplied time
            . (etc. -- see the structures of DebugMessage and DebugMessageExchange)
```

Debug Sessions

The Message Debugger Agent tags each message with a debug session ID (a serial number), which groups a set of messages that are bounded by a start time and an end time. The `list_sessions` command describes each session in the database:

```
Viewer> list_sessions
  rowid        start_time                  end_time                    num_messages
  1            2017-03-20 17:07:13.867951  -                           2243
  2            2017-03-20 17:17:35.725224  -                           1320
  3            2017-03-20 17:33:35.103204  2017-03-20 17:46:15.657487  12388
```

A new session is started by default when the Agent is started. After that, the `stop_session` and `start_session` commands can be used to create new session boundaries. If the Agent is running but no session is active (i.e., because `stop_session` was used to stop it), messages are still written to the database, but they have no session ID.

Filtered Display

The `set_filter <property> <value>` command enables filtered display of messages. A variety of properties can be filtered.

In the following example, message filters are defined by session_id and sender, and the `display_messages` command displays the results:

```
Viewer> set_filter session_id 4
Set filters to {'session_id': '4'}
Viewer> set_filter sender testagent
Set filters to {'sender': 'testagent', 'session_id': '4'}
Viewer> display_messages
  timestamp    direction    sender       recipient                 request_id        ␣
↪             subsystem    method          topic                     device       ␣
↪point         result
  11:51:00     incoming     testagent    messageviewer.connection  -                 ␣
↪             RPC          pubsub.sync     -                         -             -␣
↪           -
  11:51:00     outgoing     testagent    pubsub                    -                 ␣
↪             RPC          pubsub.push     -                         -             -␣
↪           -
  11:51:00     incoming     testagent    platform.driver          ␣
↪1197886248649056372.284581685   RPC          get_point       -                      ␣
↪   chargepoint1  Status       -
  11:51:01     outgoing     testagent    platform.driver          ␣
↪1197886248649056372.284581685   RPC          -               -                      ␣
↪   -             -            AVAILABLE
  11:51:01     incoming     testagent    pubsub                   ␣
↪1197886248649056373.284581649   RPC          pubsub.publish  test_topic/test_
↪subtopic   -             -            -
  11:51:01     outgoing     testagent    pubsub                   ␣
↪1197886248649056373.284581649   RPC          -               -                      ␣
↪   -             -            None
```

Debug Message Exchanges

A VOLTTRON message's request ID is not unique to a single message. A group of messages in an "exchange" (essentially a small conversation among agents) will often share a common request ID, for instance during RPC request/response exchanges.

The following example uses the same filters as above, and then uses `display_exchanges` to display a single line for each message exchange, reducing the number of displayed rows from 6 to 2. Note that not all messages have a request ID; messages with no ID are absent from the responses to exchange queries.

```
Viewer> list_filters
{'sender': 'testagent', 'session_id': '4'}
Viewer> display_exchanges
  sender       recipient        sender_time  topic                     device       ␣
↪point         result
  testagent    platform.driver  11:51:00     -                         chargepoint1 ␣
↪Status        AVAILABLE
  testagent    pubsub           11:51:01     test_topic/test_subtopic  -             -
↪             None
```

Special Filters

Most filters that can be set with the `set_filter` command are simple string matches on one or another property of a message. Some filters have special characteristics, though. The `set_filter starttime <timestamp>` and `set_filter endtime <timestamp>` filters are inequalities that test for messages after a start time or before an end time.

In the following example, note the use of quotes in the endtime value supplied to set_filter. Any filter value can be delimited with quotes. Quotes must be used when a value contains embedded spaces, as is the case here:

```
Viewer> list_sessions
  rowid         start_time                  end_time                    num_messages
  1             2017-03-20 17:07:13.867951  -                           -
  2             2017-03-20 17:17:35.725224  -                           -
  3             2017-03-21 11:48:33.803288  2017-03-21 11:50:57.181136  6436
  4             2017-03-21 11:50:59.656693  2017-03-21 11:51:05.934895  450
  5             2017-03-21 11:51:08.431871  -                           74872
  6             2017-03-21 12:17:30.568260  -                           2331
Viewer> set_filter session_id 5
Set filters to {'session_id': '5'}
Viewer> set_filter sender testagent
Set filters to {'sender': 'testagent', 'session_id': '5'}
Viewer> set_filter endtime '2017-03-21 11:51:30'
Set filters to {'endtime': '2017-03-21 11:51:30', 'sender': 'testagent', 'session_id
↪': '5'}
Viewer> display_exchanges
  sender        recipient         sender_time  topic                       device        ␣
↪point          result
  testagent     platform.driver   11:51:11     -                           chargepoint1 ␣
↪Status         AVAILABLE
  testagent     pubsub            11:51:11     test_topic/test_subtopic    -             -
↪              None
  testagent     platform.driver   11:51:25     -                           chargepoint1 ␣
↪Status         AVAILABLE
  testagent     pubsub            11:51:25     test_topic/test_subtopic    -             -
↪              None
  testagent     platform.driver   11:51:26     -                           chargepoint1 ␣
↪Status         AVAILABLE
  testagent     pubsub            11:51:26     test_topic/test_subtopic    -             -
↪              None
```

Another filter type with special behavior is `set_filter topic <name>`. Ordinarily, filters do an exact match on a message property. Since message topics are often expressed as hierarchical substrings, though, the `topic` filter does a substring match on the left edge of a message's topic, as in the following example:

```
Viewer> set_filter topic test_topic
Set filters to {'topic': 'test_topic', 'endtime': '2017-03-21 11:51:30', 'sender':
↪'testagent', 'session_id': '5'}
Viewer> display_exchanges
  sender        recipient     sender_time  topic                       device        point␣
↪        result
  testagent     pubsub        11:51:11     test_topic/test_subtopic    -             -    ␣
↪        None
  testagent     pubsub        11:51:25     test_topic/test_subtopic    -             -    ␣
↪        None
  testagent     pubsub        11:51:26     test_topic/test_subtopic    -             -    ␣
↪        None
Viewer>
```

Another filter type with special behavior is `set_filter results_only 1`. In the JSON representation of a response to an RPC call, for example an RPC call to a Master Driver interface, the response to the RPC request typically appears as the value of a 'result' tag. The `results_only` filter matches only those messages that have a non-empty value for this tag.

In the following example, note that when the `results_only` filter is set, it is given a value of '1'. This is actually a meaningless value that gets ignored. It must be supplied because the set_filter command syntax requires that a value be supplied as a parameter.

In the following example, note the use of `clear_filter <property>` to remove a single named filter from the list of filters that are currently in effect. There is also a `clear_filters` command, which clears all current filters.

```
Viewer> clear_filter topic
Set filters to {'endtime': '2017-03-21 11:51:30', 'sender': 'testagent', 'session_id
↪': '5'}
Viewer> set_filter results_only 1
Set filters to {'endtime': '2017-03-21 11:51:30', 'sender': 'testagent', 'session_id
↪': '5', 'results_only': '1'}
Viewer> display_exchanges
  sender        recipient        sender_time  topic         device        point        ␣
↪result
  testagent     platform.driver  11:51:11     -             chargepoint1  Status       ␣
↪AVAILABLE
  testagent     platform.driver  11:51:25     -             chargepoint1  Status       ␣
↪AVAILABLE
  testagent     platform.driver  11:51:26     -             chargepoint1  Status       ␣
↪AVAILABLE
```

Streamed Display

In addition to exposing a set of RPC calls that allow other agents (like the Message Viewer) to query the Message Debugger Agent's SQLite database of recent messages, the Agent can also publish messages in real time as it receives them.

This feature is disabled by default due to the large quantity of data that it might need to handle. When it is enabled, the Agent applies the filters currently in effect to each message as it is received, and re-publishes the transformed, ready-for-debugging message to a socket if it meets the filter criteria. The Message Viewer can listen on that socket and display the message stream as it arrives.

In the following `display_message_stream` example, the Message Viewer displays all messages sent by the agent named 'testagent', as they arrive. It continues to display messages until execution is interrupted with ctrl-C:

```
Viewer> clear_filters
Set filters to {}
Viewer> set_filter sender testagent
Set filters to {'sender': 'testagent'}
Viewer> display_message_stream
Streaming debug messages
  timestamp    direction    sender       recipient    request_id   subsystem    ␣
↪method        topic        device       point        result
  12:28:58     outgoing     testagent    pubsub       -            RPC          ␣
↪pubsub.push   -            -            -            -
  12:28:58     incoming     testagent    platform.dr  11978862486  RPC            get_
↪point    -            chargepoint  Status       -
                                         iver         49056826.28                  ␣
↪                   1
                                                      4581713
```

(continues on next page)

(continued from previous page)

```
  12:28:58     outgoing     testagent    platform.dr  11978862486  RPC          -     ␣
↪       -            -            -             AVAILABLE
                                         iver         49056826.28
                                                      4581713
  12:28:58     incoming     testagent    pubsub       11978862486  RPC         ␣
↪pubsub.publ  test_topic/  -            -            -
                                                      49056827.28              ish   ␣
↪       test_subtop
                                                      4581685                        ␣
↪       ic
  12:28:58     outgoing     testagent    pubsub       11978862486  RPC          -     ␣
↪       -            -            -             None
                                                      49056827.28
                                                      4581685
  12:28:58     outgoing     testagent    pubsub       -            RPC         ␣
↪pubsub.push  -            -            -            -
^CViewer> stop_streaming
Stopped streaming debug messages
```

(Note the use of wrapping in the column formatting. Since these messages aren't known in advance, the Message Viewer has incomplete information about how wide to make each column. Instead, it must make guesses based on header widths, data widths in the first row received, and min/max values, and then wrap the data when it overflows the column boundaries.)

Single-Line Display

Another filter with special behavior is `set_filter freq <seconds>`. This filter, which takes a number N as its value, displays only one row, the most recently captured row that satisfies the filter criteria. (Like other filters, this filter can be used with either `display_messages` or `display_exchanges`.) It then waits N seconds, reissues the query, and overwrites the old row with the new one. It continues this periodic single-line overwritten display until it is interrupted with ctrl-C:

```
Viewer> list_filters
{'sender': 'testagent'}
Viewer> set_filter freq 10
Set filters to {'freq': '10', 'sender': 'testagent'}
Viewer> display_exchanges
  sender       recipient    sender_time  topic                    device       point␣
↪       result
  testagent    pubsub       12:31:28     test_topic/test_subtopic  -            -     ␣
↪       None
```

(Again, the data isn't known in advance, so the Message Viewer has to guess the best width of each column. In this single-line display format, data gets truncated if it doesn't fit, because no wrapping can be performed – only one display line is available.)

Displaying Exchange Details

The `display_exchange_details <request_id>` command provides a way to get more specific details about an exchange, i.e. about all messages that share a common request ID. At low or medium verbosity, when this command is used (supplying the relevant request ID, which can be obtained from the output of other commands), it displays one row for each message:

```
Viewer> set_filter sender testagent
Set filters to {'sender': 'testagent', 'session_id': '4'}
Viewer> display_messages
  timestamp    direction    sender       recipient                  request_id       ␣
↪             subsystem    method         topic                      device        ␣
↪point         result
  11:51:00     incoming     testagent    messageviewer.connection  -                 ␣
↪             RPC          pubsub.sync     -                         -             - ␣
↪             -
  11:51:00     outgoing     testagent    pubsub                     -                ␣
↪             RPC          pubsub.push     -                         -             - ␣
↪             -
  11:51:00     incoming     testagent    platform.driver           ␣
↪1197886248649056372.284581685  RPC          get_point       -                   ␣
↪  chargepoint1  Status        -
  11:51:01     outgoing     testagent    platform.driver           ␣
↪1197886248649056372.284581685  RPC          -               -                   ␣
↪  -             -             AVAILABLE
  11:51:01     incoming     testagent    pubsub                    ␣
↪1197886248649056373.284581649  RPC          pubsub.publish  test_topic/test_
↪subtopic  -             -             -
  11:51:01     outgoing     testagent    pubsub                    ␣
↪1197886248649056373.284581649  RPC          -               -                   ␣
↪  -             -             None
Viewer> display_exchange_details 1197886248649056373.284581649
  timestamp    direction    sender       recipient    request_id                ␣
↪subsystem     method          topic                    device        point      ␣
↪result
  11:51:01     incoming     testagent    pubsub       1197886248649056373.284581649 ␣
↪RPC          pubsub.publish  test_topic/test_subtopic  -             -            -
  11:51:01     outgoing     testagent    pubsub       1197886248649056373.284581649 ␣
↪RPC          -               -                        -             -           ␣
↪None
```

At high verbosity, `display_exchange_details` switches display formats, showing all properties for each message in a json-like dictionary format:

```
Viewer> set_verbosity high
Set verbosity to high
Viewer> display_exchange_details 1197886248649056373.284581649

{
    "data": "{\"params\":{\"topic\":\"test_topic/test_subtopic\",\"headers\":{\"Date\
↪":\"2017-03-21T11:50:56.293830\",\"max_compatible_version\":\"\",\"min_compatible_
↪version\":\"3.0\"},\"message\":[{\"property_1\":1,\"property_2\":2},{\"property_3\
↪":3,\"property_4\":4}],\"bus\":\"\"},\"jsonrpc\":\"2.0\",\"method\":\"pubsub.
↪publish\",\"id\":\"15828311332408898779.284581649\"}",
    "device": "",
    "direction": "incoming",
    "frame7": "",
    "frame8": "",
    "frame9": "",
    "headers": "{u'Date': u'2017-03-21T11:50:56.293830', u'max_compatible_version': u'
↪', u'min_compatible_version': u'3.0'}",
    "message": "[{u'property_1': 1, u'property_2': 2}, {u'property_3': 3, u'property_4
↪': 4}]",
    "message_size": 374,
```

(continues on next page)

(continued from previous page)

```
    "message_value": "{u'property_1': 1, u'property_2': 2}",
    "method": "pubsub.publish",
    "params": "{u'topic': u'test_topic/test_subtopic', u'headers': {u'Date': u'2017-
↪03-21T11:50:56.293830', u'max_compatible_version': u'', u'min_compatible_version': u
↪'3.0'}, u'message': [{u'property_1': 1, u'property_2': 2}, {u'property_3': 3, u
↪'property_4': 4}], u'bus': u''}",
    "point": "",
    "point_value": "",
    "recipient": "pubsub",
    "request_id": "11978862486490563 73.284581649",
    "result": "",
    "sender": "testagent",
    "session_id": 4,
    "subsystem": "RPC",
    "timestamp": "2017-03-21 11:51:01.027623",
    "topic": "test_topic/test_subtopic",
    "user_id": "",
    "vip_signature": "VIP1"
}

{
    "data": "{\"params\":{\"topic\":\"test_topic/test_subtopic\",\"headers\":{\"Date\
↪":\"2017-03-21T11:50:56.293830\",\"max_compatible_version\":\"\",\"min_compatible_
↪version\":\"3.0\"},\"message\":[{\"property_1\":1,\"property_2\":2},{\"property_3\
↪":3,\"property_4\":4}],\"bus\":\"\"},\"jsonrpc\":\"2.0\",\"method\":\"pubsub.
↪publish\",\"id\":\"15828311332408898779.284581649\"}",
    "device": "",
    "direction": "outgoing",
    "frame7": "",
    "frame8": "",
    "frame9": "",
    "headers": "{u'Date': u'2017-03-21T11:50:56.293830', u'max_compatible_version': u'
↪', u'min_compatible_version': u'3.0'}",
    "message": "[{u'property_1': 1, u'property_2': 2}, {u'property_3': 3, u'property_4
↪': 4}]",
    "message_size": 383,
    "message_value": "{u'property_1': 1, u'property_2': 2}",
    "method": "pubsub.publish",
    "params": "{u'topic': u'test_topic/test_subtopic', u'headers': {u'Date': u'2017-
↪03-21T11:50:56.293830', u'max_compatible_version': u'', u'min_compatible_version': u
↪'3.0'}, u'message': [{u'property_1': 1, u'property_2': 2}, {u'property_3': 3, u
↪'property_4': 4}], u'bus': u''}",
    "point": "",
    "point_value": "",
    "recipient": "testagent",
    "request_id": "11978862486490563 73.284581649",
    "result": "",
    "sender": "pubsub",
    "session_id": 4,
    "subsystem": "RPC",
    "timestamp": "2017-03-21 11:51:01.031183",
    "topic": "test_topic/test_subtopic",
    "user_id": "testagent",
    "vip_signature": "VIP1"
}
```

Verbosity

As mentioned in the previous section, Agent and Viewer behavior can be adjusted by changing the current verbosity with the `set_verbosity <level>` command. The default verbosity is low. low, medium and high levels are available:

```
Viewer> set_verbosity high
Set verbosity to high
Viewer> set_verbosity none
Invalid verbosity choice none; valid choices are ['low', 'medium', 'high']
```

At high verbosity, the following query formatting rules are in effect:

- When displaying timestamps, display the full date and time (including microseconds), not just HH:MM:SS.
- In responses to display_message_exchanges, use dictionary format (see example in previous section).
- Display all columns, not just "interesting" columns (see the list below).
- Don't exclude messages/exchanges based on excluded senders/receivers (see the list below).

At medium or low verbosity:

- When displaying timestamps, display HH:MM:SS only.
- In responses to display_message_exchanges, use table format.
- Display "interesting" columns only (see the list below).
- Exclude messages/exchanges for certain senders/receivers (see the list below).

At low verbosity:

- If > 1000 objects are returned by a query, display the count only.

The following "interesting" columns are displayed at low and medium verbosity levels (at high verbosity levels, all properties are displayed):

```
Debug Message       Debug Message Exchange      Debug Session

timestamp           sender_time                 rowid
direction                                       start_time
sender              sender                      end_time
recipient           recipient                   num_messages
request_id
subsystem
method
topic               topic
device              device
point               point
result              result
```

Messages from the following senders, or to the following receivers, are excluded at low and medium verbosity levels:

```
Sender                                  Receiver

(empty)                                 (empty)
None
control                                 control
config.store                            config.store
pubsub
control.connection
```

(continues on next page)

(continued from previous page)

```
messageviewer.connection
platform.messagedebugger
platform.messagedebugger.loopback_rpc
```

These choices about which columns are "interesting" and which senders/receivers are excluded are defined as parameters in Message Viewer, and can be adjusted as necessary by changing global value lists in viewer.py.

Session Statistics

One useful tactic for starting at a summary level and drilling down is to capture a set of messages for a session and then examine the counts of sending and receiving agents, or sending agents and topics. This gives hints on which values might serve as useful filters for more specific queries.

The `display_session_details_by_agent <session_id>` command displays statistics by sending and receiving agent. Sending agents are table columns, and receiving agents are table rows. This query also applies whatever filters are currently in effect; the filters can reduce the counts and can also reduce the number of columns and rows.

The following example shows the command being used to list all senders and receivers for messages sent during debug session 7:

```
Viewer> list_sessions
  rowid       start_time                 end_time                   num_messages
  1           2017-03-20 17:07:13.867951 -                          -
  2           2017-03-20 17:17:35.725224 -                          -
  3           2017-03-21 11:48:33.803288 2017-03-21 11:50:57.181136 6436
  4           2017-03-21 11:50:59.656693 2017-03-21 11:51:05.934895 450
  5           2017-03-21 11:51:08.431871 -                          74872
  6           2017-03-21 12:17:30.568260 2017-03-21 12:38:29.070000 60384
  7           2017-03-21 12:38:31.617099 2017-03-21 12:39:53.174712 3966
Viewer> clear_filters
Set filters to {}
Viewer> display_session_details_by_agent 7
  Receiving Agent              control    listener  messageviewer.connection ␣
↪platform.driver  platform.messagedebugger       pubsub    testagent
  (No Receiving Agent)               -           -                        2       ␣
↪         -                        -           -            -
  control                            -           -                        -       ␣
↪         -                        -           2            -
  listener                           -           -                        -       ␣
↪         -                        -         679            -
  messageviewer.connection           -           -                        -       ␣
↪         -                        3           -            -
  platform.driver                    -           -                        -       ␣
↪         -                        -        1249           16
  platform.messagedebugger           -           -                        3       ␣
↪         -                        -           -            -
  pubsub                             2         679                        -       ␣
↪      1249                        -           4           31
  testagent                          -           -                        -       ␣
↪        16                        -          31            -
```

The `display_session_details_by_topic <session_id>` command is similar to `display_session_details_by_agent`, but each row contains statistics for a topic instead of for a receiving agent:

```
Viewer> display_session_details_by_topic 7
  Topic                                    control     listener  messageviewer.
↪connection  platform.driver  platform.messagedebugger       pubsub    testagent
  (No Topic)                                     1          664                ␣
↪  5              640                          3        1314           39
  devices/chargepoint1/Address                   -             -               ␣
↪  -                6                          -           6            -
  devices/chargepoint1/City                      -             -               ␣
↪  -                6                          -           6            -
  devices/chargepoint1/Connector                 -             -               ␣
↪  -                5                          -           5            -
  devices/chargepoint1/Country                   -             -               ␣
↪  -                5                          -           5            -
  devices/chargepoint1/Current                   -             -               ␣
↪  -                6                          -           6            -
  devices/chargepoint1/Description               -             -               ␣
↪  -                6                          -           6            -
  devices/chargepoint1/Energy                    -             -               ␣
↪  -                5                          -           5            -
  devices/chargepoint1/Lat                       -             -               ␣
↪  -                6                          -           6            -
  devices/chargepoint1/Level                     -             -               ␣
↪  -                5                          -           5            -
  devices/chargepoint1/Long                      -             -               ␣
↪  -                6                          -           6            -
  devices/chargepoint1/Mode                      -             -               ␣
↪  -                5                          -           5            -
  devices/chargepoint1/Power                     -             -               ␣
↪  -                6                          -           6            -
  devices/chargepoint1/Reservable                -             -               ␣
↪  -                5                          -           5            -
  devices/chargepoint1/State                     -             -               ␣
↪  -                6                          -           6            -
  devices/chargepoint1/Status                    -             -               ␣
↪  -                5                          -           5            -
  devices/chargepoint1/Status.TimeSta            -             -               ␣
↪  -                6                          -           6            -
  mp
  devices/chargepoint1/Type                      -             -               ␣
↪  -                6                          -           6            -
  devices/chargepoint1/Voltage                   -             -               ␣
↪  -                5                          -           5            -
  devices/chargepoint1/alarmTime                 -             -               ␣
↪  -                6                          -           6            -
  devices/chargepoint1/alarmType                 -             -               ␣
↪  -                6                          -           6            -
  devices/chargepoint1/all                       -             -               ␣
↪  -                5                          -           5            -
  devices/chargepoint1/allowedLoad               -             -               ␣
↪  -                6                          -           6            -
  devices/chargepoint1/clearAlarms               -             -               ␣
↪  -                6                          -           6            -
  devices/chargepoint1/currencyCode              -             -               ␣
↪  -                6                          -           6            -
  devices/chargepoint1/driverAccountN            -             -               ␣
↪  -                5                          -           5            -
  umber
```

(continues on next page)

(continued from previous page)

```
devices/chargepoint1/driverName                  -             -                    ␣
↪  -                5                        -            5             -
devices/chargepoint1/endTime                     -             -                    ␣
↪  -                5                        -            5             -
devices/chargepoint1/mainPhone                   -             -                    ␣
↪  -                6                        -            6             -
devices/chargepoint1/maxPrice                    -             -                    ␣
↪  -                5                        -            5             -
devices/chargepoint1/minPrice                    -             -                    ␣
↪  -                5                        -            5             -
devices/chargepoint1/numPorts                    -             -                    ␣
↪  -                6                        -            6             -
devices/chargepoint1/orgID                       -             -                    ␣
↪  -                5                        -            5             -
devices/chargepoint1/organizationNa              -             -                    ␣
↪  -                5                        -            5             -
me
devices/chargepoint1/percentShed                 -             -                    ␣
↪  -                6                        -            6             -
devices/chargepoint1/portLoad                    -             -                    ␣
↪  -                6                        -            6             -
devices/chargepoint1/portNumber                  -             -                    ␣
↪  -                6                        -            6             -
devices/chargepoint1/sessionID                   -             -                    ␣
↪  -                5                        -            5             -
devices/chargepoint1/sessionTime                 -             -                    ␣
↪  -                6                        -            6             -
devices/chargepoint1/sgID                        -             -                    ␣
↪  -                6                        -            6             -
devices/chargepoint1/sgName                      -             -                    ␣
↪  -                6                        -            6             -
devices/chargepoint1/shedState                   -             -                    ␣
↪  -                5                        -            5             -
devices/chargepoint1/startTime                   -             -                    ␣
↪  -                6                        -            6             -
devices/chargepoint1/stationID                   -             -                    ␣
↪  -                5                        -            5             -
devices/chargepoint1/stationMacAddr              -             -                    ␣
↪  -                6                        -            6             -
devices/chargepoint1/stationManufac              -             -                    ␣
↪  -                5                        -            5             -
turer
devices/chargepoint1/stationModel                -             -                    ␣
↪  -                6                        -            6             -
devices/chargepoint1/stationName                 -             -                    ␣
↪  -                5                        -            5             -
devices/chargepoint1/stationRightsP              -             -                    ␣
↪  -                6                        -            6             -
rofile
devices/chargepoint1/stationSerialN              -             -                    ␣
↪  -                6                        -            6             -
um
heartbeat/control                                1             -                    ␣
↪  -                -                        -            1             -
heartbeat/listener                               -             15                   ␣
↪  -                -                        -            15            -
heartbeat/platform.driver                        -             -                    ␣
↪  -                1                        -            1
```

(continues on next page)

(continued from previous page)

```
  heartbeat/pubsub                                          -            -                              ␣
↪  -                -                         -            2            -
  test_topic/test_subtopic                                  -            -                              ␣
↪  -                -                         -            8            8
```

Database Administration

The Message Debugger Agent stores message data in a SQLite database's DebugMessage, DebugMessageExchange and DebugSession tables. If the database isn't present already when the Agent is started, it is created automatically.

The SQLite database can consume a lot of disk space in a relatively short time, so the Message Viewer has command-line options that recover that space by deleting the database or by deleting all messages belonging to a given debug session.

The `delete_session <session_id>` command deletes the database's DebugSession row with the indicated ID, and also deletes all DebugMessage and DebugMessageExchange rows with that session ID. In the following example, `delete_session` deletes the 60,000 DebugMessages that were captured during a 20-minute period as session 6:

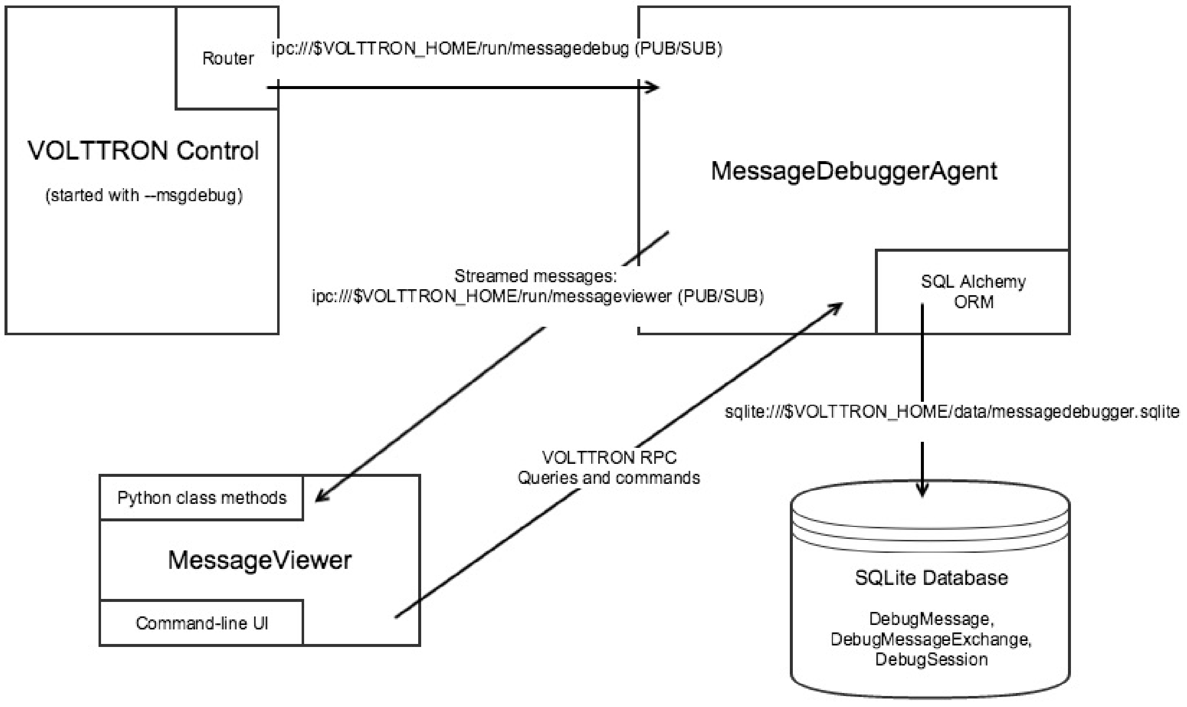
```
Viewer> list_sessions
  rowid        start_time                  end_time                    num_messages
  1            2017-03-20 17:07:13.867951  -                           -
  2            2017-03-20 17:17:35.725224  -                           -
  3            2017-03-21 11:48:33.803288  2017-03-21 11:50:57.181136  6436
  4            2017-03-21 11:50:59.656693  2017-03-21 11:51:05.934895  450
  5            2017-03-21 11:51:08.431871  -                           74872
  6            2017-03-21 12:17:30.568260  2017-03-21 12:38:29.070000  60384
  7            2017-03-21 12:38:31.617099  2017-03-21 12:39:53.174712  3966
  8            2017-03-21 12:42:08.482936  -                           3427
Viewer> delete_session 6
Deleted debug session 6
Viewer> list_sessions
  rowid        start_time                  end_time                    num_messages
  1            2017-03-20 17:07:13.867951  -                           -
  2            2017-03-20 17:17:35.725224  -                           -
  3            2017-03-21 11:48:33.803288  2017-03-21 11:50:57.181136  6436
  4            2017-03-21 11:50:59.656693  2017-03-21 11:51:05.934895  450
  5            2017-03-21 11:51:08.431871  -                           74872
  7            2017-03-21 12:38:31.617099  2017-03-21 12:39:53.174712  3966
  8            2017-03-21 12:42:08.482936  -                           4370
```

The `delete_database` command deletes the entire SQLite database, removing all records of previously-captured DebugMessages, DebugMessageExchanges and DebugSessions. The database will be re-created the next time a debug session is started.

```
Viewer> delete_database
Database deleted
Viewer> list_sessions
No query results
Viewer> start_session
Message debugger session 1 started
Viewer> list_sessions
  rowid        start_time                  end_time     num_messages
  1            2017-03-22 12:39:40.320252  -            180
```

It's recommended that the database be deleted if changes are made to the DebugMessage, DebugMessageExchange or DebugSession object structures that are defined in agent.py. A skew between these data structures in Python code vs. the ones in the database can cause instability in the Message Debugger Agent, perhaps causing it to fail. If a failure of this kind prevents use of the Message Viewer's `delete_database` command, the database can be deleted directly from the filesystem. By default, it is located in $VOLTTRON_HOME's `run` directory.

Implementation Details


Router
ipc:///$VOLTTRON_HOME/run/messagedebug (PUB/SUB)
VOLTTRON Control
(started with --msgdebug)
MessageDebuggerAgent
Streamed messages:
ipc:///$VOLTTRON_HOME/run/messageviewer (PUB/SUB)
SQL Alchemy ORM
sqlite:///$VOLTTRON_HOME/data/messagedebugger.sqlite
VOLTTRON RPC
Queries and commands
Python class methods
MessageViewer
Command-line UI
SQLite Database
DebugMessage, DebugMessageExchange, DebugSession

Router changes: MessageDebuggerAgent reads and stores all messages that pass through the VIP router. This is accomplished by subscribing to the messages on a new socket published by the platform's `Router.issue()` method.

The ''direction'' property: Most agent interactions result in at least two messages, an incoming request and an outgoing response. `Router.issue()` has a `topic` parameter with values INCOMING, OUTGOING, ERROR and UNROUTABLE. The publication on the socket that happens in issue() includes this "issue topic" (not to be confused with a message's `topic`) along with each message. MessageDebuggerAgent records it as a DebugMessage property called `direction`, since its value for almost all messages is either INCOMING or OUTGOING.

SQLite Database and SQL Alchemy: MessageDebuggerAgent records each messsage as a DebugMessage row in a relational database. SQLite is used since it's packaged with Python and is already being used by other VOLTTRON agents. Database semantics are kept simple through the use of a SQL Alchemy object-relational mapping framework. Python's "SQLAlchemy" plug-in must be loaded in order for MessageDebuggerAgent to run.

Calling MessageViewer Directly: The viewer.py module that starts the Message Viewer command line also contains a MessageViewer class. It exposes class methods which can be used to make direct Python calls that, in turn, make Message Debugger Agent's RPC calls. The MessageViewer class-method API includes the following calls:

- delete_debugging_db()
- delete_debugging_session(session_id)
- disable_message_debugging()

- display_db_objects(db_object_name, filters=None)
- display_message_stream()
- enable_message_debugging()
- message_exchange_details(message_id)
- session_details_by_agent(session_id)
- session_details_by_topic(session_id)
- set_filters(filters)
- set_verbosity(verbosity_level)
- start_streaming(filters=None)
- stop_streaming()

The command-line UI's `display_messages` and `display_exchanges` commands are implemented here as `display_db_objects('DebugMessage')` and `display_db_objects(DebugMessageExchange)`. These calls return json-encoded representations of DebugMessages and DebugMessageExchanges, which are formatted for display by MessageViewerCmd.

MessageViewer connection: MessageViewer is not actually a VOLTTRON agent. In order for it make MessageDebuggerAgent RPC calls, which are agent-agent interactions, it builds a "connection" that manages a temporary agent. This is a standard VOLTTRON pattern that is also used, for instance, by Volttron Central.

View the *message debugging specification* for more information on the message debugging implementation for ZeroMQ.

Message Bus Debugging Specification

NOTE: This is a planning document, created prior to implementation of the VOLTTRON Message Debugger. It describes the tool's general goals, but it's not always accurate about specifics of the ultimate implementation. For a description of Message Debugging as implemented, with advice on how to configure and use it, please see *Message-Debugging*.

Description

VOLTTRON agents send messages to each other on the VOLTTRON message bus. It can be useful to examine the contents of this message stream while debugging and troubleshooting agents and drivers.

In satisfaction of this specification, a new Message Monitor capability will be implemented allowing VOLTTRON agent/driver developers to monitor the message stream, filter it for an interesting set of messages, and display the contents and characteristics of each message.

Some elements below are central to this effort (required), while others are useful improvements (optional) that may be implemented if time permits.

Feature: Capture Messages and Display a Message Summary

When enabled, the Message Monitor will capture details about a stream of routed messages. On demand, it will display a message summary, either in real time as the messages are routed, or retrospectively.

A summary view will convey the high level interactions occurring between VOLTTRON agents as conversations that may be expanded for more detail. A simple RPC call that involves 4 message send/recv segments will be displayed as a single object that can be expanded. In this way, the message viewer will provide a higher-level view of message bus activity than might be gleaned from verbose logs using grep.

Pub/sub interactions will be summarized at the topic level with high-level statistics such as the number of subscribers, # of messages published during the capture period, etc. Drilling into the interaction might show the last message published with the ability to drill deeper into individual messages. A diff display would show how the published data is changing.

Summary view

```
- 11:09:31.0831   RPC       set_point             charge.control  platform.driver
| -  params: ('set_load', 10)   return: True
- 11:09:31.5235   Pub/Sub   devices/my_device     platform.driver     2 subscribers
| - Subscriber: charge.control
     | - Last message 11:09:31.1104:
          [
                {
                    'Heartbeat': True,
                    'PowerState': 0,
                    'temperature': 50.0,
                    'ValveState': 0
                },
                ...
            ]
     | - Diff to 11:09:21.5431:
                'temperature': 48.7,
```

The summary's contents and format will vary by message subsystem.

RPC request/response pairs will be displayed on a single line:

```
(volttron) d1:volttron myname$ msmon --agent='(Agent1,Agent2)'

Agent1                                                                      Agent2
2016-11-22T11:09:31.083121+00:00 rpc: devices/my_topic; 2340972387; sent    2016-11-
↪22T11:09:31.277933+00:00 responded: 0.194 sec
2016-11-22T11:09:32.005938+00:00 rpc: devices/my_topic; 2340972388; sent    2016-11-
↪22T11:09:32.282193+00:00 responded: 0.277 sec
2016-11-22T11:09:33.081873+00:00 rpc: devices/my_topic; 2340972389; sent    2016-11-
↪22T11:09:33.271199+00:00 responded: 0.190 sec
2016-11-22T11:09:34.049139+00:00 rpc: devices/my_topic; 2340972390; sent    2016-11-
↪22T11:09:34.285393+00:00 responded: 0.236 sec
2016-11-22T11:09:35.053183+00:00 rpc: devices/my_topic; 2340972391; sent    2016-11-
↪22T11:09:35.279317+00:00 responded: 0.226 sec
2016-11-22T11:09:36.133948+00:00 rpc: devices/my_topic; 2340972392; sent    2016-11-
↪22T11:09:36.133003+00:00 dequeued
```

When PubSub messages are displayed, each message's summary will include its count of subscribers:

```
(volttron) d1:volttron myname$ msmon --agent=(Agent1)

Agent1
2016-11-22T11:09:31.083121+00:00 pubsub: devices/my_topic; 2340972487; sent; 2 subs
2016-11-22T11:09:32.005938+00:00 pubsub: devices/my_topic; 2340972488; sent; 2 subs
2016-11-22T11:09:33.081873+00:00 pubsub: devices/my_topic; 2340972489; sent; 2 subs
2016-11-22T11:09:34.049139+00:00 pubsub: devices/my_topic; 2340972490; sent; 2 subs
2016-11-22T11:09:35.053183+00:00 pubsub: devices/my_topic; 2340972491; sent; 2 subs
```

While streaming output of a message summary, a defined keystroke sequence will "pause" the output, and another keystroke sequence will "resume" displaying the stream.

Feature: Capture and Display Message Details

The Message Monitor will capture a variety of details about each message, including:

1. Sending agent ID
2. Receiving agent ID
3. User ID
4. Message ID
5. Subsystem
6. Topic
7. Message data
8. Message lifecycle timestamps, in UTC (when sent, dequeued, responded)
9. Message status (sent, responded, error, timeout)
10. Message size
11. Other message properties TBD (e.g., queue depth?)

On demand, it will display these details for a single message ID:

```
(volttron)d1:volttron myname$ msmon --id='2340972390'

2016-11-22T11:09:31.053183+00:00 (Agent1)
INFO:
    Subsystem: 'pubsub',
    Sender: 'Agent1',
    Topic: 'devices/my_topic',
    ID: '2340972390',
    Sent: '2016-11-22T11:09:31.004986+00:00',
    Message:
    [
        {
            'Heartbeat': True,
            'PowerState': 0,
            'temperature': 50.0,
            'ValveState': 0
        },
        {
            'Heartbeat':
            {
                'units': 'On/Off',
                'type': 'integer'
            },
            'PowerState':
            {
                'units': '1/0',
                'type': 'integer'
            },
            'temperature':
            {
                'units': 'Fahrenheit',
```

(continues on next page)

(continued from previous page)

```
                'type': 'integer'
            },
            'ValveState':
            {
                'units': '1/0',
                'type': 'integer'
            }
        }
    ]
```

A VOLTTRON message ID is not unique to a single message. A group of messages in a "conversation" may share a common ID, for instance during RPC request/response exchanges. When detailed display of all messages for a single message ID is requested, they will be displayed in chronological order.

Feature: Display Message Statistics

Statistics about the message stream will also be available on demand:

1. Number of messages sent, by agent, subsystem, topic
2. Number of messages received, by agent, subsystem, topic

Feature: Filter the Message Stream

The Message Monitor will be able to filter the message stream display to show only those messages that match a given set of criteria:

1. Sending agent ID(s)
2. Receiving agent ID(s)
3. User ID(s)
4. Subsystem(s)
5. Topic - Specific topic(s)
6. Topic - Prefix(es)
7. Specific data value(s)
8. Sampling start/stop time
9. Other filters TBD

User Interface: Linux Command Line

A Linux command-line interface will enable the following user actions:

1. Enable message tracing
2. Disable message tracing
3. Define message filters
4. Define verbosity of displayed-message output
5. Display message stream

6. Begin recording messages
7. Stop recording messages
8. Display recorded messages
9. Play back (re-send) recorded messages

Feature (not implemented): Watch Most Recent

Optionally, the Message Monitor can be asked to "watch" a specific data element. In that case, it will display the value of that element in the most recent message matching the filters currently in effect. As the data to be displayed changes, the display will be updated in place without scrolling (similar to "top" output):

```
(volttron) d1:volttron myname$ msmon --agent='(Agent1)' --watch='temperature'

Agent1
2016-11-22T11:09:31.053183+00:00 pubsub: my_topic; 2340972487; sent; 2 subs;␣
↪temperature=50
```

Feature (not implemented): Regular Expression Support

It could help for the Message Monitor's filtering logic to support regular expressions. Regex support has also been requested (Issue #207) when identifying a subscribed pub/sub topic during VOLTTRON message routing.

Optionally, regex support will be implemented in Message Monitor filtering criteria, and also (configurably) during VOLTTRON topic matching.

Feature (not implemented): Message Stream Record and Playback

The Message Monitor will be able to "record" and "play back" a message sequence:

1. Capture a set of messages as a single "recording"
2. Inspect the contents of the "recording"
3. "Play back" the recording – re-send the recording's messsage sequence in VOLTTRON

Feature (not implemented): On-the-fly Message Inspection and Modification

VOLTTRON message inspection and modification, on-the-fly, may be supported from the command line. The syntax and implementation would be similar to pdb (Python Debugger), and might be written as an extension to pdb.

Capabilities:

1. Drill-down inspection of message contents.
2. Set a breakpoint based on message properties, halting upon routing a matching message.
3. While halted on a breakpoint, alter a message's contents.

Feature (not implemented): PyCharm Debugging Plugin

VOLTTRON message debugging may also be published as a PyCharm plugin. The plugin would form a more user-friendly interface for the same set of capabilities described above – on-the-fly message inspection and modification, with the ability to set a breakpoint based on message properties.

User Interface (not implemented): PCAP/Wireshark

Optionally, we may elect to render the message trace as a stream of PCAP data, thereby exploiting Wireshark's filtering and display capabilities. This would be in accord with the enhancement suggested in VOLTTRON Issue #260.

User Interface (not implemented): Volttron Central Dashboard Widget

Optionally, the Message Monitor will be integrated as a new Volttron Central dashboard widget, supporting each of the following:

1. Enable/Disable the monitor
2. Filter messages
3. Configure message display details
4. Record/playback messages

User Interface (not implemented): Graphical Display of Message Sequence

Optionally, the Volttron Central dashboard widget will provide graphical display of message sequences, allowing enhanced visualization of request/response patterns.

Related Development: PyCharm Documentation

Also included in this effort will be a contribution to VOLTTRON documentation about installing and configuring a PyCharm environment for developing, debugging and testing VOLTTRON agents and drivers.

Engineering Design Notes

Grabbing Messages Off the Bus

This tool depends on reading and storing all messages that pass through the VIP router. The Router class already has hooks that allow for the capturing of messages at various points in the routing workflow. The BaseRouter abstract class defines `issue(self, topic, frames, extra)`. This method is called from `BaseRouter.route` and `BaseRouter._send` during the routing of messasges. The `topic` parameter (not to be confused with a message topic found in `frames`) identifies the point or state in the routing worflow at which the issue was called.

The defined `topics` are: INCOMING, OUTGOING, ERROR and UNROUTABLE. Most messages will result in two calls, one with the INCOMING topic as the message enters the router and one with the OUTGOING topic as the message is sent on to its destination. Messages without a recipient are intended for the router itself and do not result in an OUTGOING call to `issue`.

`Router.issue` contains the concrete implementation of the method. It does two things:

1. It writes the topic, frames and optional extra parameters to the logger using the FramesFormatter.

2. It invokes `self._tracker.hit(topic, frames, extra)`. The Tracker class collects statistics by topic and counts the messages within a topic by peer, user and subsystem.

The issue method can be modified to optionally publish the `issue` messages to an in-process ZMQ address that the message-viewing tool will subscribe to. This will minimize changes to core VOLTTRON code and minimize the impact of processing these messages for debugging.

Message Processor

The message processor will subscribe to messages coming out of the Router.issue() method and process these messages based on the current message viewer configuration. Messages will be written to a SQLite db since this is packaged with Python and currently used by other VOLTTRON agents.

Message Viewer

The message viewer will display messages from the SQLite db. We need to consider whether it should also subscribe to receiving messages in real-time. The viewer will be responsible for displaying message statistics and will provide a command line interface to filter and display messages.

Message Db Schema

```
message(id, created_on, issue_topic, extras, sender, recipient, user_id, msg_id,␣
↪subsystem, data)
```

msg_id will be used to associate pairs of incoming/outgoing messages.

Note: data will be a jsonified list of frames, alternatively we could add a message_data table with one row per frame.

A session table will track the start and end of a debug session and, at the end of a session, record statistics on the messages in the session.

```
session(id, created_on, name, start_time,  end_time, num_messages)
```

The command line tool will allow users to delete old sessions and select a session for review/playback.

1.22.5 System Monitoring Agent

The System Monitoring Agent (colloquially “SysMon”) can be installed on the platform to monitor various system resource metrics, including percent CPU utilization, percent system memory (RAM) utilization, and percent storage (disk) utilization based on disk path.

Configuration

The SysMon agent configuration includes options for setting the base publish topic as well as intervals in seconds for checking the various system resource utilization levels.

```
{
    "base_topic": "datalogger/log/platform",
    "cpu_check_interval": 5,
    "memory_check_interval": 5,
    "disk_check_interval": 5,
    "disk_path": "/"
}
```

The base topic will be formatted with the name of the function call used to determine the utilization percentage for the resource. For example, using the configuration above, the topic for cpu utilization would be "datalogger/log/platform/cpu_percent".

The disk path string can be set to specify the full path to a specific system data storage "disk". Currently the SysMon agent supports configuration for only a single disk at a time.

Periodic Publish

At the interval specified by the configuration option for each resource, the agent will automatically query the system for the resource utilization statistics and publish it to the message bus using the topic as previously described. The message content for each publish will contain only a single numeric value for that specific topic. Currently "scrape_all" style publishes are not supported.

Example Publishes:

```
2020-03-10 11:20:33,755 (listeneragent-3.3 7993) listener.agent INFO: Peer: pubsub,␣
↪Sender: platform.sysmon:, Bus: , Topic: datalogger/log/platform/cpu_percent,␣
↪Headers: {'min_compatible_version': '3.0', 'max_compatible_version': ''}, Message:
4.8
2020-03-10 11:20:33,804 (listeneragent-3.3 7993) listener.agent INFO: Peer: pubsub,␣
↪Sender: platform.sysmon:, Bus: , Topic: datalogger/log/platform/memory_percent,␣
↪Headers: {'min_compatible_version': '3.0', 'max_compatible_version': ''}, Message:
35.6
2020-03-10 11:20:33,809 (listeneragent-3.3 7993) listener.agent INFO: Peer: pubsub,␣
↪Sender: platform.sysmon:, Bus: , Topic: datalogger/log/platform/disk_percent,␣
↪Headers: {'min_compatible_version': '3.0', 'max_compatible_version': ''}, Message:
```

JSON RPC Methods

The VIP subsystem developed for the VOLTTRON message bus supports remote procedure calls (RPC), which can be used to more directly fetch data from the SysMon agent. Examples are provided below for each RPC call.

```
# Get Percent CPU Utilization
self.vip.rpc.call(PLATFORM.SYSMON, "cpu_percent).get()

# Get Percent System Memory Utilization
self.vip.rpc.call(PLATFORM.SYSMON, "memory_percent).get()

# Get Percent Storage "disk" Utilization
self.vip.rpc.call(PLATFORM.SYSMON, "disk_percent).get()
```

1.22.6 Threshold Detection Agent

The ThresholdDetectionAgent will publish an alert when a value published to a topic exceeds or falls below a configured value. The agent can be configured to watch topics are associated with a single value or to watch devices' all topics.

Configuration

The Threshold Detection Agent supports the *config store* and can be configured with a file named "config".

The file must be in the following format:

- Topics and points in device publishes may have maximum and minimum thresholds but both are not required
- A device's point entries are configured the same way as standard topic entries

```
{
    "topic": {
        "threshold_max": 10
    },

    "devices/some/device/all": {
        "point0": {
            "threshold_max": 10,
            "threshold_min": 0
        },
        "point1": {
            "threshold_max": 42
        }
    }
}
```

1.22.7 Topic Watcher Agent

The Topic Watcher Agent listens to a set of configured topics and publishes an alert if they are not published within some time limit. In addition to "standard" topics the Topic Watcher Agent supports inspecting device *all* topics. This can be useful when a device contains volatile points that may not be published.

Requirements

The Topic Watcher agent requires the Sqlite 3 package. This package can be installed in an activated environment with:

```
pip install sqlite3
```

Configuration

Topics are organied by groups. Any alerts raised will summarize all missing topics in the group.

Individual topics have two configuration options. For standard topics configuration consists of a key value pair of the topic to its time limit.

The other option is for *all* publishes. The topic key is paired with a dictionary that has two keys, *"seconds"* and *"points"*. *"seconds"* is the topic's time limit and *"points"* is a list of points to watch.

```
{
    "groupname": {
        "devices/fakedriver0/all": 10,

        "devices/fakedriver1/all": {
            "seconds": 10,
            "points": ["temperature", "PowerState"]
        }
    }
}
```

1.23 Historian Framework

Historian Agents are the way by which *device*, *actuator*, *datalogger*, and *analysis* topics are automatically captured and stored in some sort of data store. Historians exist for the following storage options:

- A general *SQL Historian* implemented for MySQL, SQLite, PostgreSQL, and Amazon Redshift
- *MongoDB Historian*
- *Crate Historian*
- *Forward Historian* for sending data to another VOLTTRON instance
- *OpenEIS Historian*
- *MQTT Historian* Forwards data to an MQTT broker
- *InfluxDB Historian*

Other implementations of Historians can be created by following the *Developing Historian Agents* guide.

1.23.1 Base Historian

Historians are all built upon the *BaseHistorian* which provides general functionality the specific implementations are built upon.

This base Historian will cache all received messages to a local database before publishing it to the Historian. This allows recovery from unexpected happenings before the successful writing of data to the Historian.

1.23.2 Configuration

In most cases the default configuration settings are fine for all deployments.

All Historians support the following settings:

```
{
    # Maximum amount of time to wait before retrying a failed publish in seconds.
    # Will try more frequently if new data arrives before this timelime expires.
    # Defaults to 300
    "retry_period": 300.0,

    # Maximum number of records to submit to the historian at a time.
    # Defaults to 1000
    "submit_size_limit": 1000,
```

(continues on next page)

(continued from previous page)

```
    # In the case where a historian needs to catch up after a disconnect
    # the maximum amount of time to spend writing to the database before
    # checking for and caching new data.
    # Defaults to 30
    "max_time_publishing": 30.0,

    # Limit how far back the historian will keep data in days.
    # Partial days supported via floating point numbers.
    # A historian must implement this feature for it to be enforced.
    "history_limit_days": 366,

    # Limit the size of the historian data store in gigabytes.
    # A historian must implement this feature for it to be enforced.
    "storage_limit_gb": 2.5

    # Size limit of the backup cache in Gigabytes.
    # Defaults to no limit.
    "backup_storage_limit_gb": 8.0,

    # Do not actually gather any data. Historian is query only.
    "readonly": false,

    # capture_device_data
    #   Defaults to true. Capture data published on the `devices/` topic.
    "capture_device_data": true,

    # capture_analysis_data
    #   Defaults to true. Capture data published on the `analysis/` topic.
    "capture_analysis_data": true,

    # capture_log_data
    #   Defaults to true. Capture data published on the `datalogger/` topic.
    "capture_log_data": true,

    # capture_record_data
    #   Defaults to true. Capture data published on the `record/` topic.
    "capture_record_data": true,

    # Replace a one topic with another before saving to the database.
    "topic_replace_list": [
    #{"from": "FromString", "to": "ToString"}
    ],

    # For historian developers. Adds benchmarking information to gathered data.
    # Defaults to false and should be left that way.
    "gather_timing_data": false

    # Allow for the custom topics or for limiting topics picked up by a historian␣
↪instance.
    # the key for each entry in custom topics is the data handler.  The topic and␣
↪data must
    # conform to the syntax the handler expects (e.g., the capture_device_data␣
↪handler expects
    # data the driver framework). Handlers that expect specific data format are
    # capture_device_data, capture_log_data, and capture_analysis_data. All other␣
↪handlers will be
```

(continues on next page)

(continued from previous page)

```
    # treated as record data. The list associated with the handler is a list of custom
    # topics to be associated with that handler.
    #
    # To restrict collection to only the custom topics, set the following config␣
↪variables to False
    # capture_device_data
    # capture_analysis_data
    # capture_log_data
    # capture_record_data
    "custom_topics": {
        "capture_device_data": ["devices/campus/building/device/all"],
        "capture_analysis_data": ["analysis/application_data/example"],
        "capture_record_data": ["example"]
    },
    # To restrict the points processed by a historian for a device or set of devices␣
↪(i.e., this configuration
    # parameter only filters data on topics with base 'devices).  If the 'device' is␣
↪in the
    # topic (e.g.,'devices/campus/building/device/all') then only points in the list␣
↪will be passed to the
    # historians capture_data method, and processed by the historian for storage in␣
↪its database (or forwarded to a
    # remote platform (in the case of the ForwardHistorian).  The key in the device_
↪data_filter dictionary can
    # be made more restrictive (e.g., "device/subdevice") to limit unnecessary␣
↪searches through topics that may not
    # contain the point(s) of interest.
    "device_data_filter":
        {
            "device": ["point_name1", "point_name2"]
        }
}
```

1.23.3 Topics

By default the base historian will listen to 4 separate root topics:

- *datalogger/**
- *record/**
- *analysis/**
- *devices/**

Each root topic has a *specific message syntax* that it is expecting for incoming data.

Messages published to *datalogger* will be assumed to be *timepoint* data that is composed of units and specific types with the assumption that they have the ability to be plotted easily.

Messages published to *devices* are data that comes directly from drivers.

Messages published to *analysis* are analysis data published by agents in the form of key value pairs.

Finally, messages that are published to *record* will be handled as string data and can be customized to the user specific situation.

1.23.4 Platform Historian

A platform historian is a *"friendly named"* historian on a VOLTTRON instance. It always has the identity of *platform.historian*. A platform historian is made available to a VOLTTRON Central agent for monitoring of the VOLTTRON instances health and plotting topics from the platform historian. In order for one of the historians to be turned into a platform historian the *identity* keyword must be added to it's configuration with the value of *platform.historian*. The following configuration file shows a SQLite based platform historian configuration:

```
{
    "agentid": "sqlhistorian-sqlite",
    "identity": "platform.historian",
    "connection": {
        "type": "sqlite",
        "params": {
            "database": "~/.volttron/data/platform.historian.sqlite"
        }
    }
}
```

Historian Topic Syntax

Each historian will subscribe to the following message bus topics:

- *datalogger/**
- *anaylsis/**
- *record/**
- *devices/**

For each of these topics there is a different message syntax that must be adhered to in order for the correct interpretation of the data being specified.

record/*

The record topic is the most flexible of all of the topics. This topic allows any serializable message to be published to any topic under the root topic *record/*.

Note: This topic is not recommended to plot, as the structure of the messages are not necessarily numeric

```
# Example messages that can be published

# Dictionary data
{'foo': 'world'}

# Numerical data
52

# Time data (note: not a `datetime` object)
'2015-12-02T11:06:32.252626'
```

devices/*

The *devices* topic is meant to be data structured from a scraping of a Modbus or BACnet device. Currently drivers for both of these protocols write data to the message bus in the proper format. VOLTTRON drivers also publish an aggregation of points in an *all* topic.

Only the 'all' topic messages are read and published to a historian.

Both the all topic and point topic have the same header information, but the message body for each is slightly different. For a complete working example of these messages please see `examples.ExampleSubscriber.subscriber.subscriber_agent`

The format of the header and message for device topics (i.e. messages published to topics with pattern "devices/*/all") follows the following pattern:

```
# Header contains the data associated with the message.
{
    # python code to get this is
    # from datetime import datetime
    # from volttron.platform.messaging import headers as header_mod
    # from volttron.platform.agent import utils
    # now = utils.format_timestamp( datetime.utcnow())
    # {
    #     headers_mod.DATE: now,
    #     headers_mod.TIMESTAMP: now
    # }
    "Date": "2015-11-17 21:24:10.189393+00:00",
    "TimeStamp": "2015-11-17 21:24:10.189393+00:00"
}

# Message Format:

# WITH METADATA
# Messages contains a two element list.  The first element contains a
# dictionary of all points under a specific parent.  While the second
# element contains a dictionary of meta data for each of the specified
# points.  For example devices/pnnl/building/OutsideAirTemperature and
# devices/pnnl/building/MixedAirTemperature ALL message would be created as:
[
    {"OutsideAirTemperature ": 52.5, "MixedAirTemperature ": 58.5},
    {
       "OutsideAirTemperature ": {'units': 'F', 'tz': 'UTC', 'type': 'float'},
       "MixedAirTemperature ": {'units': 'F', 'tz': 'UTC', 'type': 'float'}
    }
]

#WITHOUT METADATA
# Message contains a dictionary of all points under a specific parent
{"OutsideAirTemperature ": 52.5, "MixedAirTemperature ": 58.5}
```

analysis/*

Data sent to *analysis/** topics is result of analysis done by applications. The format of data sent to *analysis/** topics is similar to data sent to *devices/*/all* topics.

datalogger/*

Messages published to *datalogger/** will be assumed to be time point data that is composed of units and specific types with the assumption that they have the ability to be graphed easily.

```
{"MixedAirTemperature": {"Readings": ["2015-12-02T00:00:00",
                                      <mixed_reading],
                         "Units": "F",
                         "tz": "UTC",
                         "data_type": "float"}}
```

If no datetime value is specified as a part of the reading, current time is used. A Message can be published without any header. In the above message *Readings* and *Units* are mandatory.

Crate Historian

Crate is an open source SQL database designed on top of a No-SQL design. It allows automatic data replication and self-healing clusters for high availability, automatic sharding, and fast joins, aggregations and sub-selects.

Find out more about crate from https://crate.io/.

Prerequisites

1. Crate Database

For Arch Linux, Debian, RedHat Enterprise Linux and Ubuntu distributions there is a simple installer to get Crate up and running on your system.

```
sudo bash -c "$(curl -L https://try.crate.io)"
```

This command will download and install all of the requirements for running Crate, create a Crate user and install a Crate service. After the installation the service will be available for viewing at `http://localhost:4200` by default.

Note: There is no authentication support within crate.

2. Crate Driver

There is a Python library for crate that must be installed in the VOLTTRON Python virtual environment in order to access Crate. From an activated environment, in the root of the volttron folder, execute the following command:

```
python bootstrap.py --crate
```

or

```
python bootstrap.py --databases
```

or

```
pip install crate
```

Configuration

Because there is no authorization to access a crate database the configuration for the Crate Historian is very easy.

```
{
    "connection": {
        "type": "crate",
        # Optional table prefix defaults to historian
        "schema": "testing",
        "params": {
            "host": "localhost:4200"
        }
    }
}
```

Finally package, install and start the Crate Historian agent.

See also:

Agent Development Walk-through

Influxdb Historian

InfluxDB is an open source time series database with a fast, scalable engine and high availability. It's often used to build DevOps Monitoring (Infrastructure Monitoring, Application Monitoring, Cloud Monitoring), IoT Monitoring, and real-time analytics solutions.

More information about InfluxDB is available from https://www.influxdata.com/.

Prerequisites

InfluxDB Installation

To install InfluxDB on an Ubuntu or Debian operating system, run the script:

```
services/core/InfluxdbHistorian/scripts/install-influx.sh
```

For installation on other operating systems, see https://docs.influxdata.com/influxdb/v1.4/introduction/installation/.

Authentication in InfluxDB

By default, the InfluxDB *Authentication* option is disabled, and no user authentication is required to access any InfluxDB database. You can enable authentication by updating the InfluxDB configuration file. For detailed information on enabling authentication, see: https://docs.influxdata.com/influxdb/v1.4/query_language/authentication_and_authorization/.

If *Authentication* is enabled, authorization privileges are enforced. There must be at least one defined admin user with access to administrative queries as outlined in the linked document above. Additionally, you must pre-create the `user` and `database` that are specified in the configuration file (the default configuration file for InfluxDB is *services/core/InfluxdbHistorian/config*). If your `user` is a non-admin user, they must be granted a full set of privileges on the desired `database`.

InfluxDB Driver

In order to connect to an InfluxDb client, the Python library for InfluxDB must be installed in VOLTTRON's virtual environment. From the command line, after enabling the virtual environment, install the InfluxDB library as follows:

```
python bootstrap.py --influxdb
```

or

```
python bootstrap.py --databases
```

or

```
pip install influxdb
```

Configuration

The default configuration file for VOLTTRON's InfluxDB Historian agent should be in the format:

```
{
  "connection": {
    "params": {
      "host": "localhost",
      "port": 8086,          # Don't change this unless default bind port
                             # in influxdb config is changed
      "database": "historian",
      "user": "historian",   # user is optional if authentication is turned off
      "passwd": "historian" # passwd is optional if authentication is turned off
    }
  },
  "aggregations": {
    "use_calendar_time_periods": true
  }
}
```

The InfluxDB Historian agent can be packaged, installed and started according to the standard VOLTTRON agent creation procedure. A sample VOLTTRON configuration file has been provided: *services/core/InfluxdbHistorian/config*.

See also:

Agent Development Walk-through

Connection

The `host`, `database`, `user` and `passwd` values in the VOLTTRON configuration file can be modified. `user` and `passwd` are optional if InfluxDB *Authentication* is disabled.

Note: Be sure to initialize or pre-create the `database` and `user` defined in the configuration file, and if `user` is a non-admin user, be make sure to grant privileges for the user on the specified `database`. For more information, see *Authentication in InfluxDB*.

Aggregations

In order to use aggregations, the VOLTTRON configuration file must also specify a value, either `true` or `false`, for `use_calendar_time_periods`, indicating whether the aggregation period should align to calendar time periods. If this value is omitted from the configuration file, aggregations cannot be used.

For more information on historian aggregations, see: *Aggregate Historian Agent Specification*.

Supported Influxdb aggregation functions:

- Aggregations: COUNT(), DISTINCT(), INTEGRAL(), MEAN(), MEDIAN(), MODE(), SPREAD(), STDDEV(), SUM()
- Selectors: FIRST(), LAST(), MAX(), MIN()
- Transformations: CEILING(),CUMULATIVE_SUM(), DERIVATIVE(), DIFFERENCE(), ELAPSED(), NON_NEGATIVE_DERIVATIVE(), NON_NEGATIVE_DIFFERENCE()

More information how to use those functions: https://docs.influxdata.com/influxdb/v1.4/query_language/functions/

Note: Historian aggregations in InfluxDB are different from aggregations employed by other historian agents in VOLTTRON. InfluxDB doesn't have a separate agent for aggregations. Instead, aggregation is supported through the `query_historian` function. Other agents can execute an aggregation query directly in InfluxDB by calling the *RPC.export* method `query`. For an example, see *Aggregate Historian Agent Specification*

Database Schema

Each InfluxDB database has a *meta* table as well as other tables for different measurements, e.g. one table for "power_kw", one table for "energy", one table for "voltage", etc. (An InfluxDB *measurement* is similar to a relational table, so for easier understanding, InfluxDB measurements will be referred to below as tables.)

Measurement Table

Example: If a topic name is *CampusA/Building1/Device1/Power_KW*, the *power_kw* table might look as follows:

time	building	campus	device	source	value
2017-12-28T20:41:00.004260096Z	building1	campusa	device1	scrape	123.4
2017-12-30T01:05:00.004435616Z	building1	campusa	device1	scrape	567.8
2018-01-15T18:08:00.126345Z	building1	campusa	device1	scrape	10

`building`, `campus`, `device`, and `source` are InfluxDB *tags*. `value` is an InfluxDB *field*.

Note: The topic is converted to all lowercase before being stored in the table. In other words, a set of *tag* names, as well as a table name, are created by splitting *topic_id* into substrings (see *meta table* below).

In this example, where the typical format of a topic name is *<campus>/<building>/<device>/<measurement>*, *campus*, *building* and *device* are each stored as tags in the database.

A topic name might not confirm to that convention:

1. The topic name might contain additional substrings, e.g. *CampusA/Building1/LAB/Device/OutsideAirTemperature*. In this case, *campus* will be `campusa/building`, *building* will be `lab`, and *device* will be `device`.

2. The topic name might contain fewer substrings, e.g. *LAB/Device/OutsideAirTemperature*. In this case, the *campus* tag will be empty, *building* will be `lab`, and *device* will be `device`.

Meta Table

The meta table will be structured as in the following example:

time	last_updated	meta_dict	topic	topic_id
1970-01-01T00:00:00Z	2017-12-28T20:47:00.003051+00:00	{u'units': u'kw', u'tz': u'US/Pacific', u'type': u'float'}	CampusA/Building1/Device1/Power_KW	campus/building1/device1/power_kw
1970-01-01T00:00:00Z	2017-12-28T20:47:00.003051+00:00	{u'units': u'kwh', u'tz': u'US/Pacific', u'type': u'float'}	CampusA/Building1/Device1/Energy_KWH	campus/building1/device1/energy_kwh

In the InfluxDB, *last_updated*, *meta_dict* and *topic* are *fields* and *topic_id* is a *tag*.

Since InfluxDB is a time series database, the `time` column is required, and a dummy value (`time=0`, which is `1970-01-01T00:00:00Z` based on epoch unix time) is assigned to all topics for easier metadata updating. Hence, if the contents of *meta_dict* change for a specific topic, both *last_updated* and *meta_dict* values for that topic will be replaced in the table.

Mongo Historian

MongoDB is a NoSQL document database, which allows for great performance for transactional data. Because MongoDB documents do not have a schema, it is easy to store and query data which changes over time. MongoDB also scales horizontally using sharding.

For more information about MongoDB, read the MongoDB documentation

Prerequisites

1. Mongodb

Setup mongodb based on using one of the three installation scripts for the corresponding environment:

1. Install as root on Redhat or Cent OS

   ```
   sudo scripts/historian-scripts/root_install_mongo_rhel.sh
   ```

   The above script will prompt user for os version, db user name, password and database name. Once installed you can start and stop the service using the command:

   ```
   **sudo service mongod [start|stop|service]**
   ```

2. Install as root on Ubuntu

   ```
   sudo scripts/historian-scripts/root_install_mongo_ubuntu.sh
   ```

   The above script will prompt user for os version, db user name, password and database name. Once installed you can start and stop the service using the command:

```
**sudo service mongod [start|stop|service]**
```

3. Install as non root user on any Linux machine

```
scripts/historian-scripts/install_mongodb.sh
```

Usage:

```
install_mongodb.sh [-h] [-d download_url] [-i install_dir] [-c config_
↪file] [-s]
```

Optional arguments:

-s setup admin user and test collection after install and startup

-d download url. defaults to https://fastdl.mongodb.org/linux/mongodb-linux-x86_64-3.2.4.tgz

-i install_dir. defaults to current_dir/mongo_install

-c config file to be used for mongodb startup. Defaults to default_mongodb.conf in the same directory as this script. Any data path mentioned in the config file should already exist and should have write access to the current user

-h print the help message

2. Mongodb connector

This historian requires a mongodb connector installed in your activated VOLTTRON virtual environment to talk to MongoDB. Please execute the following from an activated shell in order to install it:

```
python bootstrap.py --mongo
```

or

```
python bootstrap.py --databases
```

or

```
pip install pymongo
```

3. Configuration Options

The historian configuration file can specify

```
"history_limit_days": <n days>
```

which will remove entries from the data and rollup collections older than *n* days. Timestamps passed to the `manage_db_size` method are truncated to the day.

MQTT Historian

Overview

The MQTT Historian agent publishes data to an MQTT broker. The `mqttlistener.py` script will connect to the broker and print all messages.

Dependencies

The Paho MQTT library from Eclipse is needed for the agent and can be installed with:

```
pip install paho-mqtt
```

The Mosquitto MQTT broker may be useful for testing and can be installed with

```
apt-get install mosquitto
```

OpenEIS Historian

An OpenEIS Historian has been developed to integrate real time data ingestion into the OpenEIS platform. In order for the OpenEIS Historian to be able to communicate with an OpenEIS server a datasource must be created on the OpenEIS server.

The process of creating a dataset is documented in the OpenEIS User's Guide under *Creating a Dataset* heading.

Configuration

Once a dataset is created you will be able to add datasets through the configuration file. An example configuration for the historian is as follows:

```
{
    # The agent id is used for display in volttron central.
    "agentid": "openeishistorian",
    # The vip identity to use with this historian.
    # should not be a platform.historian!
    #
    # Default value is un referenced because it listens specifically to the bus.
    #"identity": "openeis.historian",

    # Require connection section for all historians.  The openeis historian
    # requires a url for the openis server and login credentials for publishing
    # to the correct user's dataset.
    "connection": {
        "type": "openeis",
        "params": {
            # The server that is running openeis
            # the rest path for the dataset is dataset/append/{id}
            # and will be populated from the topic_dataset list below.
            "uri": "http://localhost:8000",

            # Openeis requires a username/password combination in order to
            # login to the site via rest or the ui.
            #
            "login": "volttron",
            "password": "volttron"
```

(continues on next page)

(continued from previous page)

```
        }
    },

    # All datasets that are going to be recorded by this historian need to be
    # defined here.
    #
    # A dataset definition consists of the following parts
    #    "ds1": {
    #
    #        The dataset id that was created in openeis.
    #        "dataset_id": 1,
    #
    #        Setting to 1 allows only the caching of data that actually meets
    #        the mapped point criteria for this dataset.
    #        Defaults to 0
    #        "ignore_unmapped_points": 0,
    #
    #        An ordered list of points that are to be posted to openeis. The
    #        points must contain a key specifying the incoming topic with the
    #        value an openeis schema point:
    #        [
    #            {"rtu4/OutsideAirTemp": "campus1/building1/rtu4/OutdoorAirTemperature
↪"}
    #        ]
    #    },
    "dataset_definitions": {
        "ds1": {
            "dataset_id": 1,
            "ignore_unmapped_points": 0,
            "points": [
                {"campus1/building1/OutsideAirTemp": "campus1/building1/
↪OutdoorAirTemperature"},
                {"campus1/building1/HVACStatus": "campus1/building1/HVACStatus"},
                {"campus1/building1/CompressorStatus": "campus1/building1/
↪LightingStatus"}
            ]
        }
#,
#"ds2": {
#    "id": 2,
#    "points": [
#        "rtu4/OutsideAirTemp",
#        "rtu4/MixedAirTemp"
#    ]
#        }
    }
}
```

SQL Historian

An SQL Historian is available as a core service (*services/core/SQLHistorian* in the VOLTTRON repository).

The SQL Historian has been programmed to handle for inconsistent network connectivity (automatic re-connection to tcp based databases). All additions to the historian are batched and wrapped within a transaction with commit and rollback functions. This allows the maximum throughput of data with the most protection.

Configuration

The following example configurations show the different options available for configuring the SQL Historian Agent:

MySQL Specifics

MySQL requires a third party driver (mysql-connector) to be installed in order for it to work. Please execute the following from an activated shell in order to install it.

```
pip install --allow-external mysql-connector-python mysql-connector-python
```

or

```
python bootstrap.py --mysql
```

or

```
python bootstrap.py --databases
```

In addition, the mysql database must be created and permissions granted for select, insert and update before the agent is started. In order to support timestamp with microseconds you need at least MySql 5.6.4. Please see this MySql documentation for more details

The following is a minimal configuration file for using a MySQL based historian. Other options are available and are documented http://dev.mysql.com/doc/connector-python/en/connector-python-connectargs.html. **Not all parameters have been tested, use at your own risk**.

```
{
    "agentid": "sqlhistorian-mysql",
    "connection": {
        "type": "mysql",
        "params": {
            "host": "localhost",
            "port": 3306,
            "database": "volttron",
            "user": "user",
            "passwd": "pass"
        }
    }
}
```

Sqlite3 Specifics

An Sqlite Historian provides a convenient solution for under powered systems. The database is parameter is a location on the file system. By default it is relative to the agents installation directory, however it will respect a rooted or relative path to the database.

```
{
    "agentid": "sqlhistorian-sqlite",
    "connection": {
        "type": "sqlite",
```

(continues on next page)

(continued from previous page)

```
        "params": {
            "database": "data/historian.sqlite"
        }
    }
}
```

PostgreSQL and Redshift

Installation notes

1. The PostgreSQL database driver supports recent PostgreSQL versions. It has been tested on 10.x, but should work with 9.x and 11.x.
2. The user must have SELECT, INSERT, and UPDATE privileges on historian tables.
3. The tables in the database are created as part of the execution of the SQL Historian Agent, but this will fail if the database user does not have CREATE privileges.
4. Care must be exercised when using multiple historians with the same database. This configuration may be used only if there is no overlap in the topics handled by each instance. Otherwise, duplicate topic IDs may be created, producing strange results.
5. Redshift databases do not support unique constraints. Therefore, it is possible that tables may contain some duplicate data. The Redshift driver handles this by using distinct queries. It does not remove duplicates from the tables.

Dependencies

The PostgreSQL and Redshift database drivers require the *psycopg2* Python package.

From an activated shell execute:

```
pip install psycopg2-binary
```

PostgreSQL and Redshift Configuration

The following are minimal configuration files for using a psycopg2-based historian. Other options are available and are documented.

Warning: Not all parameters have been tested, use at your own risk.

Local PostgreSQL Database

The following snippet demonstrates how to configure the SQL Historian Agent to use a PostgreSQL database on the local system that is configured to use Unix domain sockets. The user executing VOLTTRON must have appropriate privileges.

```
{
    "connection": {
        "type": "postgresql",
        "params": {
            "dbname": "volttron"
        }
    }
}
```

Remote PostgreSQL Database

The following snippet demonstrates how to configure the SQL Historian Agent to use a remote PostgreSQL database.

```
{
    "connection": {
        "type": "postgresql",
        "params": {
            "dbname": "volttron",
            "host": "historian.example.com",
            "port": 5432,
            "user": "volttron",
            "password": "secret"
        }
    }
}
```

TimescaleDB Support

Both of the above PostgreSQL connection types can make use of TimescaleDB's high performance Hypertable backend for the primary time-series table. The agent assumes you have completed the TimescaleDB installation and setup the database by following the instructions here.

To use, simply add `timescale_dialect: true` to the connection params in the Agent Config as below:

```
{
    "connection": {
        "type": "postgresql",
        "params": {
            "dbname": "volttron",
            "host": "historian.example.com",
            "port": 5432,
            "user": "volttron",
            "password": "secret",
            "timescale_dialect": true
        }
    }
}
```

Redshift Database

The following snippet demonstrates how to configure the SQL Historian Agent to use a Redshift database.

```
{
    "connection": {
        "type": "redshift",
        "params": {
            "dbname": "volttron",
            "host": "historian.example.com",
            "port": 5432,
            "user": "volttron",
            "password": "secret"
        }
    }
}
```

Data Mover Historian

The Data Mover sends data from its platform to a remote platform in cases where there are not sufficient resources to store data locally. It shares this functionality with the *Forward Historian*, however the Data Mover does not have the goal of data appearing "live" on the remote platform. This allows DataMover to be more efficient by both batching data and by sending an RPC call to a remote historian instead of publishing data on the remote message bus. This allows allows the Data Mover to be more robust by ensuring that the receiving historian is running. If the target is unreachable, the Data Mover will cache data until it is available.

Configuration

The default configuration file is *services/core/DataMover/config*. Change the *destination-vip* value to point towards the foreign Volttron instance.

The following is an example configuration:

```
{
    "destination-vip": "ipc://@/home/volttron/.volttron/run/vip.socket",
    "destination-serverkey": null,
    "required_target_agents": [],
    "custom_topic_list": [],
    "services_topic_list": [
        "devices", "analysis", "record", "datalogger", "actuators"
    ],
    "topic_replace_list": [
        #{"from": "FromString", "to": "ToString"}
    ]
}
```

The *services_topic_list* allows you to specify which of the main data topics to forward. If there is no entry, the historian defaults to sending all.

topic_replace_list allows you to replace portions of topics if needed. This could be used to correct or standardize topics or to replace building/device names with an anonymous version. The receiving platform will only see the replaced values.

Adding the configuration option below will limit the backup cache to *n* gigabytes. This will keep a hard drive from filling up if the agent is disconnected from its target for a long time.

```
"backup_storage_limit_gb": n
```

See also:

Historian Framework

Forward Historian

The primary use case for the Forward Historian is to send data to another instance of VOLTTRON as if the data were live. This allows agents running on a more secure and/or more powerful machine to run analysis on data being collected on a potentially less secure/powerful board.

Given this use case, it is not optimized for batching large amounts of data when "live-ness" is not needed. For this use case, please see the *Data Mover Historian*.

The Forward Historian can be found in the *services/core directory*.

Configuration

The default configuration file is *services/core/ForwardHistorian/config*. Change the *destination-vip* value to point towards the foreign VOLTTRON instance.

```
{
    "agentid": "forwarder",
    "destination-vip": "ipc://@/home/volttron/.volttron/run/vip.socket"
}
```

In order to send to a remote platform, you will need its VIP address and server key. The server key can be found by running:

```
vctl auth serverkey
```

Put the result into the following example:

Note: The example shown uses the local IP address, the IP address for your configuration should match the intended target

```
{
    "agentid": "forwarder",
    "destination-vip": "tcp://127.0.0.1:22916",
    "destination-serverkey": "<SOME_KEY>"
}
```

Adding the configuration option below will limit the backup cache to *n* gigabytes. This will help keep a hard drive from filling up if the agent is disconnected from its target for a long time.

```
"backup_storage_limit_gb": n
```

See also:

Historian Framework

1.24 Web Framework

This document describes the interaction between web enabled agents and the Master Web Service agent.

The web framework enables agent developers to expose JSON, static, and websocket endpoints.

1.24.1 Web SubSystem

Enabling

The web subsystem is not enabled by default as it is only required by a small subset of agents. To enable the web subsystem the platform instance must have an enabled the web server and the agent must pass enable_web=True to the agent constructor.

Methods

The web subsystem allows an agent to register three different types of endpoints; path based, JSON and websocket. A path based endpoint allows the agent to specify a prefix and a static path on the file system to serve static files. The prefix can be a regular expression.

Note: The web subsystem is only available when the constructor contains enable_web=True.

The below examples are within the context of an object that has extended the `volttron.platform.vip.agent.Agent` base class.

Note: For all endpoint methods the first match wins. Therefore ordering which endpoints are registered first becomes important.

```
@Core.receiver('onstart')
def onstart(self, sender, **kwargs):
    """
    Allow serving of static content from /var/www
    """
    self.vip.web.register_path(r'^/vc/.*', '/var/www')
```

JSON endpoints allows an agent to serve data responses to specific queries from a web client.non-static responses. The agent will pass a callback to the subsystem which will be called when the endpoint is triggered.

```
def jsonrpc(env, data):
"""
The main entry point for jsonrpc data
"""
    return {'dyamic': 'data'}

@Core.receiver('onstart')
def onstart(self, sender, **kwargs):
"""
Register the /vc/jsonrpc endpoint for doing json-rpc based methods
"""
    self.vip.web.register_endpoint(r'/vc/jsonrpc', self.jsonrpc)
```

Websocket endpoints allow bi-directional communication between the client and the server. Client connections can be authenticated during the opening of a websocket through the response of an open callback.

```
def _open_authenticate_ws_endpoint(self, fromip, endpoint):
    """
    A client attempted to open an endpoint to the server.

```

(continues on next page)

(continued from previous page)

```
    Return True or False if the endpoint should be allowed.

    :rtype: bool
    """
    return True

def _ws_closed(self, endpoint):
    _log.debug("CLOSED endpoint: {}".format(endpoint))

def _ws_received(self, endpoint, message):
    _log.debug("RECEIVED endpoint: {} message: {}".format(endpoint,
                                                          message))

@Core.receiver('onstart')
def onstart(self, sender, **kwargs):
    self.vip.web.register_websocket(r'/vc/ws', self.open_authenticate_ws_endpoint,␣
↪self._ws_closed, self._ws_received)
```

1.25 Simulation Integration Framework

This framework provides a way to integrate different type of simulation platforms with VOLTTRON. Integration with specific simulation platforms are all built upon the BaseSimIntegration class which provides common APIs needed to interface with different types of simulation platforms. Each of the concrete simulation class extends BaseSimIntegration class and is responsible for interfacing with a particular simulation platform. Using these concrete simulation objects, agents will be able to use the APIs provided by them to participate in a simulation, send inputs to the simulation and receive outputs from the simulation and act on it. Currently, we have implementations for integrating with HELICS, GridAPPSD and EnergyPlus. If one wants to integrate with a new simulation platform, then one has to extend BaseSimIntegration class and provide concrete implementation for each of the APIs provided by BaseSimIntegration class. For details on BaseSimIntegration class, please refer to `volttron/platform/agent/base_simulation_integration/base_sim_integration.py`

1.25.1 Specification For Simplifying Integration With Simulation Platforms

There are several simulation platforms that can be integrated with VOLTTRON to run as a single cohesive simulated environment for different type of applications. Some of the platforms are FNCS, HELICS, GridAPPS-D and EnergyPlus. They all have unique application areas and differ in the type of simulations they run, inputs they accept and outputs they produce. There are some similarities in the some of the basic steps of integrating with VOLTTRON such as

1. Start simulation
2. Subscribe to outputs from the simulation
3. Publish outputs from simulation to VOLTTRON
4. Subscribe to topics from VOLTTRON
5. Send inputs to simulation
6. Advance simulation time step
7. Pause simulation
8. Resume simulation
9. Stop simulation

Currently, VOLTTRON has individual implementations for integrating with many of the above simulation platforms. For example, an example of integrating with GridAPPSD can be found in *examples/GridAPPSD/gridappsd_example/agent.py*. EnergyPlus agent can be found in ‘‘. These implementations will still be available for users. Instead, in this specification we are proposing a base simulation integration class that will provide common APIs and concrete simulation integration classes that will have implementation of the these APIs as per the needs of the individual simulation platforms. Users can use appropriate simulation classes based on which simulation platform they want to integrate with.

Features:

1. **Start simulation** This will start the simulation or register itself to be participant in the simulation.
2. **Register for inputs from simulation** A list of points need to be made available in a config file. The inputs are then read from the config file and registered with simulation platform. Whenever there is any change in those particular points, they are made available to this class to process. The agent using this class object can process it or publish it over VOLTTRON message bus to be consumed by other agents.
3. **Send inputs to simulation** Send inputs such as set points (for example, charge_EV5), data points etc to the simulation. The simulation would then act on these inputs.
3. **Receive outputs from simulation** Receive outputs generated by the simulation (for example, OutdoorAirTemperature for a energyPlus simulation). The agent can then act on these output values. If the config file has an associated topic, the output value will be republished on the VOLTTRON message bus.
4. **Simulation time management** Typically, in a simulation environment, one can run applications in real time mode or in fast execution mode. All the participants in the simulation have to be in sync with respect to time for simulation to be correct. There is typically a central unit which acts as a global timekeeper. This timekeeper can possibly be configured to use periodic time keeping, which means it periodically advances in time (based on pre-configured time period) or based on time advancement message. After each advancement, it would send out all the output messages to the registered participants. Another way of advancing the simulation would be based on concept of time request-time grant. Each of the participants would request for certain time after it is done with its work and get blocked until that is granted. The global time keeper would grant time (and hence advance in simulation) that is lowest among the list of time requests and all participants would advance to that time.
5. **Pause the simulation** Some simulation platforms can pause the simulation if needed. We need provide wrapper API to call simulation specific pause API.
6. **Resume the simulation** Some simulation platforms can resume the simulation if needed. We need provide API to call simulation specific resume API.
7. **Stop the simulation** This will unregister itself from the simulation and stop the simulation.

APIs

1. **start_simulation():**
    - Connect to the simulation platform.
    - Register with the platform as a participant
2. **register_inputs(config=None, callback=None)**
    - Register the config containing inputs and outputs with the simulation platform.
    - If agent provides a callback method, this will be called when new output values is received from simulation
3. **publish_to_simulation(topic, message)**

   - Send message to simulation

4. **make_time_request(time_steps)**
   - Make request to simulation to advance to next time delta
5. **pause_simulation()**
   - Pause simulation
6. **resume_simulation()**
   - Resume simulation
7. **stop_simulation()**
   - Stops the simulation
8. **is_sim_installed()**
   - Flag indicating if simulation is installed

1.25.2 Configuration for Integrating With Simulation Platforms

Configurations for interfacing with simulation platforms will vary depending on the specifications of that platform but there may be few common configuration options that we can group together as separate sections such as

- Config parameters that help us setup the simulation such as connection parameters (connection address), unique name for the participant, total simulation time
- List of topics for subscribing with simulation platform
- List of topics for publishing to the simulation platform
- List of topics subscribing with VOLTTRON message bus

We have grouped these four categories of configuration into four different sections - properties, inputs, outputs and volttron_subscriptions. The simulation integration class will read these four sections and register with simulation platform appropriately. If an agent needs to interface with EnergyPlus or HELICS using the simulation integration framework, then it will need to group the configurations into above four sections.

Note

GridAPPS-D can run complex power system simulations using variety of simulators such as GridLAB-D, HELICS, MatPower etc. So the configuration for GridAPPS-D cannot follow the above format. Because of this, the configuration for GridAPPSD is taken in the raw format and passed drectly to the GridAPPS-D simulation.

Example Configuration

The configuration for interfacing with a simulation platform is described by using integration with HELICS as an example. Each participant in a HELICS co-simulation environment is called a federate.

Below is an example HELICS config file.

```
# Config parameters for setting up HELICS federate
properties:
    name: federate1 # unique name for the federate
    loglevel: 5 # log level
    coreType: zmq # core type
    timeDelta: 1.0 # time delta (defaults to 1s)
```

(continues on next page)

(continued from previous page)

```yaml
    uninterruptible: true
    simulation_length: 360 # simulation length in seconds (defaults to 360s)

# configuration for subscribing to HELICS simulation
outputs:
    # List of subscription information, typically contains
    # - subscription topic,
    # - datatype
    # - publication topic for VOLTTRON (optional) to republish the
    #   message on VOLTTRON message bus
    # - additional/optional simulation specific configuration
    - sim_topic: federate2/totalLoad
      volttron_topic: helics/abc
      type: complex
      required: true
    - sim_topic: federate2/charge_EV6
      volttron_topic: helics/ev6
      type: complex
      required: true

# configuration for publishing to HELICS simulation
inputs:
    # List of publication information, containing
    # - HELICS publication topic,
    # - datatype
    # - metadata associated with the topic (for example unit)
    # - subscription topic for VOLTTRON message bus (optional) which can then be
    #   republished on HELICS with HELICS publication topic
    # - additional/optional publication specific configuration
    - sim_topic: pub1 # HELICS publication key
      type: double    # datatype
      unit: m         # unit
      info: this is an information string for use by the application #additional info
      volttron_topic: pub1/all # topic to subscribe on VOLTTRON bus
      global: true
    - sim_topic: pub2
      type: double
      volttron_topic: pub2/all

volttron_subscriptions:
    - feeder0_output/all
```

The properties section may contain the following.

- Unique name for the federate
- core type (for example, zmq, tcp, mpi)
- time step delta in seconds
- total simulation time etc

Note The individual fields under this section may vary depending on whether the agent is interfacing with HELICS or EnergyPlus.

In the inputs section, list of subscriptions (if any) need to be provided. Each subscription will contain the following.

- subscription topic
- data type

- VOLTTRON topic to republish the message on VOLTTRON message bus (optional)
- required flag (optional)

In the outputs section, list of publications (if any) need to be provided. Each publication will contain the following.

- publication topic
- data type
- metadata associated with the topic
- VOLTTRON topic to subscribe on the VOLTTRON message bus which will be republished on simulation bus (optional)
- additional information (optional)

In the volttron_subscriptions, list of topics need to be subscribed on VOLTTRON bus can be provided.

1.25.3 Integrating With Simulation Platforms

An agent wanting to integrate with a simulation platform has to create an object of concrete simulation integration class (HELICSSimIntegration). This is best described with an example agent. The example agent will interface with HELICS co-simulation platform. For more info about HELICS, please refer to https://helics.readthedocs.io/en/latest/installation/linux.html.

```
class HelicsExample(Agent):
    """
    HelicsExampleAgent demonstrates how VOLTTRON agent can interact with HELICS␣
↪simulation environment
    """
    def __init__(self, config, **kwargs):
        super(HelicsExample, self).__init__(enable_store=False, **kwargs)
            self.config = config
            self.helics_sim = HELICSSimIntegration(config, self.vip.pubsub)
```

Register With Simulation Platform

The agent has to first load the configuration file containing parameters such as connection address, simulation duration, input and output topics etc., and register with simulation platform. The concrete simulation object will then register the agent with simulation platform (in this case, HELICS) using appropriate APIs. The registration steps include connecting to the simulation platform, passing the input and outputs topics to the simulation etc. In addition to that, the agent has to provide a callback method in order for the concrete simulation object to pass the messages received from the simulation to the agent. The best place to call the register_inputs API is within the onstart method of the agent.

```
@Core.receiver("onstart")
def onstart(self, sender, **kwargs):
    """
    Register config parameters with HELICS.
    Start HELICS simulation.
    """
    # Register inputs with HELICS and provide callback method to receive messages␣
↪from simulation
    try:
```

(continues on next page)

(continued from previous page)

```
        self.helics_sim.register_inputs(self.config, self.do_work)
    except ValueError as ex:
        _log.error("Unable to register inputs with HELICS: {}".format(ex))
        self.core.stop()
        return
```

Start the Simulation Platform

After registering with the simulation platform, the agent can go ahead and start the simulation.

```
# Register inputs with HELICS and provide callback method to receive messages from␣
↪simulation
try:
    self.helics_sim.start_simulation()
except ValueError as ex:
    _log.error("Unable to register inputs with HELICS: {}".format(ex))
    self.core.stop()
    return
```

Receive outputs from the simulation

The concrete simulation object spawns a continuous loop that waits for any incoming messages (subscription messages) from the simulation platform. On receiving a message, it passes the message to the callback method registered by the agent during the register with simulation step <Register-Simulation>‘_. The agent can now choose to work on the incoming message based on it’s use case. The agent can also choose to publish some message back to the simulation at this point of time as shown in below example. This is totally optional and is based on agent’s usecase. At the end of the callback method, the agent needs to make time request to the simulation, so that it can advance forward in simulation. Please note, this is a necessary step for HELICS co-simulation integration as the HELICS broker waits for time requests from all it’s federates before advancing the simulation. If no time request is made, the broker blocks the simulation.

```
def do_work(self):
    """
    Perform application specific work here using HELICS messages
    :return:
    """
    current_values = self.helics_sim.current_values
    _log.debug("Doing work: {}".format(self.core.identity))
    _log.debug("Current set of values from HELICS: {}".format(current_values))
    # Do something with HELICS messages
    # agent specific work!!!

    for pub in self.publications:
        key = pub['sim_topic']
        # Check if VOLTTRON topic has been configured. If no, publish dummy value for␣
↪the HELICS
        # publication key
        volttron_topic = pub.get('volttron_topic', None)
        if volttron_topic is None:
            value = 90.5
            global_flag = pub.get('global', False)
            # If global flag is False, prepend federate name to the key
            if not global_flag:
```

(continues on next page)

(continued from previous page)

```
                key = "{fed}/{key}".format(fed=self._federate_name, key=key)
                value = 67.90
            self.helics_sim.publish_to_simulation(key, value)

    self.helics_sim.make_time_request()
```

Publish to the simulation

The agent can publish messages to the simulation using publish_to_simulation API. The code snippet iterates over all the publication keys (topics) and uses publish_to_simulation API to publish a dummy value of 67.90 for every publication key.

```
for pub in self.publications:
    key = pub['sim_topic']
    value = 67.90
    self.helics_sim.publish_to_simulation(key, value)
```

Advance the simulation

With some simulation platforms such as HELICS, the federate can make explicit time request to advance in time by certain number of time steps. There will be a global time keeper (in this case HELICS broker) which will be responsible for maintaining time within the simulation. In the time request mode, each federate has to request for time advancement after it has completed it's work. The global time keeper grants the lowest time among all time requests. All the federates receive the granted time and advance forward in simulation time together in a synchronized manner. Please note, the granted time may not be the same as the requested time by the agent.

Typically, the best place to make the time request is in the callback method provided to the simulation integration object.

```
self.helics_sim.make_time_request()
```

Pause the simulation

Some simulation platforms such as GridAPPS-D have the capability to pause the simulation. The agent can make use of this functionality by calling the appropriate wrapper API exposed by the concrete simulation class. In case of HELICS, we do not have capability of pause/resume simulation, so calling pause_simulation() API will result in no operation.

```
self.helics_sim.pause_simulation()
```

Resume the simulation

If the simulation platform provides the pause simulation functionality then it will also provide capability to resume the simulation. The agent can call resume_simulation API to resume the simulation. In case of HELICS, we do not have the capability of pause/resume simulation, so calling resume_simulation() API will result in no operation.

```
self.helics_sim.resume_simulation()
```

Stop the simulation

The agent can stop the simulation at any point of point. In the case of HELICSSimIntegration object, it will disconnect the federate from the HELICS core and close the library. Generally, it is a good practice to call the stop_simulation API within the onstop() method of the agent. In this way, the agent stops the simulation before exiting the process.

```
@Core.receiver("onstop")
def onstop(self, sender, **kwargs):
    """
    This method is called when the Agent is about to shutdown, but before it
    disconnects from the message bus.
    """
    self.helics_sim.stop_simulation()
```

1.26 Platform Service Standardization

Service will interact with the message bus through three topics.

- Request - The service agent will listen to incoming requests on this topic
- Response - The service agent will respond on this topic
- Error - The service will "throw" errors on this topic

Agents which are using these services agents should publish to the above Request topic and listen on the Reponse and Error topics. Response and Errors will retain the header that was sent into the request.

Headers

- Request Headers
- Common Header Formats
- type - Unique type of request for the service agent to handle (If an agent handles more than one request type on a specific topic)
- requesterID - Name of the requesting agent

1.26.1 Header List

- type - Unique type of request for the service agent to handle (If an agent handles more than one request type on a specific topic)
- priority - HIGH, LOW, LOW_PREEMPT (Found in Scheduler and Activator)
- taskId - Unique task among scheduled tasks.
- window - Seconds remaining in timeslot (actuator agent)
- SourceName - used as name to publish to in smap for archiver agent.
- FROM - Same as requestor id (volttron.messaging.headers.FROM)
- CONTENT_TYPE - volttron.messaging.headers.CONTENT_TYPE.JSON, volttron.messaging.headers.CONTENT_TYPE.PLAIN_TEXT Datalogger location is specified in the message itself.
- Multibuilding
- Cookie

1.26.2 Request Formats (Content-Types)

- volttron.messaging.headers.CONTENT_TYPE.JSON
- volttron.messaging.headers.CONTENT_TYPE.PLAIN_TEXT

1.26.3 Topic List

- Actuator and Scheduling Agent
  - devices/actuators/schedule/request (NEW_SCHEDULE request)
  - devices/actuators/schedule/request (CANCEL_SCHEDULE request)
  - devices/actuators/schedule/response
  - devices/actuators/schedule/announce/[full device path]
  - devices/actuators/schedule/response (Response for preempted task)
  - devices/actuators/get/[full device path]/[actuation point]
  - devices/actuators/set/[full device path]/[actuation point]
  - devices/actuators/value/[full device path]/[actuation point]
  - devices/actuators/error/[full device path]/[actuation point]
- Archiver Agent
  - archiver/request/[path to the value desired/ full device path]
- Logger Agent
  - datalogger/log/
  - datalogger/log/[path in SMAP for the data point]
  - datalogger/status (Status of the storage request)
- Mobility Agent
  - platform/move/request/[agent id]
  - platform/move/reply/[agent id]
- Multi-Building Agent
  - building/recv/[campus]/[building]/[topic]
  - building/send/[campus]/[building]/[topic]
  - building/error/[campus]/[building]/[topic]
- Weather Agent
  - Weather agent topic list
- Platform Topics
  - platform/shutdown
  - agent/[agent]/shutdown

1.27 Acquiring Third Party Agent Code

Third party agents developed from a variety of sources are available from the volttron-applications repository (https://github.com/VOLTTRON/volttron-applications.git). The current best practice is to have the main volttron and the volttron-applications repository within the same common ancestry folder.

```
volttron-repositories/
|
|--- volttron/
|
|--- volttron-applications/
```

One can clone the latest applications from the repository via the following command:

```
git clone https://github.com/VOLTTRON/volttron-applications.git
```

1.28 Driver Framework Overview

VOLTTRON drivers act as an interface between agents on the platform and a device. While running on the platform, drivers are special purpose agents which instead of being run as a separate process, are run as a greenlet in the Master Driver process.

Driver instances are created by the Master Driver when a new driver configuration is added to the configuration store. Drivers use the following topic pattern *devices/<campus>/<building>/<device id>*. When a configuration file is added to the Master Driver's store using this pattern, the Master Driver creates a Driver Agent. The Driver agent is in turn instantiated with a instance of the Interface class corresponding to the *driver_type* parameter in the configuration file. The Interface class is responsible for implementing the communication paradigms of a device or protocol. Once configured, the Master Driver periodically polls the Driver Agent for data which is collected from the interface class. Additionally, points can be requested ad-hoc via the Master Driver's JSON-RPC method "get_point". Points may be set by using JSON-RPC with the Actuator agent to set up a schedule and calling the "set_point" method.

1.28.1 Driver Conventions

- Drivers are polled by the Master Driver agent and values can be set using the *Actuator Agent*
- Drivers should have a 1-to-1 relationship with a device
- Driver modules should be written in Python files in the *services/core/MasterDriverAgent/master_driver/interfaces* directory in the VOLTTRON repository. The master driver will search for a Python file in this directory matching the name provided by the *driver_type* value from the driver configuration when creating the Driver agent.
- Driver code consists of an Interface class (exactly named), supported in most cases by one or more Register classes

1.28.2 Agent-Driver Communication Patterns

The VOLTTRON message bus has been developed to allow agents on the platform to interact with each other, as well as with ICS (Industrial Control Systems) and IOT (Internet of Things) devices via the VOLTTRON driver framework. Agents and drivers have the ability to publish data to the message bus and to subscribe to message bus topics to read in data as it is published. Additionally, agents may implement JSONRPC calls and expose JSONRPC endpoints to communicate more directly with other agents. The following diagram demonstrates typical platform communication patterns for a single platform deployment.

Typical Single Platform Behavior

The diagram features several entities that comprise the platform and its connected components:

- The VOLTTRON message bus - The message bus is the means of transmission of information in VOLTTRON. The VOLTTRON message bus is built around existing message bus software; currently VOLTTRON supports RabbitMQ and ZeroMQ. The VOLTTRON integration includes Pub/Sub and JSON RPC interfaces for agent and driver communication.
- VOLTTRON Platform Agents and Subsystems - These agents and subsystems are installed on the platform to manage the platform. They provide many user facing functions, aid in communication and manage other agents and drivers.
- User's Agents - These agents are either agents included in the core repository but installed by a user, or user built agent modules. They may perform a huge variety of user specified tasks, including data collection, device control, simulation, etc.
- Master Driver Agent - This agent is installed by a user to facilitate communication with drivers. Drivers should not communicated with directly - the master driver implements several features for communicating with drivers to ensure smooth operation and consistent driver behavior.
- Actuator agent - This agent is installed by user to provide scheduling capability for controlling drivers. The master driver does not include protections for race conditions, etc. It is always recommended to use the Actuator agent to set values on a device.
- Device Driver - Drivers are special purpose agents which provide an interface between the master driver and devices such as Modbus, and BACnet devices. Drivers implement a specific set of features for protecting device communication ensuring uniform behaviors across different devices.
- Device - Devices may be low level physical computers for controlling various systems such as PLCs (Programmable Logic Controller), devices which communicate on the local network (such as a Smart T.V.), or devices which are accessed via a remote web API (other smart devices).

Lines of Communication

Connectivity of the platform follows the following paradigm:

- Platform agents (including the Master Driver and Actuator), subsystems, and user agents communicate with the message bus via a publish/subscribe system.
- Agents can communicate "directly" to each other via JSONRPC calls - JSONRPC calls use the VOLTTRON message bus router to "direct" messages to an intended recipient. RPC calls from an agent specify a function for the recipient to perform including input parameters, and the response to the sender should contain the value output by the specified function.
- The Master Driver will periodically poll device drivers. This functionality is intentionally not user-facing. The Master Driver iterates over the configured drivers and calls their respective "scrape_all" methods. This will trigger the drivers to collect point values.
- The Driver will communicate with its configured end devices to collect data points which it then returns to the driver. The driver then publishes the point data to the bus under the *<campus>/<building>/<device id>/all* topic.
- To get an individual device point, the user agent should send an RPC call to the Master Driver for "get_point", providing the point's corresponding topic. After the Master Driver processes the request, communication happens very similarly to polling, but rather than an "all" publish, the data is returned via the Master Driver to the user agent.

- To set a point on a device, it is recommended to use an Actuator Agent. The user agent sends an RPC request to the Actuator to schedule time for the agent to control the device. During that scheduled time the user agent may send it a set point request. If the schedule has been created, the actuator will then forward that request to the Master Driver, at which point the communication happens similarly to a "get_point" request.

The general paradigm for the device-driver relationship as specified by the VOLTTRON driver framework is a 1-to-1 relationship. Each end device should be interacted with via a single device driver configured on one platform. To distribute device data, the DataPuller and forwarder agents can be used at the platform level. Multiple platforms are not intended to collect data or share control of a single device.

Special Case Drivers

Some drivers require a different communication paradigm. One common alternative is shown in the diagram below:

This example describes an alternative pattern wherein BACnet drivers communicate via a BACnet proxy agent to communicate with end devices. This behavior is derived from the networking requirements of the BACnet specification. BACnet specifies a star topology for a given network; "slave" devices in a BACnet network communicate with a single "master". In this case, the BACnet proxy acts as a virtual BACnet master, and device drivers forward their requests to this agent which then performs the BACnet communication (whereas the typical pattern would have devices communicate directly with the corresponding device). There are many other situations which may require this paradigm to be adopted (such as working with remote APIs with request limits), and it is up to the party implementing the driver to determine if this pattern or another pattern may be the most appropriate implementation pattern for their respective use case.

1.28.3 Installing the Fake Driver

The Fake Driver is included as a way to quickly see data published to the message bus in a format that mimics what a true driver would produce. This is a simple implementation of the VOLTTRON driver framework.

See *instructions for installing the fake driver*

To view data being published from the fake driver on the message bus, one can *install the Listener Agent* and read the VOLTTRON log file:

```
cd <root volttron directory>
tail -f volttron.log
```

1.29 Master Driver

The Master Driver agent is a special purpose agent a user can install on the platform to manage communication of the platform with devices. The Master driver features a number of endpoints for collecting data and sending control signals using the message bus and automatically publishes data to the bus on a specified interval.

1.29.1 How does it work?

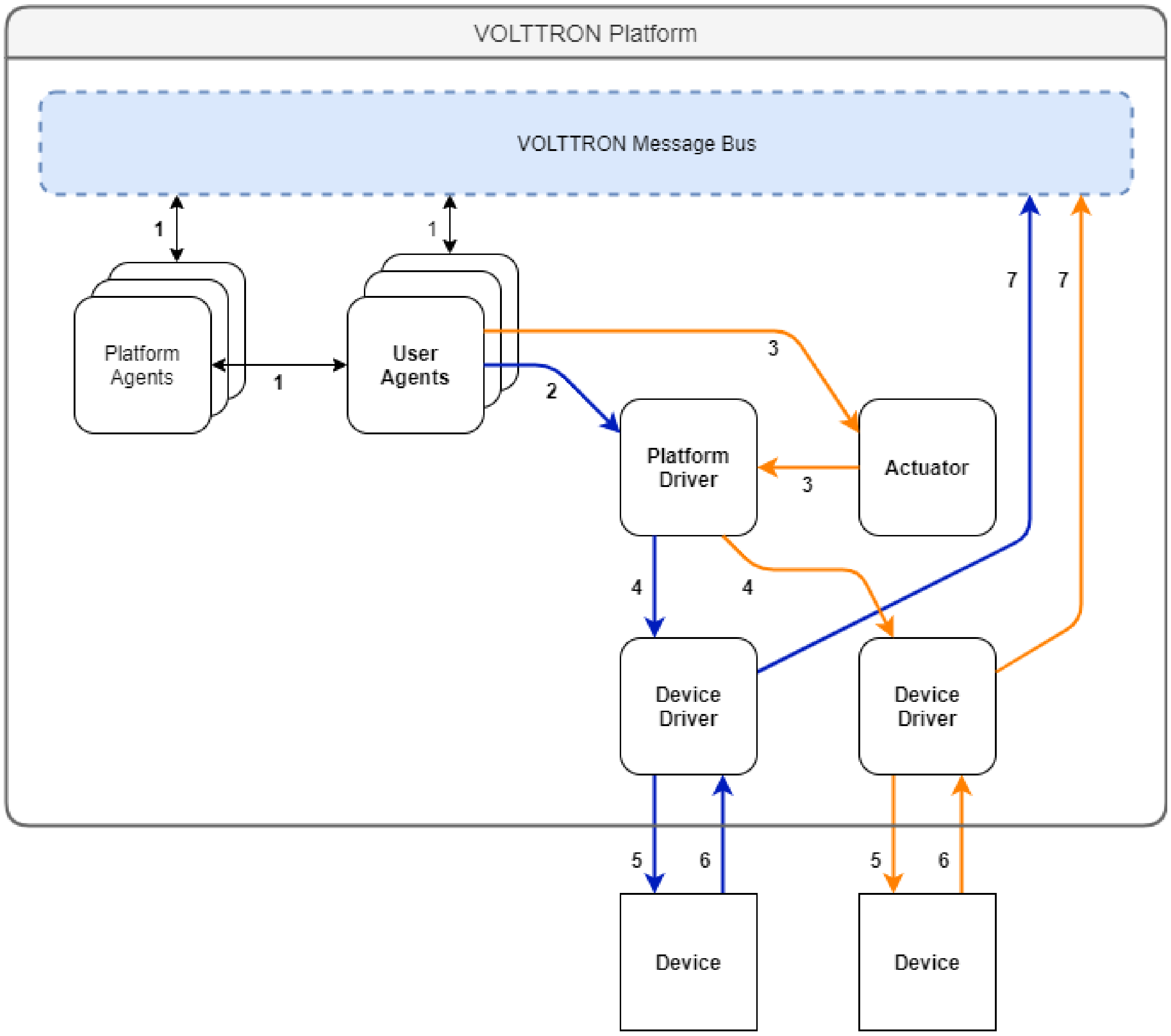
The Master Driver creates a number of driver instances based on the contents of its config store; for each combination of driver configuration, registry configuration and other referenced config files, a driver instance is created by the Master Driver. When configuration files are removed, the corresponding driver instance is removed by the Master Driver.

Drivers are special-purpose agents for device communication, and unlike most agents, run as separate threads under the Master Driver (typically agents are spawned as their own process). While running, the driver periodically "scrapes" device data and publishes the scrape to the message bus, as well as handling ad-hoc data collection and control signalling commands issued from the Master Driver. The actual commands are issued to devices by the driver's "Interface" class.

An Interface class is a Python class which serves as the interface between the driver and the device. The Interface does this by implementing a set of well-defined actions using the communication paradigms and protocols used by the device. For devices such as BACnet and Modbus devices, interfaces wrap certain protocol functions in Python code to be used by the driver. In other cases, interfaces interact with web-API's, etc.

Device/Driver Communication

The below diagram demonstrates driver communication on the platform in a typical case.



Communication occurs using the following steps:

1. Platform agents and the user's agents communicate between themselves and the message bus using publish/subscribe or JSON-RPC
2. The user's agent sends a JSON-RPC request to the Platform Driver to *get_point*

3. And/Or the user's agent sends a JSON-RPC request to the Actuator to *set_point*
4. The Platform Driver forwards the request to the driver instance specified in the request
5. The device driver communicates with the end device
6. The end device returns a response to the driver indicating its current status
7. The driver publishes the device's response to the message bus using a publish

1.29.2 Installation

The Master Driver must first be *configured*, similarly to other agents.

Then, the user must add driver configurations, registry configurations, and any other referenced configurations to the config store if they do not already exist.

Adding Device Configurations to the Configuration Store

Configurations are added to the Configuration Store using the command line:

```
volttron-ctl config store platform.driver <name> <file name> <file type>
```

- **name** - The name used to refer to the file from the store.
- **file name** - A file containing the contents of the configuration.
- **file type** - `--raw`, `--json`, or `--csv`. Indicates the type of the file. Defaults to `--json`.

The main configuration must have the name `config`

Device configuration but **not** registry configurations must have a name prefixed with `devices/`. Scripts that automate the process will prefix registry configurations with `registry_configs/`, but that is not a requirement for registry files.

The name of the device's configuration in the store is used to create the topic used to reference the device. For instance, a configuration named *devices/PNNL/ISB1/vav1* will publish scrape results to *devices/PNNL/ISB1/vav1* and is accessible with the Actuator Agent via *PNNL/ISB1/vav1*.

The name of a registry configuration must match the name used to refer to it in the driver configuration. The reference is not case sensitive.

If the Master Driver Agent is running any changes to the configuration store will immediately affect the running devices according to the changes.

Example

Consider the following three configuration files: A master driver configuration called *master-driver.agent*, a Modbus device configuration file called *modbus_driver.config* and corresponding Modbus registry configuration file called *modbus_registry.csv*

To store the master driver configuration run the command:

```
volttron-ctl config store platform.driver config master-driver.agent
```

To store the registry configuration run the command (note the `--csv` option):

```
volttron-ctl config store platform.driver registry_configs/modbus_registry.csv modbus_
↪registry.csv --csv
```

Note: The *registry_configs/modbus_registry.csv* argument in the above command must match the reference to the *registry_config* found in *modbus_driver.config*.

To store the driver configuration run the command:

```
volttron-ctl config store platform.driver devices/my_campus/my_building/my_device␣
↪modbus_config.config
```

Converting Old Style Configuration

The new Master Driver no longer supports the old style of device configuration. The old *device_list* setting is ignored.

To simplify updating to the new format *scripts/update_master_driver_config.py* is provide to automatically update to the new configuration format.

With the platform running run:

```
python scripts/update_master_driver_config.py <old configuration> <output>
```

old_configuration`` is the main configuration file in the old format. The script automatically modifies the driver files to create references to CSV files and adds the CSV files with the appropriate name.

output is the target output directory.

If the `--keep-old` switch is used the old configurations in the output directory (if any) will not be deleted before new configurations are created. Matching names will still be overwritten.

The output from *scripts/update_master_driver_config.py* can be automatically added to the configuration store for the Master Driver agent with *scripts/install_master_driver_configs.py*.

Creating and naming configuration files in the form needed by *scripts/install_master_driver_configs.py* can speed up the process of changing and updating a large number of configurations. See the `--help` message for *scripts/install_master_driver_configs.py* for more details.

1.29.3 Usage

After installing the Master Driver and loading driver configs into the config store, the installed drivers begin polling and JSON-RPC endpoints become usable.

Polling

Once running, the Master Driver will spawn drivers using the *driver_type* parameter of the *driver configuration* and periodically poll devices for all point data specified in the *registry configuration* (at the interval specified by the interval parameter of the driver configuration).

Using the default configuration provided in the repository, device data collected during a "scrape all" is published to the *depth_first_all* topic for the device.

For more information on device data topics, please view the *device state publish* docs.

JSON-RPC Endpoints

get_point - Returns the value of specified device set point

Parameters

- **path** - device topic string (typical format is devices/campus/building/device)
- **point_name** - name of device point from registry configuration file

set_point - Set value on specified device set point. If global override is condition is set, raise OverrideError exception.

Parameters

- **path** - device topic string (typical format is devices/campus/building/device)
- **point_name** - name of device point from registry configuration file
- **value** - desired value to set for point on device

> **Warning:** It is not recommended to call the *set_point* method directly. It is recommended to instead use the *Actuator* agent to set points on a device, using its scheduling capability.

scrape_all - Returns values for all set points on the specified device.

Parameters

- **path** - device topic string (typical format is devices/campus/building/device)

get_multiple_points - return values corresponding to multiple points on the same device

Parameters

- **path** - device topic string (typical format is devices/campus/building/device)
- **point_names** - iterable of device point names from registry configuration file

set_multiple_points - Set values on multiple set points at once. If global override is condition is set, raise OverrideError exception.

Parameters

- **path** - device topic string (typical format is devices/campus/building/device)
- **point_names_value** - list of tuples consisting of (point_name, value) pairs for setting a series of points

heart_beat - Send a heartbeat/keep-alive signal to all devices configured for Master Driver

revert_point - Revert the set point of a device to its default state/value. If global override is condition is set, raise OverrideError exception.

Parameters

- **path** - device topic string (typical format is devices/campus/building/device)
- **point_name** - name of device point from registry configuration file

revert_device - Revert all the set point values of the device to default state/values. If global override is condition is set, raise OverrideError exception.

Parameters

- **path** - device topic string (typical format is devices/campus/building/device)

set_override_on - Turn on override condition on all the devices matching the specified pattern (*override docs*)

Parameters

- **pattern - Override pattern to be applied. For example,**
  - If pattern is *campus/building1/** - Override condition is applied for all the devices under *campus/building1/*.
  - If pattern is *campus/building1/ahu1* - Override condition is applied for only *campus/building1/ahu1* The pattern matching is based on bash style filename matching semantics.
- **duration** - Duration in seconds for the override condition to be set on the device (default 0.0, duration <= 0.0 imply indefinite duration)
- **failsafe_revert** - Flag to indicate if all the devices falling under the override condition must to be set to its default state/value immediately.
- **staggered_revert** -

set_override_off - Turn off override condition on all the devices matching the pattern.

Parameters

- **pattern** - device topic pattern for devices on which the override condition should be removed.

get_override_devices - Get a list of all the devices with override condition.

clear_overrides - Turn off override condition for all points on all devices.

get_override_patterns - Get a list of all override condition patterns currently set.

1.30 Actuator Agent

This agent is used to manage write access to devices. Agents may request scheduled times, called Tasks, to interact with one or more devices.

1.30.1 Actuator Agent Communication

Scheduling a Task

An agent can request a task schedule by publishing to the *devices/actuators/schedule/request* topic with the following header:

```
{
    'type': 'NEW_SCHEDULE',
    'requesterID': <Ignored, VIP Identity used internally>
    'taskID': <unique task ID>, #The desired task ID for this task. It must be unique␣
↪among all other scheduled tasks.
    'priority': <task priority>, #The desired task priority, must be 'HIGH', 'LOW',␣
↪or 'LOW_PREEMPT'
}
```

with the following message:

```
[
    ["campus/building/device1", #First time slot.
     "2013-12-06 16:00:00",      #Start of time slot.
     "2013-12-06 16:20:00"],     #End of time slot.
    ["campus/building/device1", #Second time slot.
     "2013-12-06 18:00:00",      #Start of time slot.
     "2013-12-06 18:20:00"],     #End of time slot.
    ["campus/building/device2", #Third time slot.
     "2013-12-06 16:00:00",      #Start of time slot.
     "2013-12-06 16:20:00"],     #End of time slot.
    #etc...
]
```

Warning: If time zones are not included in schedule requests then the Actuator will interpret them as being in local time. This may cause remote interaction with the actuator to malfunction.

Points on Task Scheduling

- Everything in the header is required
- Task id and requester id (agentid) should be a non empty value of type string
- A Task schedule must have at least one time slot.
- The start and end times are parsed with dateutil's date/time parser. **The default string representation of a python datetime object will parse without issue.**
- Two Tasks are considered conflicted if at least one time slot on a device from one task overlaps the time slot of the other on the same device.
- The end time of one time slot can be the same as the start time of another time slot for the same device. This will not be considered a conflict. For example, `time_slot1(device0, time1, **time2**)` and `time_slot2(device0, **time2**, time3)` are not considered a conflict
- A request must not conflict with itself
- If something goes wrong see *this failure string list* for an explanation of the error.

Task Priorities

- *HIGH*: This Task cannot be preempted under any circumstance. This task may preempt other conflicting preemptable Tasks.
- *LOW*: This Task cannot be preempted **once it has started**. A Task is considered started once the earliest time slot on any device has been reached. This Task may **not** preempt other Tasks.
- *LOW_PREEMPT*: This Task may be preempted at any time. If the Task is preempted once it has begun running any current time slots will be given a grace period (configurable in the ActuatorAgent configuration file, defaults to 60 seconds) before being revoked. This Task may **not** preempt other Tasks.

Canceling a Task

A task may be canceled by publishing to the *devices/actuators/schedule/request* topic with the following header:

```
{
    'type': 'CANCEL_SCHEDULE',
    'requesterID': <Ignored, VIP Identity used internally>
    'taskID': <unique task ID>, #The desired task ID for this task. It must be unique␣
↪among all other scheduled tasks.
}
```

Points on Task Canceling

- The requesterID and taskID must match the original values from the original request header.
- After a Tasks time has passed there is no need to cancel it. Doing so will result in a *TASK_ID_DOES_NOT_EXIST* error.
- If something goes wrong see *this failure string list* for an explanation of the error.

Actuator Agent Schedule Response

In response to a Task schedule request the ActuatorAgent will respond on the topic *devices/actuators/schedule/result* with the header:

```
{
    'type': <'NEW_SCHEDULE', 'CANCEL_SCHEDULE'>
    'requesterID': <Agent VIP identity from the request>,
    'taskID': <Task ID from the request>
}
```

And the message (after parsing the json):

```
{
    'result': <'SUCCESS', 'FAILURE', 'PREEMPTED'>,
    'info': <Failure reason, if any>,
    'data': <Data about the failure or cancellation, if any>
}
```

The Actuator Agent may publish cancellation notices for preempted Tasks using the *PREEMPTED* result.

Preemption Data

Preemption data takes the form:

```
{
    'agentID': <Agent ID of preempting task>,
    'taskID': <Task ID of preempting task>
}
```

Failure Reasons

In many cases the Actuator Agent will try to give good feedback as to why a request failed.

General Failures

- *INVALID_REQUEST_TYPE*: Request type was not *NEW_SCHEDULE* or *CANCEL_SCHEDULE*.
- *MISSING_TASK_ID*: Failed to supply a taskID.
- *MISSING_AGENT_ID*: AgentID not supplied.

Task Schedule Failures

- *TASK_ID_ALREADY_EXISTS*: The supplied taskID already belongs to an existing task.
- *MISSING_PRIORITY*: Failed to supply a priority for a Task schedule request.
- *INVALID_PRIORITY*: Priority not one of *HIGH*, *LOW*, or *LOW_PREEMPT*.
- *MALFORMED_REQUEST_EMPTY*: Request list is missing or empty.
- *REQUEST_CONFLICTS_WITH_SELF*: Requested time slots on the same device overlap.
- *MALFORMED_REQUEST*: Reported when the request parser raises an unhandled exception. The exception name and info are appended to this info string.
- *CONFLICTS_WITH_EXISTING_SCHEDULES*: This schedule conflict with an existing schedules that it cannot preempt. The data item for the results will contain info about the conflicts in this form (after parsing json)

```
{
    '<agentID1>':
    {
        '<taskID1>':
        [
            ["campus/building/device1",
             "2013-12-06 16:00:00",
             "2013-12-06 16:20:00"],
            ["campus/building/device1",
             "2013-12-06 18:00:00",
             "2013-12-06 18:20:00"]
        ]
        '<taskID2>':[...]
    }
    '<agentID2>': {...}
}
```

Task Cancel Failures

- *TASK_ID_DOES_NOT_EXIST*: Trying to cancel a Task which does not exist. This error can also occur when trying to cancel a finished Task.
- *AGENT_ID_TASK_ID_MISMATCH*: A different agent ID is being used when trying to cancel a Task.

Actuator Agent Value Request

Once an Task has been scheduled and the time slot for one or more of the devices has started an agent may interact with the device using the **get** and **set** topics.

Both **get** and **set** are responded to the same way. See *Actuator Reply* below.

Getting values

While a driver for a device should always be setup to periodically broadcast the state of a device you may want an up-to-the-moment value for an actuation point on a device.

To request a value publish a message to the following topic:

```
'devices/actuators/get/<full device path>/<actuation point>'
```

Setting Values

Value are set in a similar manner:

To set a value publish a message to the following topic:

```
'devices/actuators/set/<full device path>/<actuation point>'
```

With this header:

```
#python
{
    'requesterID': <Ignored, VIP Identity used internally>
}
```

And the message contents being the new value of the actuator.

Warning: The actuator agent expects all messages to be JSON and will parse them accordingly. Use *publish_json* to send messages where possible. This is significant for Boolean values especially

Actuator Reply

The ActuatorAgent will reply to both *get* and *set* on the *value* topic for an actuator:

```
'devices/actuators/value/<full device path>/<actuation point>'
```

With this header:

```
{
    'requesterID': <Agent VIP identity>
}
```

With the message containing the value encoded in JSON.

Actuator Error Reply

If something goes wrong the Actuator Agent will reply to both *get* and *set* on the *error* topic for an actuator:

```
'devices/actuators/error/<full device path>/<actuation point>'
```

With this header:

```
{
    'requesterID': <Agent VIP identity>
}
```

The message will be in the following form:

```
{
    'type': <Error Type or name of the exception raised by the request>
    'value': <Specific info about the error>
}
```

Common Error Types

- *LockError*: Returned when a request is made when we do not have permission to use a device. (Forgot to schedule, preempted and we did not handle the preemption message correctly, ran out of time in time slot, etc. . .)
- *ValueError*: Message missing or could not be parsed as JSON

Schedule State Broadcast

Periodically the ActuatorAgent will publish the state of all currently scheduled devices. For each device the Actuator-Agent will publish to an associated topic:

```
'devices/actuators/schedule/announce/<full device path>'
```

With the following header:

```
{
    'requesterID': <VIP identity of agent with access>,
    'taskID': <Task associated with the time slot>
    'window': <Seconds remaining in the time slot>
}
```

The frequency of the updates is configurable with the *schedule_publish_interval* setting.

Task Preemption

Both *LOW* and *LOW_PREEMPT* priority Tasks can be preempted. *LOW* priority Tasks may be preempted by a conflicting *HIGH* priority Task before it starts. *LOW_PREEMPT* priority Tasks can be preempted by *HIGH* priority Tasks even after they start.

When a Task is preempted the ActuatorAgent will publish to *devices/actuators/schedule/response* with the following header:

```
{
    'type': 'CANCEL_SCHEDULE',
    'requesterID': <Agent VIP identity for the preempted Task>,
    'taskID': <Task ID for the preempted Task>
}
```

And the message (after parsing the json):

```
{
    'result': 'PREEMPTED',
    'info': '',
    'data':
    {
        'agentID': <Agent VIP identity of preempting task>,
        'taskID': <Task ID of preempting task>
    }
}
```

Preemption Grace Time

If a *LOW_PREEMPT* priority Task is preempted while it is running the Task will be given a grace period to clean up before ending. For every device which has a current time slot the window of remaining time will be reduced to the grace time. At the end of the grace time the Task will finish. If the Task has no currently open time slots on any devices it will end immediately.

ActuatorAgent Configuration

- *schedule_publish_interval*: Interval between current schedules being published to the message bus for all devices
- *preempt_grace_time*: Minimum time given to Tasks which have been preempted to clean up in seconds. Defaults to 60
- *schedule_state_file*: File used to save and restore Task states if the ActuatorAgent restarts for any reason. File will be created if it does not exist when it is needed

Sample configuration file

```
{
 "schedule_publish_interval": 30,
 "schedule_state_file": "actuator_state.pickle"
}
```

Heartbeat Signal

The ActuatorAgent can be configured to send a heartbeat message to the device to indicate the platform is running. Ideally, if the heartbeat signal is not sent the device should take over and resume normal operation.

The configuration has two parts, the interval (in seconds) for sending the heartbeat and the specific point that should be modified each iteration.

The heart beat interval is specified with a global *heartbeat_interval* setting. The ActuatorAgent will automatically set the heartbeat point to alternating "1" and "0" values. Changes to the heartbeat point will be published like any other value change on a device.

The heartbeat points are specified in the driver configuration file of individual devices.

Notes on Working With the ActuatorAgent

- An agent can watch the window value from *device state updates* to perform scheduled actions within a timeslot
  - If an Agent's Task is *LOW_PREEMPT* priority it can watch for device state updates where the window is less than or equal to the grace period (default 60.0)
- When considering if to schedule long or multiple short time slots on a single device:
  - Do we need to ensure the device state for the duration between slots?
    - Yes: Schedule one long time slot instead
    - No: Is it all part of the same Task or can we break it up in case there is a conflict with one of our time slots?
- When considering time slots on multiple devices for a single Task:
  - Is the Task really dependent on all devices or is it actually multiple Tasks?
- When considering priority:
  - Does the Task have to happen **on an exact day**?
    - Yes: Use *HIGH*
    - No: Consider *LOW* and reschedule if preempted
  - Is it problematic to prematurely stop a Task once started?
    - Yes: Consider *LOW* or *HIGH*
    - No: Consider *LOW_PREEMPT* and watch the device state updates for a small window value
- If an agent is only observing but needs to assure that no another Task is going on while taking readings it can schedule the time to prevent other agents from messing with a devices state. The schedule updates can be used as a reminder as to when to start watching
- **Any** device, existing or not, can be scheduled. This allows for agents to schedule fake devices to create reminders to start working later rather then setting up their own internal timers and schedules

1.31 Fake Driver

The FakeDriver is included as a way to quickly see data published to the message bus in a format that mimics what a true Driver would produce. This is an extremely simple implementation of the *VOLTTRON driver framework*.

1.31.1 Fake Device Driver Configuration

This driver does not connect to any actual device and instead produces random and or pre-configured values.

Driver Config

There are no arguments for the *driver_config* section of the device configuration file. The *driver_config* entry must still be present and should be left blank.

Here is an example device configuration file:

```
{
    "driver_config": {},
    "driver_type": "bacnet",
    "registry_config":"config://registry_configs/vav.csv",
    "interval": 5,
    "timezone": "UTC",
    "heart_beat_point": "heartbeat"
}
```

A sample fake device configuration file can be found in the VOLTTRON repository in *examples/configurations/drivers/fake.config*

Fake Device Registry Configuration File

The registry configuration file is a CSV file. Each row configures a point on the device.

The following columns are required for each row:

- **Volttron Point Name** - The name by which the platform and agents running on the platform will refer to this point. For instance, if the *Volttron Point Name* is *HeatCall1* (and using the example device configuration above) then an agent would use *pnnl/isb2/hvac1/HeatCall1* to refer to the point when using the RPC interface of the actuator agent.
- **Units** - Used for meta data when creating point information on the historian.
- **Writable** - Either *TRUE* or *FALSE*. Determines if the point can be written to. Only points labeled *TRUE* can be written to through the ActuatorAgent. Points labeled *TRUE* incorrectly will cause an error to be returned when an agent attempts to write to the point.

The following columns are optional:

- **Starting Value** - Initial value for the point. If the point is reverted it will change back to this value. By default, points will start with a random value (1-100).
- **Type** - Value type for the point. Defaults to "string". Valid types are:
  - string
  - integer
  - float
  - boolean

Any additional columns will be ignored. It is common practice to include a *Point Name* or *Reference Point Name* to include the device documentation's name for the point and *Notes* and *Unit Details* for additional information about a point. Please note that there is nothing in the driver that will enforce anything specified in the *Unit Details* column.

Table 15: BACnet

Volttron Point Name	Units	Units Details	Writable	Starting Value	Type	Notes
Heartbeat	On/Off	On/Off	TRUE	0	boolean	Point for heartbeat toggle
Out-sideAirTemper-ature1	F	-100 to 300	FALSE	50	float	CO2 Reading 0.00-2000.0 ppm
Sam-pleWritable-Float1	PPM	10.00 (default)	TRUE	10	float	Setpoint to enable demand con-trol ventilation
SampleLong1	Enu-mera-tion	1 through 13	FALSE	50	int	Status indicator of service switch
SampleWrita-bleShort1	%	0.00 to 100.00 (20 default)	TRUE	20	int	Minimum damper position during the standard mode
SampleBool1	On / Off	on/off	FALSE	TRUE	boolean	Status indicator of cooling stage 1
Sam-pleWritable-Bool1	On / Off	on/off	TRUE	TRUE	boolean	Status indicator

A sample fake registry configuration file can be found here or in the VOLTTRON repository in `examples/configurations/drivers/fake.csv`

1.31.2 Installation

Installing a Fake driver in the *Master Driver Agent* requires adding copies of the device configuration and registry configuration files to the Master Driver's *configuration store*

- Create a config directory (if one doesn't already exist) inside your Volttron repository:

```
mkdir config
```

All local config files will be worked on here.

- Copy over the example config file and registry config file from the VOLTTRON repository:

```
cp examples/configurations/drivers/fake.config config/
cp examples/configurations/drivers/fake.csv config/
```

- Edit the driver config *fake.config* for the paths on your system:

```
{
    "driver_config": {},
    "registry_config": "config://fake.csv",
    "interval": 5,
    "timezone": "US/Pacific",
    "heart_beat_point": "Heartbeat",
    "driver_type": "fakedriver",
    "publish_breadth_first_all": false,
    "publish_depth_first": false,
    "publish_breadth_first": false
    }
```

- Create a copy of the Master Driver config from the VOLTTRON repository:

```
cp examples/configurations/drivers/master-driver.agent config/fake-master-driver.
↪config
```

- Add fake.csv and fake.config to the *configuration store*:

```
vctl config store platform.driver devices/campus/building/fake config/fake.config
vcfl config store platform.driver fake.csv config/fake.csv --csv
```

- Edit *fake-master-driver.config* to reflect paths on your system

```
{
    "driver_scrape_interval": 0.05
}
```

- Use the scripts/install-agent.py script to install the Master Driver agent:

```
python scripts/install-agent.py -s services/core/MasterDriverAgent -c config/fake-
↪master-driver.config
```

- If you have a *Listener Agent* already installed, you should start seeing data being published to the bus.

1.32 BACnet Driver

1.32.1 BACnet Driver Configuration

Communicating with BACnet devices requires that the *BACnet Proxy Agent* is configured and running. All device communication happens through this agent.

Requirements

The BACnet driver requires the BACPypes package. This package can be installed in an activated environment with:

```
pip install bacpypes
```

Alternatively, running *bootstrap.py* with the `--drivers` option will install all requirements for drivers included in the repository including BACnet.

```
python3 bootstrap.py --drivers
```

Warning: Current versions of VOLTTRON support **only** BACPypes version 0.16.7

Driver Config

There are nine arguments for the *driver_config* section of the device configuration file:

- **device_address** - Address of the device. If the target device is behind an IP to MS/TP router then Remote Station addressing will probably be needed for the driver to find the device
- **device_id** - BACnet ID of the device. Used to establish a route to the device at startup

- **min_priority** - (Optional) Minimum priority value allowed for this device whether specifying the priority manually or via the registry config. Violating this parameter either in the configuration or when writing to the point will result in an error. Defaults to 8
- **max_per_request** - (Optional) Configure driver to manually segment read requests. The driver will only grab up to the number of objects specified in this setting at most per request. This setting is primarily for scraping many points off of low resource devices that do not support segmentation. Defaults to 10000
- **proxy_address** - (Optional) VIP address of the BACnet proxy. Defaults to "platform.bacnet_proxy". See *Communicating With Multiple BACnet Networks* for details. Unless your BACnet network has special needs you should not change this value
- **ping_retry_interval** - (Optional) The driver will ping the device to establish a route at startup. If the BACnet proxy is not available the driver will retry the ping at this interval until it succeeds. Defaults to 5
- **use_read_multiple** - (Optional) During a scrape the driver will tell the proxy to use a ReadPropertyMultipleRequest to get data from the device. Otherwise the proxy will use multiple ReadPropertyRequest calls. If the BACnet proxy is reporting a device is rejecting requests try changing this to false for that device. Be aware that setting this to false will cause scrapes for that device to take much longer. Only change if needed. Defaults to true
- **cov_lifetime** - (Optional) When a device establishes a change of value subscription for a point, this argument will be used to determine the lifetime and renewal period for the subscription, in seconds. Defaults to 180 (Added to Master Driver version 3.2)

Here is an example device configuration file:

```
{
    "driver_config": {"device_address": "10.1.1.3",
                      "device_id": 500,
                      "min_priority": 10,
                      "max_per_request": 24
                      },
    "driver_type": "bacnet",
    "registry_config":"config://registry_configs/vav.csv",
    "interval": 5,
    "timezone": "UTC",
    "heart_beat_point": "heartbeat"
}
```

A sample BACnet configuration file can be found in the VOLTTRON repository at *examples/configurations/drivers/bacnet1.config*

BACnet Registry Configuration File

The registry configuration file is a CSV file. Each row configures a point on the device.

Most of the configuration file can be generated with the *grab_bacnet_config.py* utility in *scripts/bacnet*. See *BACnet Auto-Configuration*.

Currently, the driver provides no method to access array type properties even if the members of the array are of a supported type.

The following columns are required for each row:

- **Volttron Point Name** - The name by which the platform and agents running on the platform will refer to this point. For instance, if the Volttron Point Name is *HeatCall1* (and using the example device configuration above) then an agent would use *pnnl/isb2/hvac1/HeatCall1* to refer to the point when using the RPC interface of the Actuator agent

- **Units** - Used for meta data when creating point information on the historian.
- **BACnet Object Type** - A string representing what kind of BACnet standard object the point belongs to. Examples include:
  - analogInput
  - analogOutput
  - analogValue
  - binaryInput
  - binaryOutput
  - binaryValue
  - multiStateValue
- **Property** - A string representing the name of the property belonging to the object. Usually, this will be *presentValue*
- **Writable** - Either *TRUE* or *FALSE*. Determines if the point can be written to. Only points labeled *TRUE* can be written to through the Actuator Agent. Points labeled *TRUE* incorrectly will cause an error to be returned when an agent attempts to write to the point
- **Index** - Object ID of the BACnet object

The following columns are optional:

- **Write Priority** - BACnet priority for writing to this point. Valid values are 1-16. Missing this column or leaving the column blank will use the default priority of 16
- **COV Flag** - Either *True* or *False*. Determines if a BACnet Change-of-Value subscription should be established for this point. Missing this column or leaving the column blank will result in no change of value subscriptions being established. (Added to Master Driver version 3.2)

Any additional columns will be ignored. It is common practice to include a *Point Name* or *Reference Point Name* column to include the device documentation's name for the point and *Notes* and *Unit Details* columns for additional information about a point.

Table 16: BACnet

Point Name	Volttron Point Name	Units	Unit Details	BACnet Object Type	Property	Writable	Index	Notes
Building/FCB.Local Application.PH-T	PreheatTemperature	degreesFahrenheit	-50.00 to 250.00	analogInput	presentValue	FALSE	3000119	Resolution: 0.1
Building/FCB.Local Application.RA-T	ReturnAirTemperature	degreesFahrenheit	-50.00 to 250.00	analogInput	presentValue	FALSE	3000120	Resolution: 0.1
Building/FCB.Local Application.RA-H	ReturnAirHumidity	percentRelativeHumidity	0.00 to 100.00	analogInput	presentValue	FALSE	3000124	Resolution: 0.1
Building/FCB.Local Application.CLG-O	CoolingValveOutputCommand	percent	0.00 to 100.00 (default 0.0)	analogOutput	presentValue	TRUE	3000107	Resolution: 0.1
Building/FCB.Local Application.MAD-O	MixedAirDamperOutputCommand	percent	0.00 to 100.00 (default 0.0)	analogOutput	presentValue	TRUE	3000110	Resolution: 0.1
Building/FCB.Local Application.PH-O	PreheatValveOutputCommand	percent	0.00 to 100.00 (default 0.0)	analogOutput	presentValue	TRUE	3000111	Resolution: 0.1
Building/FCB.Local Application.RH-O	ReheatValveOutputCommand	percent	0.00 to 100.00 (default 0.0)	analogOutput	presentValue	TRUE	3000112	Resolution: 0.1
Building/FCB.Local Application.SF-O	SupplyFanSpeedOutputCommand	percent	0.00 to 100.00 (default 0.0)	analogOutput	presentValue	TRUE	3000113	Resolution: 0.1

A sample BACnet registry file can be found here or in the VOLTTRON repository in *examples/configurations/drivers/bacnet.csv*

BACnet Proxy Agent

Introduction

Communication with BACnet device on a network happens via a single virtual BACnet device. In VOLTTRON driver framework, we use a separate agent specifically for communicating with BACnet devices and managing the virtual BACnet device.

Requirements

The BACnet Proxy agent requires the BACPypes package. This package can be installed in an activated environment with:

```
pip install bacpypes
```

Alternatively, running *bootstrap.py* with the *–drivers* option will install all requirements for drivers included in the repository including BACnet.

```
python3 bootstrap.py --drivers
```

Warning: Current versions of VOLTTRON support **only** BACPypes version 0.16.7

Configuration

The agent configuration sets up the virtual BACnet device.

```
{
    "device_address": "10.0.2.15",
    "max_apdu_length": 1024,
    "object_id": 599,
    "object_name": "Volttron BACnet driver",
    "vendor_id": 15,
    "segmentation_supported": "segmentedBoth"
}
```

BACnet device settings

- **device_address** - Address bound to the network port over which BACnet communication will happen on the computer running VOLTTRON. This is **NOT** the address of any target device. See *BACnet Router Addressing*.
- **object_id** - ID of the Device object of the virtual BACnet device. Defaults to 599. Only needs to be changed if there is a conflicting BACnet device ID on your network.

These settings determine the capabilities of the virtual BACnet device. BACnet communication happens at the lowest common denominator between two devices. For instance, if the BACnet proxy supports segmentation and the target device does not communication will happen without segmentation support and will be subject to those limitations. Consequently, there is little reason to change the default settings outside of the *max_apdu_length* (the default is not the largest possible value).

- **max_apdu_length** - (From bacpypes documentation) BACnet works on lots of different types of networks, from high-speed Ethernet to “slower” and “cheaper” ARCNET or MS/TP (a serial bus protocol used for a field bus defined by BACnet). For devices to exchange messages they have to know the maximum size message the device can handle. (End BACpypes docs)

  This setting determines the largest APDU (Application Protocol Data Unit) accepted by the BACnet virtual device. Valid options are 50, 128, 206, 480, 1024, and 1476. Defaults to 1024.(Optional)
- **object_name** - Name of the object. Defaults to “Volttron BACnet driver”. (Optional)
- **vendor_id** - Vendor ID of the virtual BACnet device. Defaults to 15. (Optional)
- **segmentation_supported** - (From bacpypes documentation) A vast majority of BACnet communications traffic fits into one message, but there can be times when larger messages are convenient and more efficient. Segmentation allows larger messages to be broken up into segments and spliced back together. It is not unusual for “low power” field equipment to not support segmentation. (End BACpypes docs)

  Possible setting are “segmentedBoth” (default), “segmentedTransmit”, “segmentedReceive”, or “noSegmentation” (Optional)

Device Addressing

In some cases, it will be needed to specify the subnet mask of the virtual device or a different port number to listen on. The full format of the BACnet device address is:

```
<ADDRESS>/<NETMASK>:<PORT>
```

where `<PORT>` is the port to use and `<NETMASK>` is the netmask length. The most common value is 24. See http://www.computerhope.com/jargon/n/netmask.htm

For instance, if you need to specify a subnet mask of `255.255.255.0` and the IP address bound to the network port is `192.168.1.2` you would use the address:

```
192.168.1.2/24
```

If your BACnet network is on a different port (47809) besides the default (47808) you would use the address:

```
192.168.1.2:47809
```

If you need to do both:

```
192.168.1.2/24:47809
```

Communicating With Multiple BACnet Networks

If two BACnet devices are connected to different ports they are considered to be on different BACnet networks. In order to communicate with both devices, you will need to run one BACnet Proxy Agent per network.

Each proxy will need to be bound to different ports appropriate for each BACnet network and will need a different VIP identity specified. When configuring drivers you will need to specify which proxy to use by *specifying the VIP identity*.

For example, a proxy connected to the default BACnet network:

```
{
    "device_address": "192.168.1.2/24"
}
```

and another on port 47809:

```
{
    "device_address": "192.168.1.2/24:47809"
}
```

a device on the first network:

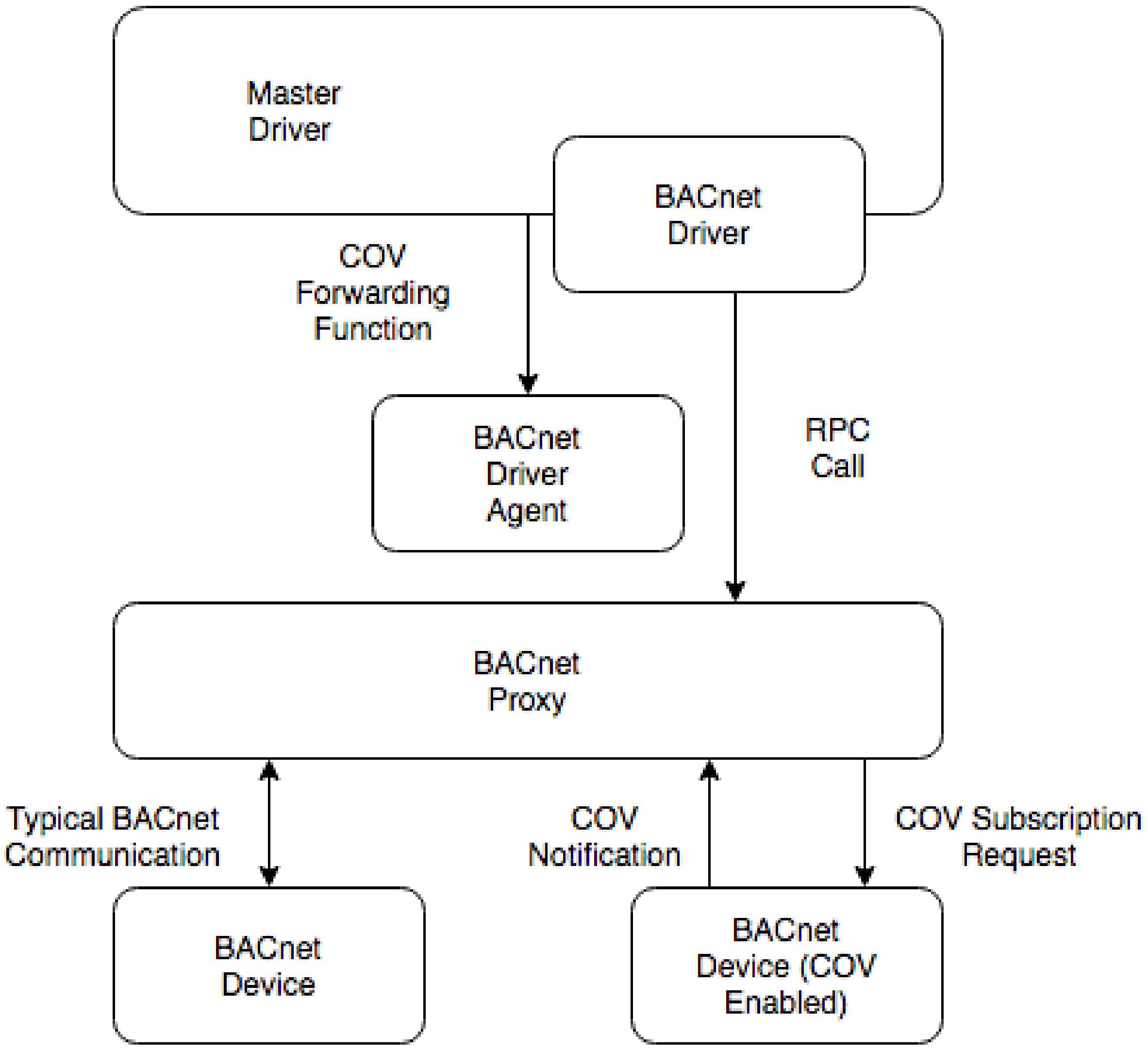
```
{
    "driver_config": {"device_address": "1002:12",
                      "proxy_address": "platform.bacnet_proxy_47808",
                      "timeout": 10},
    "driver_type": "bacnet",
    "registry_config":"config://registry_configs/bacnet.csv",
    "interval": 60,
    "timezone": "UTC",
    "heart_beat_point": "Heartbeat"
}
```

and a device on the second network:

```
{
    "driver_config": {"device_address": "12000:5",
                      "proxy_address": "platform.bacnet_proxy_47809",
                      "timeout": 10},
    "driver_type": "bacnet",
    "registry_config":"config://registry_configs/bacnet.csv",
    "interval": 60,
    "timezone": "UTC",
    "heart_beat_point": "Heartbeat"
}
```

Notice that both configs use the same registry configuration (config://registry_configs/bacnet.csv). This is perfectly fine as long as the registry configuration is appropriate for both devices. For scraping large numbers of points from a single BACnet device, there is an optional timeout parameter provided, to prevent the master driver timing out while the BACnet Proxy Agent is collecting points.

BACnet Change of Value Services



Change of Value Services added in version 0.5 of the BACnet Proxy and version 3.2 of the Master Driver.

There are a variety of scenarios in which a user may desire data from some BACnet device point values to be published independently of the regular scrape interval. Bacpypes provides a "ChangeOfValueServices" (hereby referred to as 'COV') module, which enables a device to push updates to the platform.

The BACnet COV requires that points on the device be properly configured for COV. A point on the BACnet device can be configured with the 'covIncrement' property, which determines the threshold for a COV notification (note: this property must be configured by the device operator - VOLTTRON does not provide the ability to set or modify this property).

Based on configuration options for BACnet drivers, the driver will instruct the BACnet Proxy to establish a COV subscription with the device. The subscription will last for an amount of time specified in the driver configuration, and will auto-renew the subscription. If the proxy loses communication with the device or the device driver is stopped the subscription will be removed when the lifetime expires.

While the subscription exists, the device will send (confirmed) notifications to which will be published, with the topic based on the driver's configured publish topics.

https://bacpypes.readthedocs.io/en/latest/modules/service/cov.html

BACnet Auto-Configuration

Included with the platform are two scripts for finding and configuring BACnet devices. These scripts are located in *scripts/bacnet*. *bacnet_scan.py* will scan the network for devices. *grab_bacnet_config.py* creates a CSV file for the BACnet driver that can be used as a starting point for creating your own register configuration.

Both scripts are configured with the file *BACpypes.ini*.

Configuring the Utilities

While running both scripts create a temporary virtual BACnet device using the *bacpypes* library. The virtual device must be configured properly in order to work. This configuration is stored in *scripts/bacnet/BACpypes.ini* and will be read automatically when the utility is run.

Note: The only value that (usually) needs to be changed is the **address** field.

Warning: This is the address bound to the port on the machine you are running the script from, **NOT A TARGET DEVICE**

This value should be set to the IP address of the network interface used to communicate with the remote device. If there is more than one network interface you must use the address of the interface connected to the network that can reach the device.

In Linux you can usually get the addresses bound to all interfaces by running `ifconfig` from the command line.

If a different outgoing port other than the default 47808 must be used, it can be specified as part of the address in the form:

```
<ADDRESS>:<PORT>
```

In some cases, the netmask of the network will be needed for proper configuration. This can be done following this format:

```
<ADDRESS>/<NETMASK>:<PORT>
```

where <NETMASK> is the netmask length. The most common value is 24. See http://www.computerhope.com/jargon/n/netmask.htm

In some cases, you may also need to specify a different device ID by changing the value of *objectIdentifier* so the virtual BACnet device does not conflict with any devices on the network. *objectIdentifier* defaults to 599.

Sample BACpypes.ini

```
[BACpypes]
objectName: Betelgeuse
address: 10.0.2.15/24
objectIdentifier: 599
maxApduLengthAccepted: 1024
segmentationSupported: segmentedBoth
vendorIdentifier: 15
```

Scanning for BACnet Devices

If the addresses for BACnet devices are unknown they can be discovered using the *bacnet_scan.py* utility.

To run the utility simply execute the following command:

```
python bacnet_scan.py
```

and expect output similar to this:

```
Device Address          = <Address 192.168.1.42>
Device Id               = 699
maxAPDULengthAccepted = 1024
segmentationSupported = segmentedBoth
vendorID                = 15

Device Address          = <RemoteStation 1002:11>
Device Id               = 540011
maxAPDULengthAccepted = 480
segmentationSupported = segmentedBoth
vendorID                = 5
```

Reading Output

The address where the device can be reached is listed on the *Device Address* line. The BACnet device ID is listed on the *Device Id* line. The remaining lines are informational and not needed to configure the BACnet driver.

For the first example, the IP address `192.168.1.42` can be used to reach the device. The second device is behind a BACnet router and can be reached at `1002:11`. See *BACnet router addressing*.

BACNet Scan Options

- `--address ADDRESS`: Send the WhoIs request only to a specific address. Useful as a way to ping devices on a network that blocks broadcast traffic.

- `--range LOW/HIGH`: Specify the device ID range for the results. Useful for filtering.
- `--timeout SECONDS`: Specify how long to wait for responses to the original broadcast. This defaults to 5 which should be sufficient for most networks.
- `--csv-out CSV_OUT`: Write the discovered devices to a CSV file. This can be used as inout for `grab_multiple_configs.py`. See *Scraping Multiple Devices*.

Automatically Generating a BACnet Registry Configuration File

A CSV registry configuration file for the BACnet driver can be generated with the `grab_bacnet_config.py` script.

Warning: This configuration will need to be edited before it can be used!

The utility is invoked with the command:

```
python grab_bacnet_config.py <device id>
```

This will query the device with the matching device ID for configuration information and print the resulting CSV file to the console.

In order to save the configuration to a file use the `--out-file` option to specify the output file name.

Optionally the `--address` option can be used to specify the address of the target. In some cases, this is needed to help establish a route to the device.

Output and Assumptions

- Attempts at determining if a point is writable proved too unreliable. Therefore all points are considered to be read-only in the output.
- The only property for which a point is setup for an object is *presentValue*.
- By default, the *Volttron Point Name* is set to the value of the *name* property of the BACnet object on the device. In most cases this name is vague. No attempt is made at choosing a better name. A duplicate of *Volttron Point Name* column called *Reference Point Name* is created to so that once *Volttron Point Name* is changed a reference remains to the actual BACnet device object name.
- Meta data from the objects on the device is used to attempt to put useful info in the *Units*, *Unit Details*, and `Notes` columns. Information such as the range of valid values, defaults, the resolution or sensor input, and enumeration or state names are scraped from the device.

With a few exceptions “Units” is pulled from the object’s “units” property and given the name used by the *bacpypes* library to describe it. If a value in the **Units** column takes the form

```
UNKNOWN UNIT ENUM VALUE: <value>
```

then the device is using a nonstandard value for the units on that object.

Scraping Multiple Devices

The *grab_multiple_configs.py* script will use the CSV output of *bacnet_scan.py* to automatically run *grab_bacnet_config.py* on every device listed in the CSV file.

The output is put in two directories. *devices/* contains basic driver configurations for the scrapped devices. *registry_configs/* contains the registry file generated by grab_bacnet_config.py.

grab_multiple_configs.py makes no assumptions about device names or topics, however the output is appropriate for the *install_master_driver_configs.py* script.

Grab Multiple Configs Options

- `--out-directory OUT_DIRECTORY` Specify the output directory.
- `--use-proxy` Use *proxy_grab_bacnet_config.py* to gather configuration data.

BACnet Proxy Alternative Scripts

Both *grab_bacnet_config.py* and *bacnet_scan.py* have alternative versions called *proxy_grab_bacnet_config.py* and *proxy_bacnet_scan.py* respectively. These versions require that the VOLTTRON platform is running and BACnet Proxy agent is running. Both of these agents use the same command line arguments as their independent counterparts.

Warning: These versions of the BACnet scripts are intended as a proof of concept and have not been optimized for performance. *proxy_grab_bacnet_config.py* takes about 10 times longer to grab a configuration than *grab_bacnet_config.py*

Problems and Debugging

- Both *grab_bacnet_config.py* and *bacnet_scan.py* creates a virtual device that open up a port for communication with devices. If the BACnet Proxy is running on the VOLTTRON platform it will cause both of these scripts to fail at startup. Stopping the BACnet Proxy will resolve the problem.
- Typically the utility should run quickly and finish in 30 seconds or less. In our testing, we have never seen a successful scrape take more than 15 seconds on a very slow device with many points. Many devices will scrape in less than 3 seconds.
- If the utility has not finished after about 60 seconds it is probably having trouble communicating with the device and should be stopped. Rerunning with debug output can help diagnose the problem.

To output debug messages to the console add the `--debug` switch to the **end** of the command line arguments.

```
python grab_bacnet_config.py <device ID> --out-file test.csv --debug
```

On a successful run you will see output similar to this:

```
DEBUG:<u>main</u>:initialization
DEBUG:<u>main</u>:    - args: Namespace(address='10.0.2.20', buggers=False, debug=[],␣
↪ini=<class 'bacpypes.consolelogging.ini'>, max_range_report=1e+20, out_file=<open␣
↪file 'out.csv', mode 'wb' at 0x901b0d0>)
DEBUG:<u>main</u>.SynchronousApplication:<u>init</u> (<bacpypes.app.LocalDeviceObject␣
↪object at 0x901de6c>, '10.0.2.15')
DEBUG:<u>main</u>:starting build
DEBUG:<u>main</u>:pduSource = <Address 10.0.2.20>
DEBUG:<u>main</u>:iAmDeviceIdentifier = ('device', 500)
DEBUG:<u>main</u>:maxAPDULengthAccepted = 1024
DEBUG:<u>main</u>:segmentationSupported = segmentedBoth
DEBUG:<u>main</u>:vendorID = 5
```

(continues on next page)

(continued from previous page)

```
DEBUG:<u>main</u>:device_name = MS-NCE2560-0
DEBUG:<u>main</u>:description =
DEBUG:<u>main</u>:objectCount = 32
DEBUG:<u>main</u>:object name = Building/FCB.Local Application.Room Real Temp 2
DEBUG:<u>main</u>:  object type = analogInput
DEBUG:<u>main</u>:  object index = 3000274
DEBUG:<u>main</u>:  object units = degreesFahrenheit
DEBUG:<u>main</u>:  object units details = -50.00 to 250.00
DEBUG:<u>main</u>:  object notes = Resolution: 0.1
DEBUG:<u>main</u>:object name = Building/FCB.Local Application.Room Real Temp 1
DEBUG:<u>main</u>:  object type = analogInput
DEBUG:<u>main</u>:  object index = 3000275
DEBUG:<u>main</u>:  object units = degreesFahrenheit
DEBUG:<u>main</u>:  object units details = -50.00 to 250.00
DEBUG:<u>main</u>:  object notes = Resolution: 0.1
DEBUG:<u>main</u>:object name = Building/FCB.Local Application.OSA
DEBUG:<u>main</u>:  object type = analogInput
DEBUG:<u>main</u>:  object index = 3000276
DEBUG:<u>main</u>:  object units = degreesFahrenheit
DEBUG:<u>main</u>:  object units details = -50.00 to 250.00
DEBUG:<u>main</u>:  object notes = Resolution: 0.1
...
```

and will finish something like this:

```
...
DEBUG:<u>main</u>:object name = Building/FCB.Local Application.MOTOR1-C
DEBUG:<u>main</u>:  object type = binaryOutput
DEBUG:<u>main</u>:  object index = 3000263
DEBUG:<u>main</u>:  object units = Enum
DEBUG:<u>main</u>:  object units details = 0-1 (default 0)
DEBUG:<u>main</u>:  object notes = BinaryPV: 0=inactive, 1=active
DEBUG:<u>main</u>:finally
```

Typically if the BACnet device is unreachable for any reason (wrong IP, network down/unreachable, wrong interface specified, device failure, etc) the scraper will stall at this message:

```
DEBUG:<u>main</u>:starting build
```

If you have not specified a valid interface in BACpypes.ini you will see the following error with a stack trace:

```
ERROR:<u>main</u>:an error has occurred: [Errno 99] Cannot assign requested address
<Python stack trace cut>
```

BACnet Router Addressing

The underlying library that Volttron uses for BACnet supports IP to MS/TP routers. Devices behind the router use a Remote Station address in the form:

```
<network>:<address>
```

where `<network>` is the configured network ID of the router and `<address>` is the address of the device behind the router.

For example to access the device at `<address>` 12 for a router configured for `<network>` 1002 can be accessed with this address:

```
1002:12
```

`<network>` must be number from **0 to 65534** and `<address>` must be a number from **0 to 255**.

This type of address can be used anywhere an address is required in configuration of the *Volttron BACnet driver*.

Caveats

VOLTTRON uses a UDP broadcast mechanism to establish the route to the device. If the route cannot be established it will fall back to a UDP broadcast for all communication with the device. If the IP network where the router is connected blocks UDP broadcast traffic then these addresses will not work.

1.33 Chargepoint Driver

1.33.1 Chargepoint Driver Configuration

The chargepoint driver requires at least one additional python library and has its own *requirements.txt*. Make sure to run:

```
pip install -r <chargepoint driver path>/requirements.txt
```

before using this driver.

driver_config

There are three arguments for the **driver_config** section of the device configuration file:

- **stationID** - Chargepoint ID of the station. This format is usually '1:00001'
- **username** - Login credentials for the Chargepoint API
- **password**- Login credentials for the Chargepoint API

The Chargepoint login credentials are generated in the Chargepoint web portal and require a Chargepoint account with sufficient privileges. Station IDs are also available on the web portal.

Here is an example device configuration file:

```
{
    "driver_config": {"stationID": "3:12345",
                      "username":
↪"4b90fc0ae5fe8b6628e50af1215d4fcf5743a6f3c63ee1464012875",
                      "password": "ebaf1a3cdfb80baf5b274bdf831e2648"},
    "driver_type": "chargepoint",
    "registry_config":"config://chargepoint.csv",
    "interval": 60,
    "timezone": "UTC",
    "heart_beat_point": "heartbeat"
}
```

A sample Chargepoint configuration file can be found in the VOLTTRON repository in *examples/configurations/drivers/chargepoint1.config*

Chargepoint Registry Configuration File

The registry configuration file is a CSV file. Each row configures a point on the device.

The following columns are required for each row:

- **Volttron Point Name** - The name by which the platform and agents running on the platform will refer to this point.
- **Attribute Name** - Chargepoint API attribute name. This determines the field that will be read from the API response and must be one of the allowed values.
- **Port #** - If the point describes a specific port on the Chargestation, it is defined here. (Note 0 and an empty value are equivalent.)
- **Type** - Python type of the point value.
- **Units** - Used for meta data when creating point information on the historian.
- **Writable** - Either "TRUE" or "FALSE". Determines if the point can be written to. Only points labeled TRUE can be written.
- **Notes** - Miscellaneous notes field.
- **Register Name - A string representing how to interpret the data register. Acceptable values are:**
    - StationRegister
    - StationStatusRegister
    - LoadRegister
    - AlarmRegister
    - StationRightsRegister
- **Starting Value** - Default value for writeable points. Read-only points should not have a value in this column.

Detailed descriptions for all available Chargepoint registers may be found in the `README.rst` in the Chargepoint driver directory.

A sample Chargepoint registry file can be found in the VOLTTRON repository in `examples/configurations/drivers/chargepoint.csv`

Chargepoint API Driver Specification

Spec Version 1.1

ChargePoint operates the largest independently owned EV charging network in the US. It sells charge stations to businesses and provides a web application to manage and report on these Chargestations. Chargepoint offers a Web Services API that its customers may use to develop applications that integrate with the Chargepoint network devices.

The Chargepoint API Driver for VOLTTRON will enable real-time monitoring and control of Chargepoint EVSEs within the VOLTTRON platform by creating a standard VOLTTRON device driver on top of the Chargepoint Web Services API. Each port on each managed Chargestation will look like a standard VOLTTRON device, monitored and controlled through the VOLTTRON device driver interface.

Driver Scope & Functions

This driver will enable VOLTTRON to support the following use cases with Chargepoint EVSEs:

- Monitoring of Chargestation status, load and energy consumption

- Demand charge reduction
- Time shifted charging
- Demand response program participation

The data and functionality to be made available through the driver interface will be implemented using the following Chargepoint web services:

API Method Name	Key Data/Function Provided
getStationStatus	Port status: AVAILABLE, INUSE, UNREACHABLE, UNKNOWN
shedLoad	Limit station power by percent or max load for some time period.
clearShedState	Clear all shed state and allow normal charging
getLoad	Port load in Kw, shedState, allowedLoad, percentShed
getAlarms	Only the last alarm will be available.
clearAlarms	Clear all alarms.
getStationRights	Name of station rights profile, eg. 'network_manager'
getChargingSessionData	Energy used in last session, start/end timestamps
getStations	Returns description/address/nameplate of chargestation.

The Chargepoint Driver will implement version 5.0 Rev 7 of the Chargepoint API. While the developer's guide is not yet publicly available, the WSDL Schema is.

Note: Station Reservation API has been removed from the 5.0 version of the API.*

WSDL for this API is located here:

https://webservices.chargepoint.com/cp_api_5.0.wsdl

Mapping VOLTTRON Device Interface to Chargepoint APIs

The VOLTTRON driver interface represents a single device as a list of registers accessed through a simple *get_point*/ *set_point* API. In contrast, the Chargepoint web services for real-time monitoring and control are spread across eight distinct APIs that return hierarchical XML. The Chargepoint driver is the adaptor that will make a suite of web services look like a single VOLTTRON device.

Device Mapping

The Chargepoint driver will map a single VOLTTRON device (a driver instance) to one Chargestation. Since a Chargestation can have multiple ports, each with their own set of telemetry, the registry will include a port index column on attributes that are specific to a port. This will allow deployments to use an indexing convention that has been followed with other drivers. (See Registry Configuration for more details)

Requirements

The Chargepoint driver requires at least one additional Python library and has its own *requirements.txt*. Make sure to run

```
pip install -r <chargepoint driver path>/requirements.txt
```

before using this driver.

Driver Configuration

Each device must be configured with its own driver configuration file. The driver configuration must reference the registry configuration file, defining the set of points that will be available from the device. For Chargestation devices, the *driver_config* entry of the driver Configuration file will need to contain all parameters required by the web service API:

Parameter	Purpose
username	Credentials established through Chargepoint account
password	
stationID	Unique station ID assigned by chargepoint

The *driver_type* must be `chargepoint`

A sample driver configuration file for a single device, looks like this:

```
{
    "driver_config": {
        "username"    : "1b905c936af141b98f9b0f816087f3605a30c1df1d07f146281b151",
        "password"    : "**Put your chargepoint API passqword here**",
        "stationID"   : "1:34003",
    },
    "driver_type": "chargepoint",
    "registry_config":"config://chargepoint.csv",
    "interval": 60,
    "heart_beat_point": "heartbeat"
}
```

API Plans & Access Rights

Chargepoint offers API plans that vary in available features and access rights. Some of the API calls to be implemented here are not available across all plans. Furthermore, the attributes returned in response to an API call may be limited by the API plan and access rights associated with the userid. Runtime exceptions related to plans and access rights will generate *DriverInterfaceError* exceptions. These can be avoided by using a registry configuration that does not include APIs or attributes that are not available to the <*username*>.

Registry Configuration

The registry file defines the individual points that will be exposed by the Chargepoint driver. It should only reference points that will actually be used since each point is potentially an additional web service call. The driver will be smart and limit API calls to those that are required to satisfy the points found in the CSV.

Naming of points will conform to the conventions established by the Chargepoint web services API whenever possible. Note that Chargepoint naming conventions are camel-cased with no spaces or hyphens. Multi-word names start with a lowercase letter. Single word names start uppercase.

The available registry entries for each API method name are shown below along with a description of any notable behavior associated with that register. Following that is a sample of the associated XML returned by the API.

getStationStatus

The *getStationStatus* query returns information for all ports on the Chargestation.

Note: In all the registry entries shown below, the **Attribute Name** column defines the unique name within the Chargepoint driver that must be used to reference this particular attribute and associated API. The **VOLTTRON point name** usually matches the **Attribute Name** in these examples but may be changed during deployment.

Table 17: getStationStatus

Volttron Point Name	Attribute Name	Register Name	Port #	Type	Units	Starting Value	Writable	Notes
Status	Status	Station-Status-Register	1	string			FALSE	AVAILABLE, INUSE, UNREACHABLE, UNKNOWN
Status.TimeStamp	TimeStamp	Station-Status-Register	1	datetime			FALSE	Timestamp of the last communication between the station and ChargePoint

Sample XML returned by *getStationStatus*.

```
<ns1:getStationStatusResponse xmlns:ns1="urn:dictionary:com.chargepoint.webservices">
    <responseCode>100</responseCode>
    <responseText>API input request executed successfully.</responseText>
    <stationData>
        <stationID>1:33923</stationID>
        <Port>
            <portNumber>1</portNumber>
            <Status>AVAILABLE</Status>
            <TimeStamp>2016-11-07T19:19:19Z</TimeStamp>
        </Port>
        <Port>
            <portNumber>2</portNumber>
            <Status>INUSE</Status>
            <TimeStamp>2016-11-07T19:19:19Z</TimeStamp>
        </Port>
    </stationData>
    <moreFlag>0</moreFlag>
</ns1:getStationStatusResponse>
```

getLoad, shedLoad, clearShedState

Reading any of these values will return the result of a call to *getLoad*. Writing `shedState=True` will call *shedLoad* and pass the last written value of *allowedLoad* or *percentShed*. The API allows only one of these two values to be provided. Writing to *allowedLoad* will simultaneously set *percentShed* to `None` and vice versa.

Table 18: getLoad, shedLoad, clearShedState

Volttron Point Name	Attribute Name	Register Name	Port #	Type	Units	Starting Value	Writable	Notes
shedState	shedState	LoadRegister	1	integer	0 or 1	0	TRUE	True when load shed limits are in place
portLoad	portLoad	LoadRegister	1	float	kw		FALSE	Load in kw
allowedLoad	allowedLoad	LoadRegister	1	float	kw		TRUE	Allowed load in kw when shedState is True
percentShed	percentShed	LoadRegister	1	integer	percent		TRUE	Percent of max power shed when shedState is True

Sample XML returned by *getLoad*

```
<ns1:getLoadResponse xmlns:ns1="urn:dictionary:com.chargepoint.webservices">
    <responseCode>100</responseCode>
    <responseText>API input request executed successfully.</responseText>
    <numStations></numStations>
    <groupName></groupName>
    <sgLoad></sgLoad>
    <stationData>
        <stationID>1:33923</stationID>
        <stationName>ALCOGARSTATIONS / ALCOPARK 8 -005</stationName><Address>165 13th␣
↪St, Oakland, California,  94612, United States</Address>
        <stationLoad>3.314</stationLoad>
        <Port>
            <portNumber>1</portNumber>
            <userID></userID>
            <credentialID></credentialID>
            <shedState>0</shedState>
            <portLoad>0.000</portLoad>
            <allowedLoad>0.000</allowedLoad>
            <percentShed>0</percentShed>
        </Port>
        <Port>
            <portNumber>2</portNumber>
            <userID>664719</userID>
            <credentialID>CNCP0000481668</credentialID>
            <shedState>0</shedState>
            <portLoad>3.314</portLoad>
            <allowedLoad>0.000</allowedLoad>
            <percentShed>0</percentShed>
        </Port>
    </stationData>
</ns1:getLoadResponse>
```

Sample *shedLoad* XML query to set the allowed load on a port to 3.0kw.

```
<ns1:shedLoad>
     <shedQuery>
       <shedStation>
         <stationID>1:123456</stationID>
         <Ports>
```

(continues on next page)

(continued from previous page)

```
            <Port>
              <portNumber>1</portNumber>
              <allowedLoadPerPort>3.0</allowedLoadPerPort>
            </Port>
          </Ports>
        </shedStation>
        <timeInterval/>
      </shedQuery>
    </ns1:shedLoad>
```

getAlarms, clearAlarms

The *getAlarms* query returns a list of all alarms since last cleared. The driver interface will only return data for the most recent alarm, if present. While the *getAlarm* query provides various station identifying attributes, these will be made available through registers associated with the *getStations* API. If an alarm is not specific to a particular port, it will be associated with all Chargestation ports and available through any of its device instances.

Write `True` to *clearAlarms* to submit the *clearAlarms* query to the **chargestation**. It will clear alarms across all ports on that Chargestation.

Table 19: getAlarms, clearAlarms

Volttron Point Name	Attribute Name	Register Name	Port #	Type	Units	Starting Value	Writable	Notes
alarmType	alarm-Type	Alarm-Register		string			FALSE	eg. ‘GFCI Trip’
alarmTime	alarm-Time	Alarm-Register		date-time			FALSE	
clearAlarms	clear-Alarms	Alarm-Register		int		0	TRUE	Sends the clearAlarms query when set to True

```
<Alarms>
    <stationID>1:33973</stationID>
    <stationName>ALCOGARSTATIONS / ALCOPARK 8 -003</stationName>
    <stationModel>CT2100-HD-CCR</stationModel>
    <orgID>1:ORG07225</orgID>
    <organizationName>Alameda County</organizationName>
    <stationManufacturer></stationManufacturer>
    <stationSerialNum>115110013418</stationSerialNum>
    <portNumber></portNumber>
    <alarmType>Reachable</alarmType>
    <alarmTime>2016-09-26T12:19:16Z</alarmTime>
    <recordNumber>1</recordNumber>
</Alarms>
```

getStationRights

Returns the name of the stations rights profile. A station may have multiple station rights profiles, each associated with a different station group ID. For this reason, the *stationRightsProfile* register will return a dictionary of *(sgID, name)* pairs. Since this is a Chargestation level attribute, it will be returned for all ports.

Table 20: getStationRights

Volttron Point Name	Attribute Name	Register Name	Port #	Type	Units	Starting Value	Writable	Notes
stationRightsProfile	stationRightsProfile	StationRightsRegister		dictionary			FALSE	Dictionary of sgID, rights name tuples.

```
<rightsData>
    <sgID>39491</sgID>
    <sgName>AlcoPark 8</sgName>
    <stationRightsProfile>network_manager</stationRightsProfile>
    <stationData>
        <stationID>1:34003</stationID>
        <stationName>ALCOGARSTATIONS / ALCOPARK 8 -004</stationName>
        <stationSerialNum>115110013369</stationSerialNum>
        <stationMacAddr>000D:6F00:0154:F1FC</stationMacAddr>
    </stationData>
</rightsData>
<rightsData>
    <sgID>58279</sgID>
    <sgName>AlcoGarageStations</sgName>
    <stationRightsProfile>network_manager</stationRightsProfile>
    <stationData>
        <stationID>1:34003</stationID>
        <stationName>ALCOGARSTATIONS / ALCOPARK 8 -004</stationName>
        <stationSerialNum>115110013369</stationSerialNum>
        <stationMacAddr>000D:6F00:0154:F1FC</stationMacAddr>
    </stationData>
</rightsData>
```

getChargingSessionData

Like *getAlarms*, this query returns a list of session data. The driver interface implementation will make the last session data available.

Table 21: getChargingSessionData

Volttron Point Name	Attribute Name	Register Name	Port #	Type	Units	Starting Value	Writable	Notes
sessionID	sessionID	ChargingSessionRegister	1	string			FALSE	
startTime	startTime	ChargingSessionRegister	1	datetime			FALSE	
endTime	endTime	ChargingSessionRegister	1	datetime			FALSE	
Energy	Energy	ChargingSessionRegister	1	float			FALSE	
rfidSerialNumber	rfidSerialNumber	ChargingSessionRegister	1	string			FALSE	
driverAccountNumber	driverAccountNumber	ChargingSessionRegister	1	string			FALSE	
driverName	driverName	ChargingSessionRegister	1	string			FALSE	

```
<ChargingSessionData>
    <stationID>1:34003</stationID>
    <stationName>ALCOGARSTATIONS / ALCOPARK 8 -004</stationName>
    <portNumber>2</portNumber>
    <Address>165 13th St, Oakland, California, 94612, United States</Address>
    <City>Oakland</City>
    <State>California</State>
    <Country>United States</Country>
    <postalCode>94612</postalCode>
    <sessionID>53068029</sessionID>
    <Energy>12.120572</Energy>
    <startTime>2016-10-25T15:53:35Z</startTime>
    <endTime>2016-10-25T20:14:46Z</endTime>
    <userID>452777</userID>
    <recordNumber>1</recordNumber>
    <credentialID>490178743</credentialID>
</ChargingSessionData>
```

getStations

This API call returns a complete description of the Chargestation in 40 fields. This information is essentially static and will change infrequently. It should not be scraped on a regular basis. The list of attributes will be included in the registry CSV but are only listed here:

```
stationID, stationManufacturer, stationModel, portNUmber, stationName, stationMacAddr,
↪ stationSerialNum, Address, City,
State, Country, postalCode, Lat, Long, Reservable, Level, Mode, Connector, Voltage,␣
↪Current, Power, numPorts, Type,
startTime, endTime, minPrice, maxPrice, unitPricePerHour, unitPricePerSession,␣
↪unitPricePerKWh, unitPricePerHourThereafter,
sessionTime, Description, mainPhone, orgID, organizationName, sgID, sgName,␣
↪currencyCode
```

Engineering Discussion

Questions

- **Allowed python-type** - We propose a register with a *python-type* of dictionary. Is this OK?
- **Scrape Interval** - Scrape all should not return all registers defined in the CSV, we propose fine grained control with a scrape-interval on each register. Response: ok to add extra settings to registry but don't worry about publishing static data with every scrape
- **Data currency** - Since devices are likely to share api calls, at least across ports, we need to think about the currency of the data and possibly allowing this to be a configurable parameter or derived from the scrape interval . Response: add to CSV with default values if not present

Performance

Web service calls across the internet will be significantly slower than typical VOLTTRON Bacnet or Modbus devices. It may be prohibitively expensive for each Chargepoint sub-agent instance to make individual requests on behalf of its own EVSE+port. We will need to examine the possibility of making a single request for all active Chargestations and sharing that information across driver instances. This could be done through a separate agent that regularly queries the Chargepoint network and makes the data available to each sub-agent via an RPC call.

3rd Party Library Dependencies

The Chargepoint driver implementation will depend on one additional 3rd part library that is not part of a standard VOLTTRON installation:

https://bitbucket.org/jurko/suds

Is there a mechanism for drivers to specify their own requirements.txt ?

Driver installation and configuration documentation can reference requirement.txt

1.34 DNP3 Driver

VOLTTRON's DNP3 driver enables the use of DNP3 (Distributed Network Protocol) communications, reading and writing points via a DNP3 Outstation.

In order to use a DNP3 driver to read and write point data, VOLTTRON's DNP3 Agent must also be configured and running. All communication between the VOLTTRON Outstation and a DNP3 Master happens through the DNP3 Agent.

For information about the DNP3 Agent, please see the *DNP3 Platform Specification*.

1.34.1 Requirements

The DNP3 driver requires the PyDNP3 package. This package can be installed in an activated environment with:

```
pip install pydnp3
```

1.34.2 Driver Configuration

There is one argument for the "driver_config" section of the DNP3 driver configuration file:

- **dnp3_agent_id** - ID of VOLTTRON's DNP3Agent.

Here is a sample DNP3 driver configuration file:

```
{
    "driver_config": {
        "dnp3_agent_id": "dnp3agent"
    },
    "campus": "campus",
    "building": "building",
    "unit": "dnp3",
    "driver_type": "dnp3",
    "registry_config": "config://dnp3.csv",
    "interval": 15,
    "timezone": "US/Pacific",
    "heart_beat_point": "Heartbeat"
}
```

A sample DNP3 driver configuration file can be found in the VOLTTRON repository in `services/core/MasterDriverAgent/example_configurations/test_dnp3.config`.

1.34.3 DNP3 Registry Configuration File

The driver's registry configuration file, a CSV file, specifies which DNP3 points the driver will read and/or write. Each row configures a single DNP3 point.

The following columns are required for each row:

- **Volttron Point Name** - The name used by the VOLTTRON platform and agents to refer to the point.
- **Group** - The point's DNP3 group number.
- **Index** - The point's index number within its DNP3 data type (which is derived from its DNP3 group number).
- **Scaling** - A factor by which to multiply point values.
- **Units** - Point value units.
- **Writable** - TRUE or FALSE, indicating whether the point can be written by the driver (FALSE = read-only).

Consult the **DNP3 data dictionary** for a point's Group and Index values. Point definitions in the data dictionary are by agreement between the DNP3 Outstation and Master. The VOLTTRON DNP3Agent loads the data dictionary of point definitions from the JSON file at "point_definitions_path" in the DNP3Agent's config file.

A sample data dictionary is available in `services/core/DNP3Agent/dnp3/mesa_points.config`.

Point definitions in the DNP3 driver's registry should look something like this:

Table 22: DNP3

Volttron Point Name	Group	Index	Scaling	Units	Writable
DCHD.WTgt	41	65	1.0	NA	FALSE
DCHD.WTgt-In	30	90	1.0	NA	TRUE
DCHD.WinTms	41	66	1.0	NA	FALSE
DCHD.RmpTms	41	67	1.0	NA	FALSE

A sample DNP3 driver registry configuration file is available in `services/core/MasterDriverAgent/example_configurations/dnp3.csv`.

1.35 Ecobee Driver

The Ecobee driver is an implementation of a *VOLTTRON driver framework* Interface. In this case, the Master Driver issues commands to the Ecobee driver to collect data from and send control signals to Ecobee's remote web API

Note: Reading the driver framework and driver configuration documentation prior to following this guide will help the user to understand drivers, driver communication, and driver configuration files.

This guide covers:

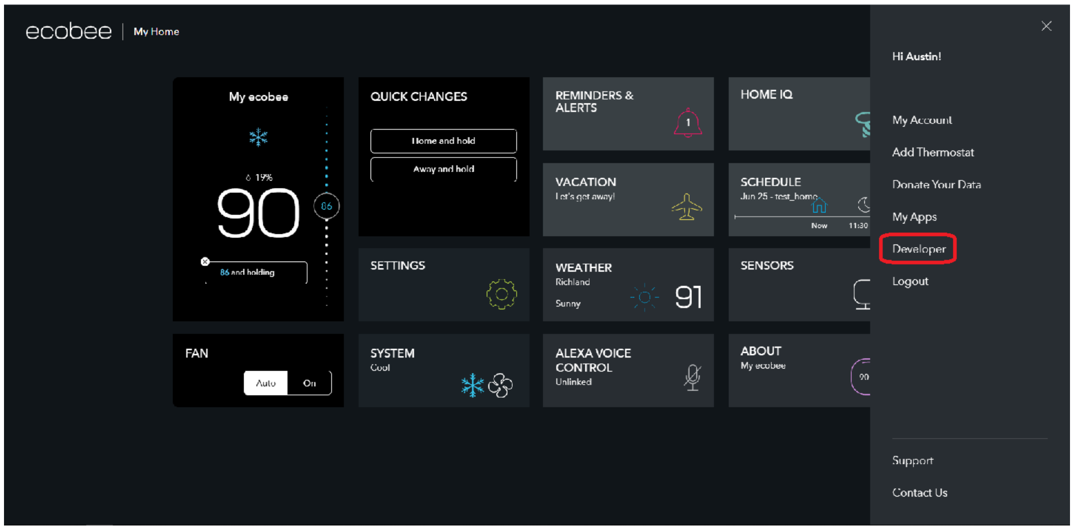
- Creating an Ecobee application via the web interface
- Creating an Ecobee driver configuration file, including finding the user's Ecobee API key and Ecobee thermostat serial number
- Creating an Ecobee registry configuration file

- Installing the Master Driver and loading Ecobee driver and registry configurations
- Starting the driver and viewing Ecobee data publishes

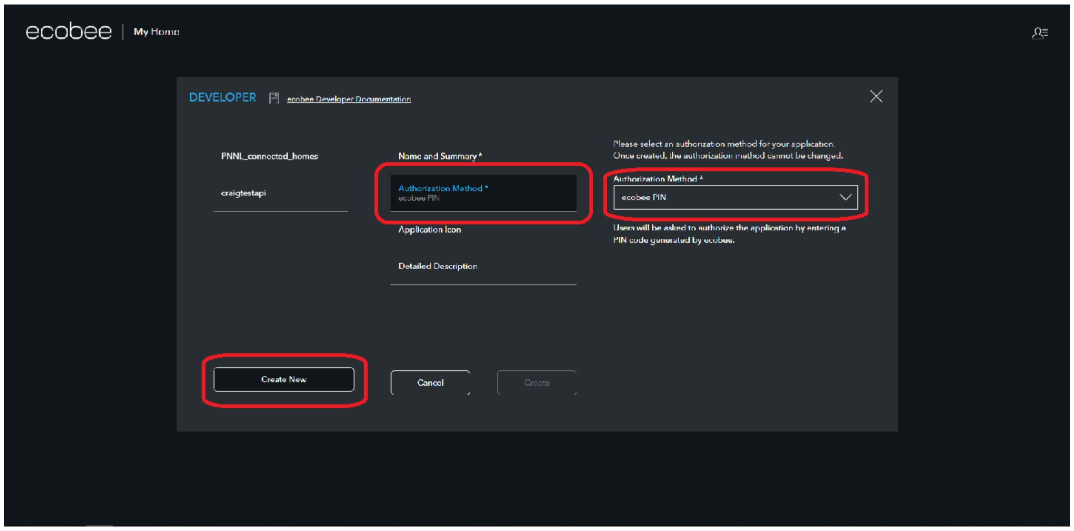
1.35.1 Ecobee Application

Connecting the Ecobee driver to the Ecobee API requires configuring your account with an Ecobee application.

1. Log into the Ecobee site
2. Click on the "hamburger" icon on the right to open the account menu, then click "Developer"



3. On the bottom-left corner of the screen that appears, click "Create New"



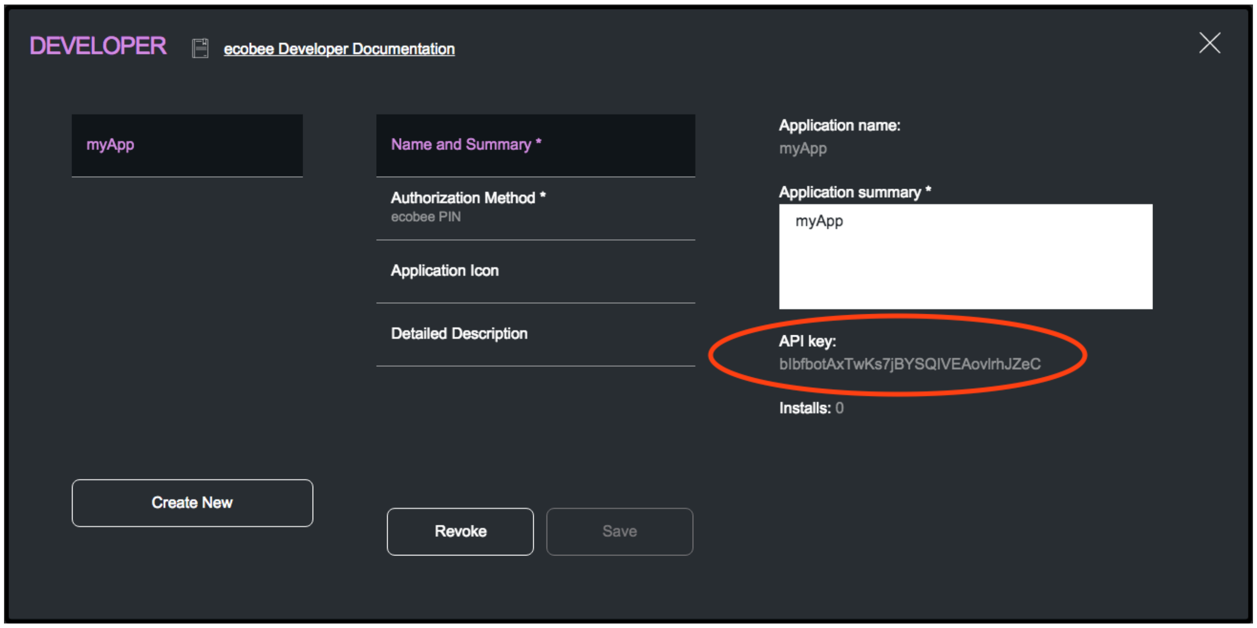
Authorization Method * ecobee PIN

Please select an authorization method for your application. Once created, the authorization method cannot be changed.

Authorization Method * ecobee PIN

Users will be asked to authorize the application by entering a PIN code generated by ecobee.

4. Fill out the name, summary, and description forms as desired. Click "Authorization Method" and from the drop-down that appears, select "ecobee PIN" (this will enable an extra layer of authentication to protect your account)
5. Record the API key for the Application from the Developer menu



Application name:
myApp

Application summary *
myApp

API key:
blbfbotAxTwKs7jBYSQIVEAovlrhJZeC

Installs: 0

Fig. 1: From Ecobee authenication docs

1.35.2 Configuration File

The Ecobee driver uses two configuration files, a driver configuration which sets the parameters of the behavior of the driver, and registry configuration which instructs the driver on how to interact with each point.

This is an example driver configuration:

```
{
    "driver_config": {
        "API_KEY": "abc123",
        "DEVICE_ID": 8675309
    },
    "driver_type": "ecobee",
    "registry_config":"config://campus/building/ecobee.csv",
    "interval": 180,
    "timezone": "UTC"
}
```

The driver configuration works as follows:

config field	description
driver_config	this section specifies values used by the driver agent during operation
API_KEY	This is the User's API key. This must be obtained by the user from the Ecobee web UI and provided in this part of the configuration. Notes on how to do this will be provided below.
DEVICE_ID	This is the device number of the Ecobee thermostat the driver is responsible for operating. This must be obtained by the user from the Ecobee web UI. Notes on how to do this will be provided below.
driver_type	This value should match the name of the python file which contains the interface class implementation for the Ecobee driver and should not change.
registry_config	This should a user specified path of the form "config://<path>. It is recommended to use the device topic string following "devices" with the file extension ("config://<campus>/<building?/ecobee.csv")to help the user keep track of configuration pairs in the store. This value must be used when storing the config (see installation step below).
interval	This should specify the time in seconds between publishes to the message bus by the Master Driver for the Ecobee driver (Note: the user can specify an interval for the Ecobee driver which is shorter than 180 seconds, however Ecobee API data is only updated at 180 second intervals, so old data will be published if a scrape occurs between updates.)
timezone	Timezone to use for publishing timestamps. This value should match the timezone from the Ecobee device

Note: Values for API_KEY and DEVICE_ID must be obtained by the user. DEVICE_ID should be added as an integer representation of the thermostat's serial number.

Getting API Key

Ecobee API keys require configuring an application using the Ecobee web UI. For more information on configuring an application and obtaining the API key, please refer to the *Ecobee Application* heading in this documentation.

Finding Device Identifier

To find your Ecobee thermostat's device identifier:

1. Log into the Ecobee customer portal
2. From the Home screen click "About My Ecobee"
3. The thermostat identifier is the serial number listed on the About screen

Registry Configuration

This file specifies how data is read from Ecobee API response data as well as how points are set via the Master Driver and actuator.

It is likely that more points may be added to obtain additional data, but barring implementation changes by Ecobee it is unlikely that the values in this configuration will need to change substantially, as most thermostats provide the same range of data in a similar format.

This is an example registry configuration:

Point Name	Volttron Point Name	Units	Type	Writable	Readable	Default Value	Notes
fanMinOnTime	fanMinOnTime	seconds	setting	True	True		
hvacMode	hvacMode	seconds	setting	True	True		
humidity	humidity	%	setting	False	True		
coolHoldTemp	coolHoldTemp	degF	hold	True	False		
heatHoldTemp	heatHoldTemp	degF	hold	True	False		
actualTemperature	actualTemperature	degF | hold		False	True		

This configuration works as follows:

config field	description
Point Name	Name of a point as it appears in Ecobee response data (example below)
Volttron Point Name	Name of a point as a user would like it to be displayed in data publishes to the message bus
Units	Unit of measurement specified by remote API
Type	The Ecobee driver registry configuration supports "setting" and "hold" register types, based on how the data is represented in Ecobee response data (example below)
Writable	Whether or not the point is able to be written to. This may be determined by what Ecobee allows, and by the operation of Ecobee's API (to set an Ecobee cool/heat hold, cool/HoldTemp is used, but to read other data points are used and therefore are not writable; this is a quirk of Ecobee's API)
Readable	Whether or not the point is able to be read as specified. This may be determined by what Ecobee allows, and by the operation of Ecobee's API (to set an Ecobee cool/heat hold, cool/HoldTemp is used, however the requested hold values are represented as desiredCool/Heat in Ecobee's response data; this is a quirk of Ecobee's API)
Default Value	Used to send device defaults to the Ecobee API, this is optional.
Notes	Any user specified notes, this is optional

An example registry configuration containing all points from the development device is available in the *examples/configurations/drivers/ecobee.csv* file in the VOLTTRON repository.

For additional explanation on the quirks of Ecobee's readable/writable points, visit: https://www.ecobee.com/home/developer/api/documentation/v1/functions/SetHold.shtml

1.35.3 Installation

The following instructions make up the minimal steps required to set up an instance of the Ecobee driver on the VOLTTRON platform and connect it to the Ecobee remote API:

1. Create a directory using the path $VOLTTRON_ROOT/configs and create two files, *ecobee.csv* and *ecobee.config*. Copy the registry config to the *ecobee.csv* file and the driver config to the *ecobee.config file*.

Modify the *API_KEY* and *DEVICE_ID* fields from the driver config with your own API key and device serial number.

2. If the platform has not been started:

```
./start-volttron
```

3. Be sure that the environment has been activated - you should see (volttron) next to <user>@<host> in your terminal window. To activate an environment, use the following command.

```
source env/bin/activate
```

4. Install a Master Driver if one is not yet installed

```
python scripts/install-agent.py --agent-source services/core/
↪MasterDriverAgent --config \
examples/configurations/drivers/master-driver.agent --tag platform.driver
```

5. Load the driver configuration into the configuration store ("vctl config list platform.driver" can be used to show installed configurations)

```
vctl config store platform.driver devices/campus/building/ecobee
↪$VOLTTRON_ROOT/configs/ecobee.config
```

6. Load the driver's registry configuration into the configuration store

```
vctl config store platform.driver campus/building/ecobee.csv $VOLTTRON_
↪ROOT/configs/ecobee.csv --csv
```

7. Start the master driver

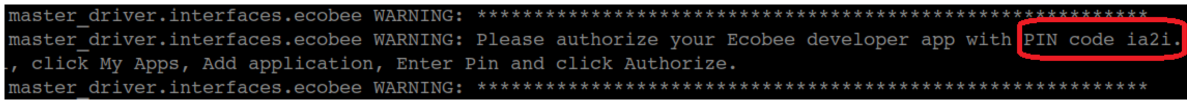
```
vctl start platform.driver
```

At this point, the master driver will start, configure the driver agent, and data should start to publish on the publish interval.

Note: If starting the driver for the first time, or if the authorization which is managed by the driver is out of date, the driver will perform some additional setup internally to authenticate the driver with the Ecobee API. This stage will require the user enter a pin provided in the *volttron.log* file to the Ecobee web UI. The Ecobee driver has a wait period of 60 seconds to allow users to enter the pin code into the Ecobee UI. Instructions for pin verification follow.

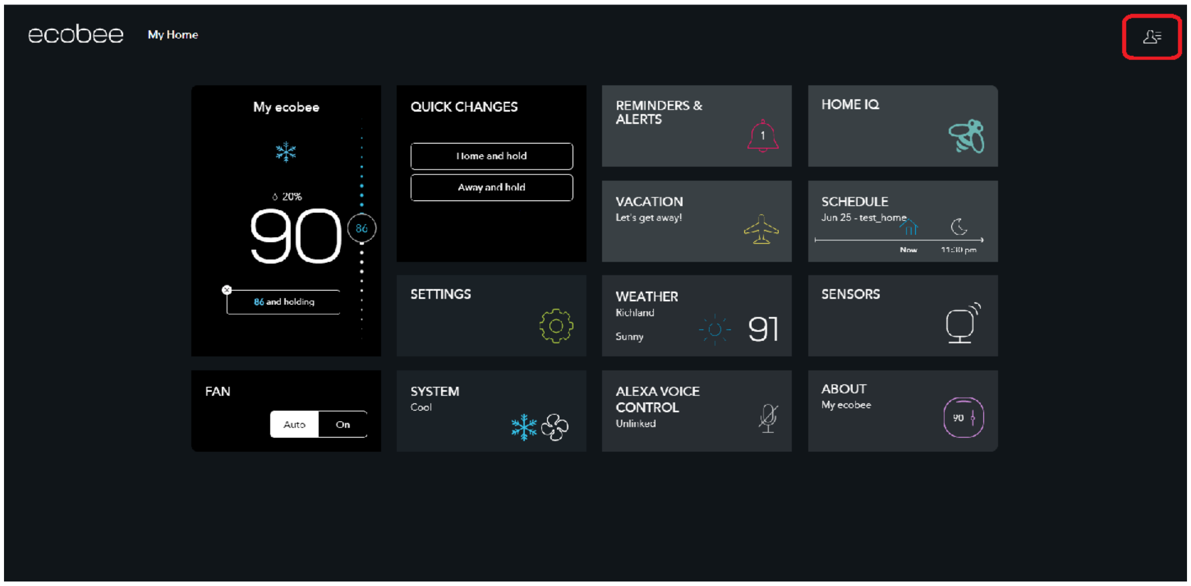
PIN Verification steps:

1. Obtain the pin from the VOLTTRON logs. The pin is a 4 character long string in the logs flanked by 2 rows of asterisks

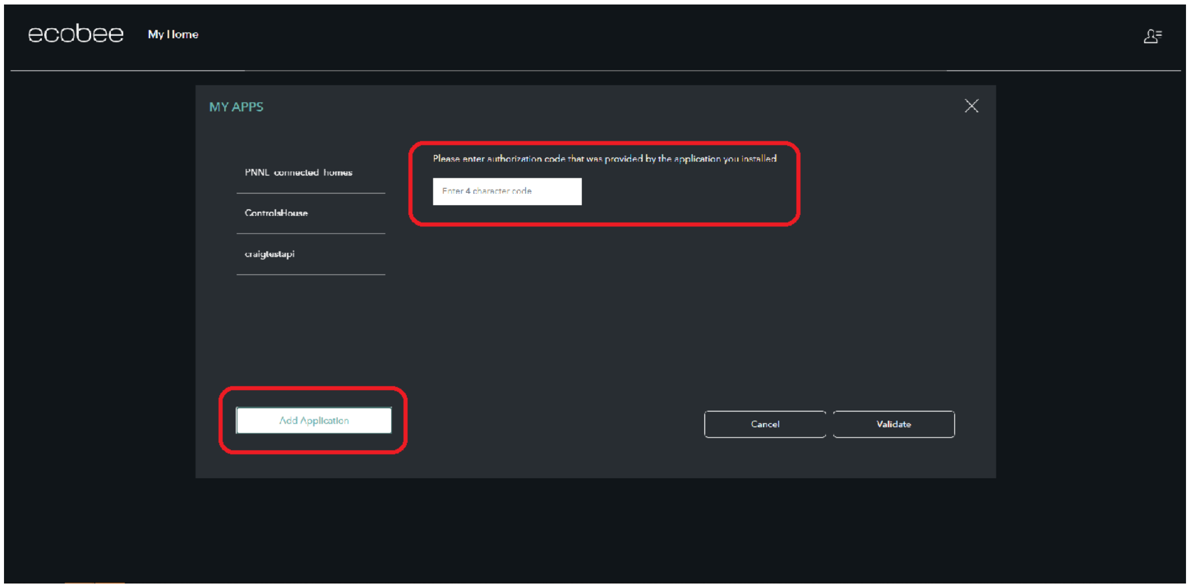


```
master_driver.interfaces.ecobee WARNING: **********************************************************
master_driver.interfaces.ecobee WARNING: Please authorize your Ecobee developer app with PIN code ia2i.
., click My Apps, Add application, Enter Pin and click Authorize.
master_driver.interfaces.ecobee WARNING: **********************************************************
```

2. Log into the Ecobee UI . After logging in, the customer dashboard will be brought up, which features a series of panels (where the serial number was found for device configuration) and a "hamburger" menu.



3. Add the application: Click the "hamburger" icon which will display a list of items in a panel that becomes visible on the right. Click "My Apps", then "Add application". A text form will appear, enter the pin provided in VOLTTRON logs here, then click "validate" and "add application."



This will complete the pin verification step.

1.35.4 Ecobee Driver Usage

At the configured interval, the master driver will publish a JSON object with data obtained from Ecobee based on the provided configuration files.

To view the publishes in the *volttron.log* file, install and start a ListenerAgent:

```
python scripts/install-agent.py -s examples/ListenerAgent
```

The following is an example publish:

```
'Status': [''],
  'Vacations': [{'coolHoldTemp': 780,
                 'coolRelativeTemp': 0,
                 'drRampUpTemp': 0,
                 'drRampUpTime': 3600,
                 'dutyCyclePercentage': 255,
                 'endDate': '2020-03-29',
                 'endTime': '08:00:00',
                 'fan': 'auto',
                 'fanMinOnTime': 0,
                 'heatHoldTemp': 660,
                 'heatRelativeTemp': 0,
                 'holdClimateRef': '',
                 'isCoolOff': False,
                 'isHeatOff': False,
                 'isOccupied': False,
                 'isOptional': True,
                 'isTemperatureAbsolute': True,
                 'isTemperatureRelative': False,
                 'linkRef': '',
                 'name': 'Skiing',
                 'occupiedSensorActive': False,
                 'running': False,
                 'startDate': '2020-03-15',
                 'startTime': '20:00:00',
                 'type': 'vacation',
                 'unoccupiedSensorActive': False,
                 'vent': 'off',
                 'ventilatorMinOnTime': 5}],
  'actualTemperature': 720,
  'desiredCool': 734,
  'desiredHeat': 707,
  'fanMinOnTime': 0,
  'humidity': '36',
  'hvacMode': 'off'},
 {'Programs': {'type': 'custom', 'tz': 'UTC', 'units': None},
  'Status': {'type': 'list', 'tz': 'UTC', 'units': None},
  'Vacations': {'type': 'custom', 'tz': 'UTC', 'units': None},
  'actualTemperature': {'type': 'integer', 'tz': 'UTC', 'units': 'degF'},
  'coolHoldTemp': {'type': 'integer', 'tz': 'UTC', 'units': 'degF'},
  'desiredCool': {'type': 'integer', 'tz': 'UTC', 'units': 'degF'},
  'desiredHeat': {'type': 'integer',S 'tz': 'UTC', 'units': 'degF'},
  'fanMinOnTime': {'type': 'integer', 'tz': 'UTC', 'units': 'seconds'},
  'heatHoldTemp': {'type': 'integer', 'tz': 'UTC', 'units': 'degF'},
  'humidity': {'type': 'integer', 'tz': 'UTC', 'units': '%'},
  'hvacMode': {'type': 'bool', 'tz': 'UTC', 'units': 'seconds'}}]
```

Individual points can be obtained via JSON RPC on the VOLTTRON Platform. In an agent:

```
self.vip.rpc.call("platform.driver", "get_point", <device topic>, <kwargs>)
```

1.35.5 Set_point Conventions

To set points using the Ecobee driver, it is recommended to use the actuator agent. Explanations of the actuation can be found in the VOLTTRON readthedocs and example agent code can be found in the CsvDriverAgent (examples/CSVDriver/CsvDriverAgent/agent.py in the VOLTTRON repository)

Setting values for Vacations and Programs requires understanding Vacation and Program object structure for Ecobee.

Documentation for Vacation structure can be found here: https://www.ecobee.com/home/developer/api/documentation/v1/functions/CreateVacation.shtml

Documentation for Program structure can be found here: https://www.ecobee.com/home/developer/api/examples/ex11.shtml

When using set_point for program, specifying a program structure will create a new program. Otherwise, if the user has not specified resume_all, Ecobee will resume the next program on the program stack. If resume_all, Ecobee will resume all programs on the program stack.

For all other points, the corresponding integer, string, boolean, etc. value may be sent.

Versioning

The Ecobee driver has been tested using the May 2019 API release as well as device firmware version 4.5.73.24

1.36 IEEE 2030.5 (SEP 2.0) Driver

Communicating with IEEE 2030.5 devices requires that the IEEE 2030.5 Agent is configured and running. All device communication happens through this agent. For information about the IEEE 2030.5 Agent, please see *IEEE 2030.5 Agent* docs.

1.36.1 Driver Config

There are two arguments for the *driver_config* section of the IEEE 2030.5 device configuration file:

- **sfdi** - Short-form device ID of the IEEE 2030.5 device.
- **ieee2030_5_agent_id** - ID of VOLTTRON's IEEE 2030.5 agent.

Here is a sample IEEE 2030.5 device configuration file:

```
{
    "driver_config": {
        "sfdi": "097935300833",
        "IEEE2030_5_agent_id": "iee2030_5agent"
    },
    "campus": "campus",
    "building": "building",
    "unit": "IEEE2030_5",
    "driver_type": "ieee2030_5",
    "registry_config": "config://ieee2030_5.csv",
    "interval": 15,
    "timezone": "US/Pacific",
    "heart_beat_point": "Heartbeat"
}
```

A sample IEEE 2030.5 driver configuration file can be found in the VOLTTRON repository in `services/core/MasterDriverAgent/example_configurations/test_ieee2030_5_1.config`.

1.36.2 Registry Configuration

For a description of IEEE 2030.5 registry values, see *IEEE 2030.5 DER Agent*.

A sample IEEE 2030.5 registry configuration file can be found in the VOLTTRON repository in `services/core/MasterDriverAgent/example_configurations/ieee2030_5.csv`.

View the *IEEE 2030.5 agent specification document* to learn more about IEEE 2030.5 and the IEEE 2030.5 agent and driver.

1.37 Modbus Driver

VOLTTRON's modbus driver supports the Modbus over TCP/IP protocol only. For Modbus RTU support, see VOLTTRON's *Modbus-TK driver <Modbus-TK-Driver>*.

About Modbus protocol

1.37.1 Modbus Driver Configuration

Requirements

The Modbus driver requires the pymodbus package. This package can be installed in an activated environment with:

```
pip install pymodbus
```

Alternatively this requirement can be installed using *bootstrap.py* with the `--drivers` option:

```
python3 bootstrap.py --drivers
```

Driver Configuration

There are three arguments for the *driver_config* section of the device configuration file:

- **device_address** - IP Address of the device.
- **port** - Port the device is listening on. Defaults to 502 which is the standard port for Modbus devices.
- **slave_id** - Slave ID of the device. Defaults to 0. Use 0 for no slave.

The remaining values are as follows:

Here is an example device configuration file:

```
{
    "driver_config": {"device_address": "10.1.1.2",
                      "port": 502,
                      "slave_id": 5},
    "driver_type": "modbus",
    "registry_config":"config://registry_configs/hvac.csv",
    "interval": 60,
    "timezone": "UTC",
    "heart_beat_point": "heartbeat"
}
```

A sample MODBUS configuration file can be found in the VOLTTRON repository in *examples/configurations/drivers/modbus.config*

Modbus Registry Configuration File

The registry configuration file is a CSV file. Each row configures a point on the device.

The following columns are required for each row:

- **Volttron Point Name** - The name by which the platform and agents running on the platform will refer to this point. For instance, if the Volttron Point Name is HeatCall1 (and using the example device configuration above) then an agent would use *pnnl/isb2/hvac1/HeatCall1* to refer to the point when using the RPC interface of the actuator agent.
- **Units** - Used for meta data when creating point information on the historian.
- **Modbus Register** - A string representing how to interpret the data register and how to read it from the device. The string takes two forms:
  - "BOOL" for coils and discrete inputs.
  - A format string for the Python struct module. See the Python3 Struct docs for full documentation. The supplied format string must only represent one value. See the documentation of your device to determine how to interpret the registers. Some Examples:
    - ">f" - A big endian 32-bit floating point number.
    - "<H" - A little endian 16-bit unsigned integer.
    - ">l" - A big endian 32-bit integer.
- **Writable** - Either *TRUE* or *FALSE*. Determines if the point can be written to. Only points labeled **TRUE** can be written to through the ActuatorAgent.
- **Point Address** - Modbus address of the point. Cannot include any offset value, it must be the exact value of the address.
- **Mixed Endian** - (Optional) Either *TRUE* or *FALSE*. For mixed endian values. This will reverse the order of the Modbus registers that make up this point before parsing the value or writing it out to the device. Has no effect on bit values.

The following column is optional:

- **Default Value** - The default value for the point. When the point is reverted by an agent it will change back to this value. If this value is missing it will revert to the last known value not set by an agent.

Any additional columns will be ignored. It is common practice to include a *Point Name* or *Reference Point Name* to include the device documentation's name for the point and *Notes* and *Unit Details* for additional information about a point.

The following is an example of a Modbus registry configuration file:

Table 23: Catalyst 371

Reference Point Name	Volttron Point Name	Units	Units Details	Modbus Register	Writable	Point Address	Default Value	Notes
CO2Sensor	ReturnAirCO2	PPM	0.00-2000.00	>f	FALSE	1001		CO2 Reading 0.00-2000.0 ppm
CO2Stpt	ReturnAirCO2Stpt	PPM	1000.00 (default)	>f	TRUE	1011	1000	Setpoint to enable demand control ventilation
Cool1Spd	CoolSupplyFanSpeed1	%	0.00 to 100.00 (75 default)	>f	TRUE	1005	75	Fan speed on cool 1 call
Cool2Spd	CoolSupplyFanSpeed2	%	0.00 to 100.00 (90 default)	>f	TRUE	1007	90	Fan speed on Cool2 Call
Damper	DamperSignal	%	0.00 - 100.00	>f	FALSE	1023		Output to the economizer damper
DaTemp	DischargeAirTemperature	F	(-)39.99 to 248.00	>f	FALSE	1009		Discharge air reading
ESMEconMin	ESMDamperMinPosition	%	0.00 to 100.00 (5 default)	>f	TRUE	1013	5	Minimum damper position during the energy savings mode
FanPower	SupplyFanPower	kW	0.00 to 100.00	>f	FALSE	1015		Fan power from drive
FanSpeed	SupplyFanSpeed	%	0.00 to 100.00	>f	FALSE	1003		Fan speed from drive
HeatCall1	HeatCall1	On / Off	on/off	BOOL	FALSE	1113		Status indicator of heating stage 1 need
HeartBeat	heartbeat	On / Off	on/off	BOOL	FALSE	1114		Status indicator of heating stage 2 need

A sample Modbus registry file can be found here or in the VOLTTRON repository in *examples/configurations/drivers/catalyst371.csv*

1.38 Modbus TK Driver

VOLTTRON's Modbus-TK driver, built on the Python Modbus-TK library, is an alternative to the original VOLTTRON modbus driver. Unlike the original modbus driver, the Modbus-TK driver supports Modbus RTU as well as Modbus over TCP/IP.

About Modbus protocol

The Modbus-TK driver introduces a map library and configuration builder, intended as a way to streamline configuration file creation and maintenance.

Warning: Currently the modbus_tk library is not able to make connections from 2 masters on one host to 2 slaves on one host - this will will prevent a single platform from being able to communicate to 2 slaves on IP as each instance of a Modbus_Tk driver creates a new Modbus master. Issue on Modbus_Tk Github.

1.38.1 Modbus-TK Driver Configuration

The Modbus-TK driver is mostly backward-compatible with the parameter definitions in the original Modbus driver's configuration (.config and .csv files). If the config file's parameter names use the Modbus driver's name conventions, they are translated to the Modbus-TK name conventions, e.g. a Modbus CSV file's `Point Address` is interpreted as a Modbus-TK "Address". Backward-compatibility exceptions are:

- If the config file has no `port`, the default is 0, not 502.
- If the config file has no `slave_id`, the default is 1, not 0.

Requirements

The Modbus-TK driver requires the modbus-tk package. This package can be installed in an activated environment with:

```
pip install modbus-tk
```

Alternatively this requirement can be installed using *bootstrap.py* with the `--drivers` option:

```
python3 bootstrap.py --drivers
```

Driver Configuration

The `driver_config` section of a Modbus-TK device configuration file supports a variety of parameter definitions, but only **device_address** is required:

- `name` (Optional) - Name of the device. Defaults to "UNKNOWN".
- `device_type` (Optional) - Name of the device type. Defaults to "UNKNOWN".
- `device_address` (Required) - IP Address of the device.
- `port` (Optional) - Port the device is listening on. Defaults to 0 (no port). Use port 0 for RTU transport.
- `slave_id` (Optional) - Slave ID of the device. Defaults to 1. Use ID 0 for no slave.
- `baudrate` (Optional) - Serial (RTU) baud rate. Defaults to 9600.
- `bytesize` (Optional) - Serial (RTU) byte size: 5, 6, 7, or 8. Defaults to 8.
- `parity` (Optional) - Serial (RTU) parity: none, even, odd, mark, or space. Defaults to none.
- `stopbits` (Optional) - Serial (RTU) stop bits: 1, 1.5, or 2. Defaults to 1.
- `xonxoff` (Optional) - Serial (RTU) flow control: 0 or 1. Defaults to 0.
- `addressing` (Optional) - Data address table: offset, offset_plus, or address. Defaults to offset.
  - address: The exact value of the address without any offset value.
  - offset: The value of the address plus the offset value.
  - offset_plus: The value of the address plus the offset value plus one.

  - : If an offset value is to be added, it is determined based on a point's properties in the CSV file:
    * Type=bool, Writable=TRUE: 0
    * Type=bool, Writable=FALSE: 10000
    * Type!=bool, Writable=TRUE: 30000
    * Type!=bool, Writable=FALSE: 40000

- `endian` (Optional) - Byte order: big or little. Defaults to big.
- `write_multiple_registers` (Optional) - Write multiple coils or registers at a time. Defaults to true.
  - If write_multiple_registers is set to false, only register types unsigned short (uint16) and boolean (bool) are supported. The exception raised during the configure process.
- `register_map` (Optional) - Register map csv of unchanged register variables. Defaults to registry_config csv.

Sample Modbus-TK configuration files are checked into the VOLTTRON repository in `services/core/MasterDriverAgent/master_driver/interfaces/modbus_tk/maps`.

Here is a sample TCP/IP Modbus-TK device configuration:

```
{
    "driver_config": {
        "device_address": "10.1.1.2",
        "port": "5020",
        "register_map": "config://modbus_tk_test_map.csv"
    },
    "driver_type": "modbus_tk",
    "registry_config": "config://modbus_tk_test.csv",
    "interval": 60,
    "timezone": "UTC",
    "heart_beat_point": "heartbeat"
}
```

Here is a sample RTU Modbus-TK device configuration, using all default settings:

```
{
    "driver_config": {
        "device_address": "/dev/tty.usbserial-AL00IEEY",
        "register_map": "config://modbus_tk_test_map.csv"
    },
    "driver_type": "modbus_tk",
    "registry_config":"config://modbus_tk_test.csv",
    "interval": 60,
    "timezone": "UTC",
    "heart_beat_point": "heartbeat"
}
```

Here is a sample RTU Modbus-TK device configuration, with completely-specified settings:

```
{
    "driver_config": {
        "device_address": "/dev/tty.usbserial-AL00IEEY",
        "port": 0,
        "slave_id": 2,
        "name": "watts_on",
        "baudrate": 115200,
```

(continues on next page)

(continued from previous page)

```
        "bytesize": 8,
        "parity": "none",
        "stopbits": 1,
        "xonxoff": 0,
        "addressing": "offset",
        "endian": "big",
        "write_multiple_registers": true,
        "register_map": "config://watts_on_map.csv"
    },
    "driver_type": "modbus_tk",
    "registry_config": "config://watts_on.csv",
    "interval": 120,
    "timezone": "UTC"
}
```

1.38.2 Modbus-TK Register Map CSV File

Modbus TK requires an additional registry configuration file compared to the paradigm of most other drivers. The registry map file is an analogue to the typical registry configuration file. The *registry configuration file* is a simple file which maps device point names to user specified point names.

The registry map file is a CSV file. Each row configures a register definition on the device.

- `Register Name` (Required) - The field name in the modbus client. This field is distinct and unchangeable.
- `Address` (Required) - The point's modbus address. The `addressing` option in the driver configuration controls whether this is interpreted as an exact address or an offset.
- `Type` (Required) - The point's data type: bool, string[length], float, int16, int32, int64, uint16, uint32, or uint64.
- `Units` (Optional) - Used for metadata when creating point information on a historian. Default is an empty string.
- `Writable` (Optional) - TRUE/FALSE. Only points for which Writable=TRUE can be updated by a VOLTTRON agent. Default is FALSE.
- `Default Value` (Optional) - The point's default value. If it is reverted by an agent, it changes back to this value. If this value is missing, it will revert to the last known value not set by an agent.
- `Transform` (Optional) - Scaling algorithm: scale(multiplier), scale_int(multiplier), scale_reg(register_name), scale_reg_power10(register_name), scale_decimal_int_signed(multiplier), mod10k(reverse), mod10k64(reverse), mod10k48(reveres) or none. Default is an empty string.
- `Table` (Optional) - Standard modbus table name defining how information is stored in slave device. There are 4 different tables:
  - discrete_output_coils: read/write coil numbers 1-9999
  - discrete_input_contacts: read only coil numbers 10001-19999
  - analog_input_registers: read only register numbers 30001-39999
  - analog_output_holding_registers: read/write register numbers 40001-49999

  If this field is empty, the modbus table will be defined by **type** and **writable** fields. By that, when user sets read only for read/write coils/registers or sets read/write for read only coils/registers, it will select wrong table, and therefore raise exception.

- `Mixed Endian` (Optional) - TRUE/FALSE. If Mixed Endian is set to TRUE, the order of the Modbus registers will be reversed before parsing the value or writing it out to the device. By setting mixed endian, transform must be None (no op). Defaults to FALSE.
- `Description` (Optional) - Additional information about the point. Default is an empty string.

Any additional columns are ignored.

Sample Modbus-TK registry map CSV files are checked into the VOLTTRON repository in `services/core/MasterDriverAgent/master_driver/interfaces/modbus_tk/maps`.

Here is a sample Modbus-TK registry map:

Register Name	Address	Type	Units	Writable	Default Value	Transform	Table
unsigned_short	0	uint16	None	TRUE	0	scale(10)	analog_output_holding_registers
unsigned_int	1	uint32	None	TRUE	0	scale(10)	analog_output_holding_registers
unsigned_long	3	uint64	None	TRUE	0	scale(10)	analog_output_holding_registers
sample_short	7	int16	None	TRUE	0	scale(10)	analog_output_holding_registers
sample_int	8	int32	None	TRUE	0	scale(10)	analog_output_holding_registers
sample_float	10	float	None	TRUE	0.0	scale(10)	analog_output_holding_registers
sample_long	12	int64	None	TRUE	0	scale(10)	analog_output_holding_registers
sample_bool	16	bool	None	TRUE	False		analog_output_holding_registers
sample_str	17	string[12]	None	TRUE	hello world!		analog_output_holding_registers

1.38.3 Modbus-TK Registry Configuration

The registry configuration file is a CSV file. Each row configures a point on the device.

- `Volttron Point Name` (Required) - The name by which the platform and agents refer to the point. For instance, if the Volttron Point Name is HeatCall1, then an agent would use `my_campus/building2/hvac1/HeatCall1` to refer to the point when using the RPC interface of the actuator agent.
- `Register Name` (Required) - The field name in the modbus client. It must be matched with the field name from `register_map`.

Any additional columns will override the existed fields from `register_map`.

Sample Modbus-TK registry CSV files are checked into the VOLTTRON repository in `services/core/MasterDriverAgent/master_driver/interfaces/modbus_tk/maps`.

Here is a sample Modbus-TK registry configuration with defined `register_map`:

Volttron Point Name	Register Name
unsigned short	unsigned_short
unsigned int	unsigned_int
unsigned long	unsigned_long
sample short	sample_short
sample int	sample_int
sample float	sample_float
sample long	sample_long
sample bool	sample_bool
sample str	sample_str

1.38.4 Modbus-TK Driver Maps Repository

To help facilitate the creation of VOLTTRON device configuration entries (.config files) for Modbus-TK devices, a library of device type definitions is now maintained in `services/core/MasterDriverAgent/master_driver/interfaces/modbus_tk/maps/maps.yaml`. A command-line tool (described below under `MODBUS TK Config Command Tool`) uses the contents of `maps.yaml` while generating `.config` files.

Each device type definition in `maps.yaml` consists of the following properties:

- `name` (Required) - Name of the device type (see the driver_config parameters).
- `file` (Required) - The name of the CSV file that defines all of the device type's supported points, e.g. watts_on.csv.
- `description` (Optional) - A description of the device type.
- `addressing` (Optional) - Data address type: offset, offset_plus, or address (see the driver_config parameters).
- `endian` (Optional) - Byte order: big or little (see the driver_config parameters).
- `write_multiple_registers` (Optional) - Write multiple registers at a time. Defaults to true.

A device type definition is a template for a device configuration. Some additional data must be supplied when a specific device's configuration is generated. In particular, the device_address must be supplied.

A sample `maps.yml` file is checked into the VOLTTRON repository in `services/core/MasterDriverAgent/master_driver/interfaces/modbus_tk/maps/maps.yaml`.

Here is a sample `maps.yaml` file:

```yaml
- name: modbus_tk_test
  description: Example of reading selected points for Modbus-TK driver testing
  file: modbus_tk_test_map.csv
  addressing: offset
  endian: little
  write_multiple_registers: true
- name: watts_on
  description: Read selected points from Elkor WattsOn meter
  file: watts_on_map.csv
  addressing: offset
- name: ion6200
  description: ION 6200 meter
  file: ion6200_map.csv
- name: ion8600
  description: ION 8600 meter
  file: ion8600_map.csv
```

1.38.5 Modbus-TK Config Command Tool

`config_cmd.py` is a command-line tool for creating and maintaining VOLTTRON driver configurations. The tool runs from the command line:

```
$ cd services/core/MasterDriverAgent/master_driver/interfaces/modbus_tk/maps
$ python config_cmd.py
```

`config_cmd.py` supports the following commands:

- `help` - List all commands.
- `quit` - Quit the command-line tool.
- `list_directories` - List all setup directories, with an option to edit their paths.
  - By default, all directories are in the VOLTTRON repository in `services/core/MasterDriverAgent/master_driver/interfaces/modbus_tk/maps`.
  - It is important to use the correct directories when adding/editing device types and driver configs, and when loading configurations into VOLTTRON.
    - map_dir: directory in which `maps.yaml` is stored.
    - config_dir: directory in which driver config files are stored.
    - csv_dir: directory in which registry config CSV files are stored.
- `edit_directories` - Add/Edit map directory, driver config directory, and/or CSV config directory. Press <Enter> if no change is needed. Exits if the directory does not exist.
- `list_device_type_description` - List all device type descriptions in `maps.yaml`. Option to edit device type descriptions.
- `list_all_device_types` - List all device type information in `maps.yaml`. Option to add more device types.
- `device_type` - List information for a selected device type. Option to select another device type.
- `add_device_type` - Add a device type to `maps.yaml`. Option to add more than one device type. Each device type includes its name, CSV file, description, addressing, and endian, as explained in `MODBUS-TK Driver Maps`. If an invalid value is entered for addressing or endian, the default value is used instead.
- `edit_device_type` - Edit an existing device type. If an invalid value is entered for addressing or endian, the previous value is left unchanged.
- `list_drivers` - List all driver config names in `config_dir`.
- `driver_config <driver_name>` - Get a driver config from `config_dir`. Option to select the driver if no driver is found with that name.
- `add_driver_config <driver_name>` - Add/Edit `<config_dir>/<driver name>.config`. Option to select the driver if no driver is found with that name. Press <Enter> to exit.
- `load_volttron` - Load a driver config and CSV into VOLTTRON. Option to add the config or CSV file to config_dir or to csv_dir. VOLTTRON must be running when this command is used.
- `delete_volttron_config` - Delete a driver config from VOLTTRON. VOLTTRON must be running when this command is used.
- `delete_volttron_csv` - Delete a registry csv config from VOLTTRON. VOLTTRON must be running when this command is used.

The `config_cmd.py` module is checked into the VOLTTRON repository as `services/core/MasterDriverAgent/master_driver/interfaces/modbus_tk/config_cmd.py`.

1.39 Obix Driver

1.39.1 Obix Driver Configuration

VOLTTRON's uses Obix's restful interface to facilitate communication.

This driver does *not* handle reading data from the history section of the interface. If the user wants data published from the management systems historical data use the *Obix History* agent.

Driver Configuration

There are three arguments for the `driver_config` section of the device configuration file:

- `url` - URL of the Obix remote API interface
- `username` - User's username for the Obix remote API
- `password` - Users' password corresponding to the username

Here is an example device configuration file:

```
{
    "driver_config": {"url": "http://example.com/obix/config/Drivers/Obix/exports/",
                      "username": "username",
                      "password": "password"},
    "driver_type": "obix",
    "registry_config":"config://registry_configs/obix.csv",
    "interval":     30,
    "timezone": "UTC"
}
```

A sample Obix configuration file can be found in the VOLTTRON repository in *examples/configurations/drivers/obix.config*

Obix Registry Configuration File

The registry configuration file is a CSV file. Each row configures a point on the device.

The following columns are required for each row:

- **Volttron Point Name** - The name by which the platform and agents running on the platform will refer to this point. For instance, if the Volttron Point Name is HeatCall1 then an agent would use *<device topic>/HeatCall1* to refer to the point when using the RPC interface of the actuator agent.
- **Obix Point Name** - Name of the point on the Obix interface. Escaping of spaces and dashes for use with the interface is handled internally.
- **Obix Type** - One of *bool*, *int*, or *real*
- **Units** - Used for meta data when creating point information on the historian.
- **Writable** - Either *TRUE* or *FALSE*. Determines if the point can be written to. Only points labeled **TRUE** can be written to through the ActuatorAgent. This can be used to protect points that should not be accessed by the platform.

The following column is optional:

- **Default Value** - The default value for the point. When the point is reverted by an agent it will change back to this value. If this value is missing it will revert to the last known value not set by an agent.

Any additional columns will be ignored. It is common practice to include a *Point Name* or *Reference Point Name* to include the device documentation's name for the point and *Notes* and *Unit Details* for additional information about a point.

The following is an example of a Obix registry configuration file:

Table 24: Obix

Volttron Point Name	Obix Point Name	Obix Type	Units	Writable	Notes
CostEL	CostEL	real	dollar	FALSE	Precision: 2
CostELBB	CostELBB	real	dollar	FALSE	Precision: 2
CDHEnergyHeartbeat	CDHEnergyHeartbeat	real	null	FALSE	
ThermalFollowing	ThermalFollowing	bool		FALSE	
CDHTestThermFollow	CDHTestThermFollow	bool		FALSE	
CollegeModeFromCDH	CollegeModeFromCDH	real	null	FALSE	Precision: 0, Min: 3.0, Max: 3.0
HospitalModeFromCDH	HospitalModeFrom-CDH	real	null	FALSE	Precision: 0, Min: 3.0, Max: 3.0
HomeModeFromCDH	HomeModeFromCDH	real	null	FALSE	Precision: 0, Min: 3.0, Max: 3.0
CostNG	CostNG	real	null	FALSE	Precision: 2
CollegeBaseload-SPFromCDH	CollegeBaseload-SPFromCDH	real	kilo-watt	FALSE	Precision: 0
CollegeImportSPFrom-CDH	CollegeImportSPFrom-CDH	real	kilo-watt	FALSE	Precision: 0
HospitalImportSPFrom-CDH	HospitalImportSPFrom-CDH	real	kilo-watt	FALSE	Precision: 0
HospitalBaseload-SPFromCDH	HospitalBaseload-SPFromCDH	real	kilo-watt	FALSE	Precision: 0
HomeImportSPFrom-CDH	HomeImportSPFrom-CDH	real	kilo-watt	FALSE	Precision: 0
ThermalFol-lowingAlarm	ThermalFol-lowingAlarm	bool		FALSE	

A sample Obix configuration can be found in the VOLTTRON repository in *examples/configurations/drivers/obix.csv*

Automatic Obix Configuration File Creation

A script that will automatically create both a device and register configuration file for a site is located in the repository at *scripts/obix/get_obix_driver_config.py*.

The utility is invoked with the command:

```
python get_obix_driver_config.py <url> <registry_file> <driver_file> -u <username> -p
↪<password>
```

If either the *registry_file* or *driver_file* is omitted the script will output those files to stdout.

If either the username or password arguments are left out the script will ask for them on the command line before proceeding.

The registry file produced by this script assumes that the *Volttron Point Name* and the *Obix Point Name* have the same value. Also, it is assumed that all points should be read only. Users are expected to fix this as appropriate.

1.40 The Energy Detective Meter Driver

The TED-Pro is an energy monitoring system that can measure energy consumption of multiple mains and supports sub-metering of individual circuits. This driver connects to a TED Pro Energy Control Center (ECC) and can collect information from multiple Measuring Transmitting Units (MTUs) and Spyder sub-metering devices connected to the ECC.

1.40.1 Configuration

The TED Pro device interface is configured as follows. You'll need the ip address or hostname of the ECC on a network segment accessible from the VOLTTRON instance, if configured to use a port other than 80, you can provide it as shown below, following a colon after the host address.

```
{
    "driver_type": "ted_meter",
    "driver_config": {
        "device_address": "192.168.1.100:8080",
        "username": "username",
        "password": "password",
        "scrape_spyder": true,
        "track_totalizers": true
    }
}
```

Parameters

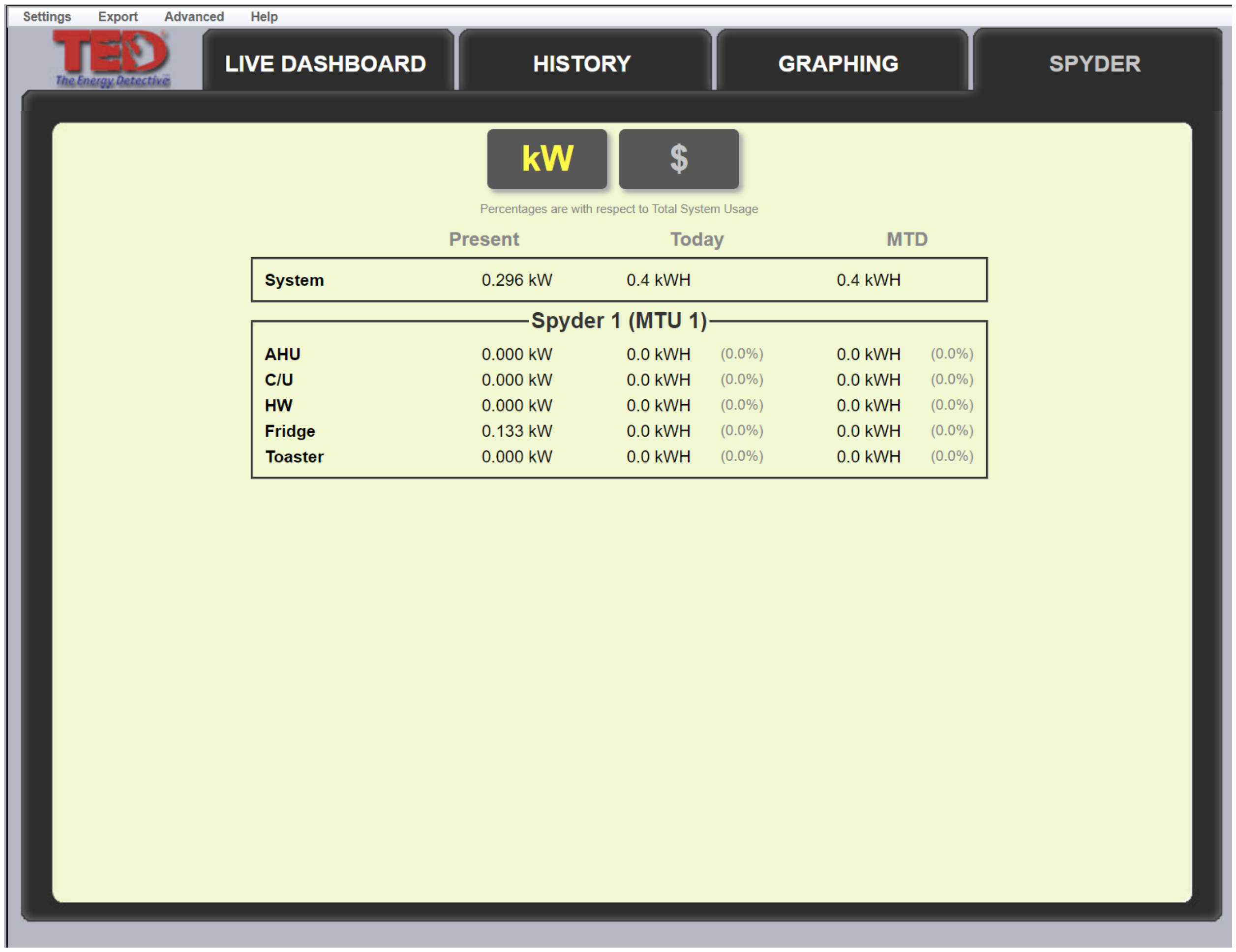
- **username** - Username if the TED Pro is configured with Basic Authentication
- **password** - Password if the TED Pro is configured with Basic Authentication
- **device_address** - Hostname or IP address of the TED Pro ECC, a non-standard port can be included if needed
- **scrape_spyder** - Default true, enables or disables collection of the sub-metering data from spyder devices connected to the TED Pro
- **track_totalizers** - Default true, enables or disables tracking of lifetime totals in the VOLTTRON Driver

Note: The TED Pro does not expose its internal lifetime "totalized" metering, instead offering month to date (MTD) and daily totals (TDY). Using the "track_totalizers" setting, the ted-meter driver will attempt to maintain monotonically increasing lifetime totalizers. To do so, it must retain state regarding the running total and the last read value. The driver makes use of the VOLTTRON Config subsystem to store this state. To reset these totals, delete the 1state/ted_meter/<device_path>1 config from the master driver config store and restart the master driver.

Note: This driver does not make use of the registry config. Because it is able to determine the configuration of the TED Pro Device via the API, it simply creates registers for each data source on the TED Pro

Note: This driver is internally aware of the appropriate HayStack Tags for its registers, however, the Master Driver makes no provision for publishing those tags during a scrape. Therefore, integration of the tagging data is left to the end user.

Examples



Percentages are with respect to Total System Usage

	Present	Today		MTD	
System	0.296 kW	0.4 kWH		0.4 kWH	
Spyder 1 (MTU 1)					
AHU	0.000 kW	0.0 kWH	(0.0%)	0.0 kWH	(0.0%)
C/U	0.000 kW	0.0 kWH	(0.0%)	0.0 kWH	(0.0%)
HW	0.000 kW	0.0 kWH	(0.0%)	0.0 kWH	(0.0%)
Fridge	0.133 kW	0.0 kWH	(0.0%)	0.0 kWH	(0.0%)
Toaster	0.000 kW	0.0 kWH	(0.0%)	0.0 kWH	(0.0%)

The above configuration in the TED will result in the following scrape from the ted-meter driver on the message bus:

```
[
    {
        'mtu-1/load_kva': 0.271,
        'mtu-1/load_kw': 0.203,
        'mtu-1/phase_angle': 195,
        'mtu-1/phase_current-a': '0',
        'mtu-1/phase_current-b': '0',
        'mtu-1/phase_current-c': '0',
        'mtu-1/phase_voltage-a': '0',
        'mtu-1/phase_voltage-b': '0',
        'mtu-1/phase_voltage-c': '0',
        'mtu-1/power_factor': 0.749,
        'mtu-1/voltage': 121.30000000000001,
        'spyder-1/AHU/load': 0.0,
        'spyder-1/AHU/mtd': 0.0,
        'spyder-1/AHU/mtd_totalized': 0.0,
        'spyder-1/C/U/load': 0.0,
        'spyder-1/C/U/mtd': 0.0,
        'spyder-1/C/U/mtd_totalized': 0.0,
        'spyder-1/Fridge/load': 0.0,
```

(continues on next page)

(continued from previous page)

```
        'spyder-1/Fridge/mtd': 0.056,
        'spyder-1/Fridge/mtd_totalized': 0.056,
        'spyder-1/HW/load': 0.0,
        'spyder-1/HW/mtd': 0.14400000000000002,
        'spyder-1/HW/mtd_totalized': 0.14400000000000002,
        'spyder-1/Toaster/load': 0.0,
        'spyder-1/Toaster/mtd': 0.24,
        'spyder-1/Toaster/mtd_totalized': 0.24,
        'system/mtd': 0.652,
        'system/mtd_totalized': 0.652
    },
    {
        'mtu-1/load_kva': {'type': 'integer', 'tz': u'', 'units': 'kVA'},
        'mtu-1/load_kw': {'type': 'integer', 'tz': u'', 'units': 'kW'},
        'mtu-1/phase_angle': {'type': 'integer', 'tz': u'', 'units': 'degrees'},
        'mtu-1/phase_current-a': {'type': 'integer', 'tz': u'', 'units': 'Amps'},
        'mtu-1/phase_current-b': {'type': 'integer', 'tz': u'', 'units': 'Amps'},
        'mtu-1/phase_current-c': {'type': 'integer', 'tz': u'', 'units': 'Amps'},
        'mtu-1/phase_voltage-a': {'type': 'integer', 'tz': u'', 'units': 'Volts'},
        'mtu-1/phase_voltage-b': {'type': 'integer', 'tz': u'', 'units': 'Volts'},
        'mtu-1/phase_voltage-c': {'type': 'integer', 'tz': u'', 'units': 'Volts'},
        'mtu-1/power_factor': {'type': 'integer', 'tz': u'', 'units': 'ratio'},
        'mtu-1/voltage': {'type': 'integer', 'tz': u'', 'units': 'Volts'},
        'spyder-1/AHU/load': {'type': 'integer', 'tz': u'', 'units': 'kW'},
        'spyder-1/AHU/mtd': {'type': 'integer', 'tz': u'', 'units': 'kWh'},
        'spyder-1/AHU/mtd_totalized': {'type': 'integer', 'tz': u'', 'units': 'kWh'},
        'spyder-1/C/U/load': {'type': 'integer', 'tz': u'', 'units': 'kW'},
        'spyder-1/C/U/mtd': {'type': 'integer', 'tz': u'', 'units': 'kWh'},
        'spyder-1/C/U/mtd_totalized': {'type': 'integer', 'tz': u'', 'units': 'kWh'},
        'spyder-1/Fridge/load': {'type': 'integer', 'tz': u'', 'units': 'kW'},
        'spyder-1/Fridge/mtd': {'type': 'integer', 'tz': u'', 'units': 'kWh'},
        'spyder-1/Fridge/mtd_totalized': {'type': 'integer', 'tz': u'', 'units': 'kWh
↪'},
        'spyder-1/HW/load': {'type': 'integer', 'tz': u'', 'units': 'kW'},
        'spyder-1/HW/mtd': {'type': 'integer', 'tz': u'', 'units': 'kWh'},
        'spyder-1/HW/mtd_totalized': {'type': 'integer', 'tz': u'', 'units': 'kWh'},
        'spyder-1/Toaster/load': {'type': 'integer', 'tz': u'', 'units': 'kW'},
        'spyder-1/Toaster/mtd': {'type': 'integer', 'tz': u'', 'units': 'kWh'},
        'spyder-1/Toaster/mtd_totalized': {'type': 'integer', 'tz': u'', 'units': 'kWh
↪'},
        'system/mtd': {'type': 'integer', 'tz': u'', 'units': 'kWh'},
        'system/mtd_totalized': {'type': 'integer', 'tz': u'', 'units': 'kWh'}
    }
]
```

1.41 Message Bus

The VOLTTRON message bus is the mechanism responsible for enabling communication between agents, drivers, and platform instances. The message bus supports communication using the *Publish/Subscribe Paradigm* and *JSON RPC*. Currently VOLTTRON may be configured to use either Zero MQ or RabbitMQ messaging software to perform messaging.

To standardize message bus communication, VOLTTRON implements VIP - VOLTTRON Interconnect Protocol. VIP defines patterns for pub/sub communication as well as JSON-RPC, and allows for the creation of agent communication subsystems.

For more information on messaging, VIP, multi-platform communication and more, please explore the message bus documentation linked below:

1.41.1 Messaging and Topics

Introduction

Agents in VOLTTRON™ communicate with each other using a publish/subscribe mechanism built on the Zero MQ or RabbitMQ Python libraries. This allows for great flexibility as topics can be created dynamically and the messages sent can be any format as long as the sender and receiver understand it. An agent with data to share publishes to a topic, then any agents interested in that data subscribe to that topic.

While this flexibility is powerful, it also could also lead to confusion if some standard is not followed. The current conventions for communicating in the VOLTTRON are:

- Topics and subtopics follow the format: `topic/subtopic/subtopic`
- Subscribers can subscribe to any and all levels. Subscriptions to *topic* will include messages for the base topic and all subtopics. Subscriptions to `topic/subtopic1` will only receive messages for that subtopic and any children subtopics. Subscriptions to empty string ("") will receive ALL messages. This is not recommended.

Agents should set the *From* header. This will allow agents to filter on the *To* message sent back.

Topics

In VOLTTRON

- **alerts** - Base topic for alerts published by agents and subsystems, such as agent health alerts
- **analysis** - Base topic for analytics being used with building data
- **config** - Base topic for managing agent configuration
- **devices** - Base topic for data being published by drivers
- **datalogger** - Base topic for agents wishing to record time series data
- **heartbeat** - Topic for publishing periodic "heartbeat" or "keep-alive"
- **market** - Base topics for market agent communication
- **record** - Base topic for agents to record data in an arbitrary format
- **weather** - Base topic for polling publishes of weather service agents

Note: Other more specific topics may exist for specific agents or purposes. Please review the documentation for the specific feature for more information.

Controller Agent Topics

See the documentation for the *Actuator Agent*.

1.41.2 VOLTTRON™ Interconnect Protocol

This document specifies VIP, the VOLTTRON™ Interconnect Protocol. The use case for VIP is to provide communications between *agents*, *controllers*, *services*, and the supervisory *platform* in an abstract fashion so that additional protocols can be built and used above VIP. VIP defines how *peers* connect to the *router* and the messages they exchange.

- Name: github.com/VOLTTRON/volttron/wiki/VOLTTRON-Interconnect-Protocol
- Editor: Brandon Carpenter <brandon (dot) carpenter (at) pnnl (dot) gov>
- State: draft
- See also: ZeroMQ, ZMTP, CurveZMQ, ZAP

Remote Procedure Calls

Remote procedure calls (RPC) is a feature of VOLTTRON Interconnect Protocol *VIP*. VIP includes the ability to create new point-to-point protocols, called subsystems, enabling the implementation of JSON-RPC 2.0. This provides a simple method for agent authors to write methods and expose or export them to other agents, making request-reply or notify communications patterns as simple as writing and calling methods.

Exporting Methods

The `export()` method, defined on the RPC subsystem class, is used to mark a method as remotely accessible. This `export()` method has a dual use:

- The class method can be used as a decorator to statically mark methods when the agent class is defined.
- The instance method dynamically exports methods, and can be used with methods not defined on the agent class.

Each take an optional export name argument, which defaults to the method name. Here are the two export method signatures:

Instance method:

```
RPC.export(method, name=None)
```

Class method:

```
RPC.export(name=None)
```

And here is an example agent definition using both methods:

```
from volttron.platform.vip import Agent, Core, RPC

def add(a, b):
    '''Add two numbers and return the result'''
    return a + b


class ExampleAgent(Agent):
    @RPC.export
    def say_hello(self, name):
        '''Build and return a hello string'''
        return 'Hello, %s!' % (name,)
```

(continues on next page)

(continued from previous page)

```
    @RPC.export('say_bye')
    def bye(self, name):
        '''Build and return a goodbye string'''
        return 'Goodbye, %s.' % (name,)

    @Core.receiver('setup')
    def onsetup(self, sender, **kwargs):
        self.vip.rpc.export('add')
```

Calling exported methods

The RPC subsystem provides three methods for calling exported RPC methods:

```
RPC.call(peer, method, *args, **kwargs)
```

Call the remote `method` exported by `peer` with the given arguments. Returns a *gevent AsyncResult* object.

```
RPC.batch(peer, requests)
```

Batch call remote methods exported by *peer*. *requests* must be an iterable of 4-tuples `(notify, method, args, kwargs)`, where `notify` is a boolean indicating whether this is a notification or standard call, `method` is the method name, `args` is a list and `kwargs` is a dictionary. Returns a list of *AsyncResult* objects for any standard calls. Returns `None` if all requests were notifications.

```
RPC.notify(peer, method, *args, **kwargs)
```

Send a one-way notification message to *peer* by calling *method* without returning a result.

Here are some examples:

```
self.vip.rpc.call(peer, 'say_hello', 'Bob').get()
results = self.vip.rpc.batch(peer, [(False, 'say_bye', 'Alice', {}), (True, 'later', 
↪[], {})])
self.vip.rpc.notify(peer, 'ready')
```

Inspection

A list of methods is available by calling the *inspect* method. Additional information can be returned for any method by appending `.inspect` to the method name. Here are a couple examples:

```
self.vip.rpc.call(peer, 'inspect')   # Returns a list of exported methods
self.vip.rpc.call(peer, 'say_hello.inspect')   # Return metadata on say_hello method
```

VCTL RPC Commands

There are two rpc subcommands available through vctl, *list* and *code*.

The list subcommand displays all of the agents that have a peer connection to the instance and which methods are available from each of these agents.

```
vctl rpc list
    config.store
            delete_config
            get_configs
            manage_delete_config
            manage_delete_store
            manage_get
            manage_get_metadata
            manage_list_configs
            manage_list_stores
            manage_store
            set_config
    .
    .
    .

    platform.historian
            get_aggregate_topics
            get_topic_list
            get_topics_by_pattern
            get_topics_metadata
            get_version
            insert
            query
    volttron.central
            get_publickey
            is_registered
```

If a single agent is specified, it will list all methods available for that agent.

```
vctl rpc list platform.historian
    platform.historian
            get_aggregate_topics
            get_topic_list
            get_topics_by_pattern
            get_topics_metadata
            get_version
            insert
            query
```

If the -v option is selected, all agent subsystem rpc methods will be displayed for each selected agent as well.

```
vctl rpc list -v platform.historian
    platform.historian
            get_aggregate_topics
            get_topic_list
            get_topics_by_pattern
            get_topics_metadata
            get_version
            insert
            query
            agent.version
            health.set_status
            health.get_status
            health.get_status_json
            health.send_alert
            heartbeat.start
```

(continues on next page)

(continued from previous page)

```
            heartbeat.start_with_period
            heartbeat.stop
            heartbeat.restart
            heartbeat.set_period
            config.update
            config.initial_update
            auth.update
```

If an agent is specified, and then a method (or methods) are specified, all parameters associated with the method(s) will be output.

```
vctl rpc list platform.historian get_version query
    platform.historian
        get_version
        Parameters:
        query
        Parameters:
            topic:
                    {'kind': 'POSITIONAL_OR_KEYWORD', 'default': None}
            start:
                    {'kind': 'POSITIONAL_OR_KEYWORD', 'default': None}
            end:
                    {'kind': 'POSITIONAL_OR_KEYWORD', 'default': None}
            agg_type:
                    {'kind': 'POSITIONAL_OR_KEYWORD', 'default': None}
            agg_period:
                    {'kind': 'POSITIONAL_OR_KEYWORD', 'default': None}
            skip:
                    {'kind': 'POSITIONAL_OR_KEYWORD', 'default': 0}
            count:
                    {'kind': 'POSITIONAL_OR_KEYWORD', 'default': None}
            order:
                    {'kind': 'POSITIONAL_OR_KEYWORD', 'default': 'FIRST_TO_LAST'}
```

By adding the '-v' option to this stage, the doc-string description of the method will be displayed along with the method and parameters if available.

```
vctl rpc list -v platform.historian get_version
    platform.historian
        get_version
        Documentation:
            RPC call to get the version of the historian

            :return: version number of the historian used
            :rtype: string

        Parameters:

vctl rpc code
vctl rpc list <peer identity>
vctl rpc list <peer identity> <method>
vctl rpc list -v <peer identity>
vctl rpc list -v <peer identity> <method>
vctl rpc code -v
vctl rpc code <peer identity>
vctl rpc code <peer identity> <method>
```

The code subcommand functions similarly to list, except that it will output the code to be used in an agent when writing an rpc call. Any available parameters are included as a list in the line of code where the parameters will need to be provided. These will need to be modified based on the use case.

```
vctl rpc code
    self.vip.rpc.call(config.store, delete_config, ['config_name', 'trigger_callback',
↪ 'send_update']).get()
    self.vip.rpc.call(config.store, get_configs).get()
    self.vip.rpc.call(config.store, manage_delete_config, ['args', 'kwargs']).get()
    self.vip.rpc.call(config.store, manage_delete_store, ['args', 'kwargs']).get()
    self.vip.rpc.call(config.store, manage_get, ['identity', 'config_name', 'raw']).
↪get()
    self.vip.rpc.call(config.store, manage_get_metadata, ['identity', 'config_name']).
↪get()
    self.vip.rpc.call(config.store, manage_list_configs, ['identity']).get()
    self.vip.rpc.call(config.store, manage_list_stores).get()
    self.vip.rpc.call(config.store, manage_store, ['args', 'kwargs']).get()
    self.vip.rpc.call(config.store, set_config, ['config_name', 'contents', 'trigger_
↪callback', 'send_update']).get()
    .
    .
    .
    self.vip.rpc.call(platform.historian, get_aggregate_topics).get()
    self.vip.rpc.call(platform.historian, get_topic_list).get()
    self.vip.rpc.call(platform.historian, get_topics_by_pattern, ['topic_pattern']).
↪get()
    self.vip.rpc.call(platform.historian, get_topics_metadata, ['topics']).get()
    self.vip.rpc.call(platform.historian, get_version).get()
    self.vip.rpc.call(platform.historian, insert, ['records']).get()
    self.vip.rpc.call(platform.historian, query, ['topic', 'start', 'end', 'agg_type',
↪ 'agg_period', 'skip', 'count', 'order']).get()
    self.vip.rpc.call(volttron.central, get_publickey).get()
    self.vip.rpc.call(volttron.central, is_registered, ['address_hash', 'address']).
↪get()
```

As with rpc list, the code subcommand can be filtered based on the vip identity and/or the method(s).

```
vctl rpc code platform.historian
    self.vip.rpc.call(platform.historian, get_aggregate_topics).get()
    self.vip.rpc.call(platform.historian, get_topic_list).get()
    self.vip.rpc.call(platform.historian, get_topics_by_pattern, ['topic_pattern']).
↪get()
    self.vip.rpc.call(platform.historian, get_topics_metadata, ['topics']).get()
    self.vip.rpc.call(platform.historian, get_version).get()
    self.vip.rpc.call(platform.historian, insert, ['records']).get()
    self.vip.rpc.call(platform.historian, query, ['topic', 'start', 'end', 'agg_type',
↪ 'agg_period', 'skip', 'count', 'order']).get()
```

```
vctl rpc code platform.historian query
    self.vip.rpc.call(platform.historian, query, ['topic', 'start', 'end', 'agg_type',
↪ 'agg_period', 'skip', 'count', 'order']).get()
```

Implementation

See the RPC module for implementation details.

Also see *Multi-Platform RPC Communication* and *RPC in RabbitMQ* for additional resources.

VIP Known Identities

It is critical for systems to have known locations for receiving resources and services from in a networked environment. The following table details the vip identities that are reserved for VOLTTRON specific usage.

VIP Identity	Sphere of Influence	Notes
platform	Platform	
platform.agent	Platform	The PlatformAgent is responsible for this identity. It is used to allow the VolttronCentralAgent to control and individual platform.
volttron.central	Multi-Network	The default identity for a VolttronCentralAgent. The PlatformAgent by default will use this as it's manager, but can be overridden in the configuration file of individual agents.
platform.historian	platform	An individual platform may have many historians available to it, however the only one that will be available through Volttron Central by default will be called this. Note that this does not require a specific type of historian, just that it's VIP Identity.
control	platform	Control is used to control the individual platform. From the command line when issuing any volttron-ctl operations or when using Volttron Central.
pubsub	platform	Pub/Sub subsystem router
platform.actuator	actuator	Agent which coordinates sending control commands to devices.
config.store	platform	The configuration subsystem service agent on the platform.
platform.driver	devices	The default identity for the Master Driver Agent.

VIP Authentication

VIP (VOLTTRON Interconnect Protocol) authentication is implemented in the `auth module` and extends the ZeroMQ Authentication Protocol ZAP to VIP by including the ZAP User-Id in the VIP payload, thus allowing peers to authorize access based on ZAP credentials. This document does not cover ZAP in any detail, but its understanding is fundamental to securely configuring ZeroMQ. While this document will attempt to instruct on securely configuring VOLTTRON for use on the Internet, it is recommended that the ZAP documentation also be consulted.

Default Encryption

By default, ZeroMQ operates in plain-text mode, without any sort of encryption. While this is okay for in-process and interprocess communications, via UNIX domain sockets, it is insecure for any kind of inter-network communications, especially when traffic must traverse the Internet. Therefore, VOLTTRON automatically generates an encryption key and enables CurveMQ by default on all TCP connections.

To see VOLTTRON's public key run the `vctl auth serverkey` command. For example:

```
(volttron)[user@home]$ volttron-ctl auth serverkey
FSG7LHhy3v8tdNz3gK35G6-oxUcyln54pYRKu5fBJzU
```

Peer Authentication

ZAP defines a method for verifying credentials exchanged when a connection is initially established. The authentication mechanism provides three main pieces of information useful for authentication:

- domain: a name assigned to a locally bound address (to which peers connect)
- address: the remote address of the peer
- credentials: includes the authentication method and any associated credentials

During authentication, VOLTTRON checks these pieces against a list of accepted peers defined in a file, called the “auth file” in this document. This JSON-formatted file is located at `$VOLTTRON_HOME/auth.json` and must have a matching entry in the allow list for remote connections to be accepted.

The auth file should not be modified directly. To change the auth file, use `vctl auth` subcommands: `add`, `list`, `remove`, and `update`. (Run `vctl auth --help` for more details and see the *authentication commands documentation*.)

Here are some example entries:

```
(volttron)[user@home]$ vctl auth list

INDEX: 0
{
  "domain": null,
  "user_id": "platform",
  "roles": [],
  "enabled": true,
  "mechanism": "CURVE",
  "capabilities": [],
  "groups": [],
  "address": null,
  "credentials": "k1C9-FPRAVjL-cH1iQqAJaCHUNVXaAlkVc7EqK0u9mI",
  "comments": "Automatically added by platform on start"
}

INDEX: 2
{
  "domain": null,
  "user_id": "platform.sysmon",
  "roles": [],
  "enabled": true,
  "mechanism": "CURVE",
  "capabilities": [],
  "groups": [],
  "address": null,
  "credentials": "5UD_GTk5dM2g4pk8d1-wM-BYgt4RAKiHf4SnT_YU6jY",
  "comments": "Automatically added on agent install"
}
```

Note: If using regular expressions in the “address” portion, denote this with “/”. Backslashes must be escaped “".

This is a valid regular expression: `"/192\\.168\\.1\\..*/"`

These are invalid: `"/192\.168\.1\..*/", "/192\.168\.1\..*", "192\\.168\\.1\\..*"`

When authenticating, the credentials are checked. If they don’t exist or don’t match, authentication fails. Otherwise, if domain and address are not present (or are null), authentication succeeds. If address and/or domain exist, they must match as well for authentication to succeed.

CURVE credentials include the remote peer's public key. Watching the **INFO** level log output of the auth module can help determine the required values for a specific peer.

Configuring Agents

A remote agent must know the platform's public key (also called the server key) to successfully authenticate. This server key can be passed to the agent's `__init__` method in the `serverkey` parameter, but in most scenarios it is preferable to add the server key to the *known-hosts file*.

URL-style Parameters

VOLTTRON extends ZeroMQ's address scheme by supporting URL-style parameters for configuration. The following parameters are supported when connecting:

- serverkey: encoded public key of remote server
- secretkey: agent's own private/secret key
- publickey: agent's own public key
- ipv6: instructs ZeroMQ to attempt to use IPv6

  Note: Although these parameters are still supported they should rarely need to be specified in the VIP-address URL. Agent *key stores* and the *known-hosts file* are automatically used when possible.

Platform Configuration

By default, the platform only listens on the local IPC VIP socket. Additional addresses may be bound using the `--vip-address` option, which can be provided multiple times to bind multiple addresses. Each VIP address should follow the standard ZeroMQ convention of prefixing with the socket type (*ipc://* or *tcp://*) and may include any of the following additional URL parameters:

- domain: domain name to associate with this endpoint (defaults to "vip")
- secretkey: alternate private/secret key (defaults to generated key for *tcp://*)
- ipv6: instructs ZeroMQ to attempt to use IPv6

Example Setup

Suppose agent `A` needs to connect to a remote platform `B`. First, agent `A` must know platform `B`'s public key (the *server key*) and platform `B`'s IP address (including port). Also, platform `B` needs to know agent `A`'s public key (let's say it is `HOVXfTspZWcpHQcYT_xGcqypBHzQHTgqEzVb4iXrcDg`).

Given these values, a user on agent `A`'s platform adds platform `B`'s information to the *known-hosts file*.

At this point agent `A` has all the infomration needed to connect to platform `B`, but platform `B` still needs to add an authentication entry for agent `A`.

If agent `A` tried to connect to platform `B` at this point both parties would see an error. Agent `A` would see an error similar to:

```
No response to hello message after 10 seconds.
A common reason for this is a conflicting VIP IDENTITY.
Shutting down agent.
```

Platform B (if started with *-v* or *-vv*) will show an error:

```
2016-10-19 14:21:20,934 () volttron.platform.auth INFO: authentication failure:␣
↪domain='vip', address='127.0.0.1', mechanism='CURVE', credentials=[
↪'HOVXfTspZWcpHQcYT_xGcqypBHzQHTgqEzVb4iXrcDg']
```

Agent A failed to authenticat to platform B because the platform didn't have agent A's public in the authentication list.

To add agent A's public key, a user on platform B runs:

```
(volttron)[user@platform-b]$ volttron-ctl auth add
domain []:
address []:
user_id []: Agent-A
capabilities (delimit multiple entries with comma) []:
roles (delimit multiple entries with comma) []:
groups (delimit multiple entries with comma) []:
mechanism [CURVE]:
credentials []: HOVXfTspZWcpHQcYT_xGcqypBHzQHTgqEzVb4iXrcDg
comments []:
enabled [True]:
```

Now if agent A can successfully connect to platform B, and platform B's log will show:

```
2016-10-19 14:26:16,446 () volttron.platform.auth INFO: authentication success:␣
↪domain='vip', address='127.0.0.1', mechanism='CURVE', credentials=[
↪'HOVXfTspZWcpHQcYT_xGcqypBHzQHTgqEzVb4iXrcDg'], user_id='Agent-A'
```

For a more details see the *authentication walk-through*.

VIP Authorization

VIP *authentication* and authorization go hand in hand. When an agent authenticates to a VOLTTRON platform that agent proves its identity to the platform. Once authenticated, an agent is allowed to connect to the *message bus*. VIP authorization is about giving a platform owner the ability to limit the capabilities of authenticated agents.

There are two parts to authorization:

1. Required capabilities (specified in agent's code)
2. Authorization entries (specified via `volttron-ctl auth` commands)

The following example will walk through how to specify required capabilities and grant those capabilities in authorization entries.

Single Capability

For this example suppose there is a temperature agent that can read and set the temperature of a particular room. The agent author anticipates that building managers will want to limit which agents can set the temperature.

In the temperature agent, a required capability is specified by using the `RPC.allow` decorator:

```
@RPC.export
def get_temperature():
   ...

@RPC.allow('CAP_SET_TEMP')
```

(continues on next page)

(continued from previous page)

```
@RPC.export
def set_temperature(temp):
    ...
```

In the code above, any agent can call the `get_temperature` method, but only agents with the `CAP_SET_TEMP` capability can call `set_temperature`.

Note: Capabilities are arbitrary strings. This example follows the general style used for Linux capabilities, but it is up to the agent author.

Now that a required capability has been specified, suppose a VOLTTRON platform owner wants to allow a specific agent, say *Alice Agent*, to set the temperature.

The platform owner runs `vctl auth add` to add new authorization entries or `vctl auth update` to update an existing entry. If *Alice Agent* is installed on the platform, then it already has an authorization entry. Running `vctl auth list` shows the existing entries:

```
...
INDEX: 3
{
  "domain": null,
  "user_id": "AliceAgent",
  "roles": [],
  "enabled": true,
  "mechanism": "CURVE",
  "capabilities": [],
  "groups": [],
  "address": null,
  "credentials": "JydrFRRv-kdSejL6Ldxy978pOf8HkWC9fRHUWKmJfxc",
  "comments": null
}
...
```

Currently AliceAgent cannot set the temperature because it does not have the `CAP_SET_TEMP` capability. To grant this capability the platform owner runs `vctl auth update 3`:

```
(For any field type "clear" to clear the value.)
domain []:
address []:
user_id [AliceAgent]:
capabilities (delimit multiple entries with comma) []: CAP_SET_TEMP
roles (delimit multiple entries with comma) []:
groups (delimit multiple entries with comma) []:
mechanism [CURVE]:
credentials [JydrFRRv-kdSejL6Ldxy978pOf8HkWC9fRHUWKmJfxc]:
comments []:
enabled [True]:
updated entry at index 3
```

Now *Alice Agent* can call `set_temperature` via RPC. If other agents try to call that method they will get the following exception:

```
error: method "set_temperature" requires capabilities set(['CAP_SET_TEMP']),
but capability list [] was provided
```

Multiple Capabilities

Expanding on the temperature-agent example, the `set_temperature` method can require agents to have multiple capabilities:

```
@RPC.allow(['CAP_SET_TEMP', 'CAP_FOO_BAR'])
@RPC.export
def set_temperature():
   ...
```

This requires an agent to have both the `CAP_SET_TEMP` and the `CAP_FOO_BAR` capabilities. Multiple capabilities can also be specified by using multiple `RPC.allow` decorators:

```
@RPC.allow('CAP_SET_TEMP')
@RPC.allow('CAN_FOO_BAR')
@RPC.export
def temperature():
   ...
```

Capability with parameter restriction

Capabilities can also be used to restrict access to a rpc method only with certain parameter values. For example, if *Agent A* exposes a method bar which accepts parameter *x*.

AgentA's capability enabled exported RPC method:

```
@RPC.export
@RPC.allow('can_call_bar')
def bar(self, x):
   return 'If you can see this, then you have the required capabilities'
```

You can restrict access to *Agent A*'s *bar* method to *Agent B* with `x=1`. To add this auth entry use the `vctl auth add` command as show below:

```
vctl auth add --capabilities '{"test1_cap2":{"x":1}}' --user_id AgentB --credential␣
↪vELQORgWOUcXo69DsSmHiCCLesJPa4-CtVfvoNHwIR0
```

The auth.json file entry for the above command would be:

```
{
  "domain": null,
  "user_id": "AgentB",
  "roles": [],
  "enabled": true,
  "mechanism": "CURVE",
  "capabilities": {
    "test1_cap2": {
      "x": 1
    }
  },
  "groups": [],
  "address": null,
  "credentials": "vELQORgWOUcXo69DsSmHiCCLesJPa4-CtVfvoNHwIR0",
  "comments": null
}
```

Parameter values can also be regular expressions:

```
(volttron)volttron@volttron1:~/git/myvolttron$ vctl auth add
domain []:
address []:
user_id []:
capabilities (delimit multiple entries with comma) []: {'test1_cap2':{'x':'/.*'}}
roles (delimit multiple entries with comma) []:
groups (delimit multiple entries with comma) []:
mechanism [CURVE]:
credentials []: vELQORgWOUcXo69DsSmHiCCLesJPa4-CtVfvoNHwIR0
comments []:
enabled [True]:
added entry domain=None, address=None, mechanism='CURVE', credentials=u
↪'vELQORgWOUcXo69DsSmHiCCLesJPa4-CtVfvoNHwIR0', user_id='b22e041d-ec21-4f78-b32e-
↪ab7138c22373'
```

The auth.json file entry for the above command would be:

```
{
  "domain": null,
  "user_id": "90f8ef35-4407-49d8-8863-4220e95974c7",
  "roles": [],
  "enabled": true,
  "mechanism": "CURVE",
  "capabilities": {
    "test1_cap2": {
      "x": "/.*"
    }
  },
  "groups": [],
  "address": null,
  "credentials": "vELQORgWOUcXo69DsSmHiCCLesJPa4-CtVfvoNHwIR0",
  "comments": null
}
```

Protecting Pub/Sub Topics

VIP *authorization* enables VOLTTRON platform owners to protect pub/sub topics. More specifically, a platform owner can limit who can publish to a given topic. This protects subscribers on that platform from receiving messages (on the protected topic) from unauthorized agents.

Example

To protect a topic, add the topic name to `$VOLTTRON_HOME/protected_topics.json`. For example, the following protected-topics file declares that the topic `foo` is protected:

```
{
   "write-protect": [
      {"topic": "foo", "capabilities": ["can_publish_to_foo"]}
   ]
}
```

Note: The capability name `can_publish_to_foo` is not special; It can be any string, but it is easier to manage capabilities with meaningful names.

Now only agents with the capability `can_publish_to_foo` can publish to the topic `foo`. To add this capability to authenticated agents, run `vctl auth update` (or `volttron-ctl auth add` for new authentication entries), and enter `can_publish_to_foo` in the capabilities field:

```
capabilities (delimit multiple entries with comma) []: can_publish_to_foo
```

Agents that have the `can_publish_to_foo` capabilities can publish to topic `foo`. That is, such agents can call:

```
self.vip.pubsub.publish('pubsub', 'foo', message='Here is a message')
```

If unauthorized agents try to publish to topic `foo` they will get an exception:

```
to publish to topic "foo" requires capabilities ['can_publish_to_foo'], but␣
↪capability list [] was provided
```

Regular Expressions

Topic names in `$VOLTTRON_HOME/protected_topics.json` can be specified as regular expressions. In order to use a regular expression, the topic name must begin and end with a "/". For example:

```
{
   "write-protect": [
      {"topic": "/foo/*.*/", "capabilities": ["can_publish_to_foo"]}
   ]
}
```

This protects topics such as `foo/bar` and `foo/anything`.

VIP Enhancements

When creating VIP for VOLTTRON 3.0 we wanted to address two security concerns and one user request:

- Security Concern 1: Agents can spoof each other on the VOLTTRON message bus and fake messages.
- Security Concern 2: Agents can subscribe to topics that they are not authorized to subscribe to.
- User Request 1: Several users requested means to transfer large amounts of data between agents without using the message bus.

VOLTTRON Interconnect Protocol (VIP) was created to address these issues but unfortunately, it broke the easy to use pub-sub messaging model of VOLTTRON. Additionally to use the security features of VOLTTRON in 3.0 code has become an ordeal especially when multiple platforms are concerned. Finally, VIP has introduced the requirement for knowledge of specific other platforms to agents written by users in order to be able to communicate. The rest of this memo focuses on defining the way VOLTTRON message bus will work going forward indefinitely and should be used as the guiding principles for any future work on VIP and VOLTTRON.

VOLTTRON Message Bus Guiding Principles:

1. All communications between two or more different VOLTTRON platforms MUST go through the VIP Router. Said another way, a user agent (application) should have *NO* capability to reach out to an agent on a different VOLTTRON platform directly.

All communications between two or more VOLTTRON platforms must be in the form of topics on the message bus. Agents *MUST* not use a distinct platform address or name to communicate via a direct connection between two platforms.

2. VOLTTRON will use two TCP ports. One port is used to extend VIP across platforms. A second port is used for the VOLTTRON discovery protocol (more on this to come on a different document). VIP will establish bi-directional communication via a single TCP port.
3. In order to solve the bootstrapping problem that CurveMQ has punted on, we will modify VIP to operate similar (behaviorally) to SSH.

A. On a single VOLTTRON platform, the platform's public key will be made available via an API so that all agents will be able to communicate with the platform. Additionally, the behavior of the platform will be changed so that agents on the same platform will automatically be added to the *auth.json* file. No more need for user to add the agents manually to the file. The desired behavior is similar to how SSH handles *known_hosts*.

Note: This behavior still addresses the security request 1 & 2.

B. When connecting VOLTTRON platforms, VOLTTRON Discovery Protocol (VDP) will be used to discover the other platforms public key to establish the router to router connection. Note that since we *BANNED* agent to agent communication between two platforms, we have prevented an "O(N^2)" communication pattern and key bootstrapping problem.

C. Authorization determines what agents are allowed to access what topics. Authorization MUST be managed by the VOLTTRON Central platform on a per organization basis. It is not recommended to have different authorization profiles on different VOLTTRON instances belonging to the same organization.

D. VOLTTRON message bus uses topics such as and will adopt an information model agreed upon by the VOLTTRON community going forward. Our initial information model is based on the OpenEIS schema going forward. A different document will describe the information model we have adopted going forward. All agents are free to create their own topics but the VOLTTRON team (going forward) will support the common VOLTTRON information model and all agents developed by PNNL will be converted to use the new information model.

E. Two connected VOLTTRON systems will exchange a list of available topics via the message router. This will allow each VIP router to know what topics are available at what VOLTTRON platform.

F. Even though each VOLTTRON platform will have knowledge of what topics are available around itself, no actual messages will be forwarded between VOLTTRON platforms until an agent on a specific platform subscribes to a topic. When an agent subscribes to a topic that has a publisher on a different VOLTTRON platform, the VIP router will send a request to its peer routers so that the messages sent to that topic will be forwarded. There will be cases (such as clean energy transactive project) where the publisher to a topic may be multiple hops away. In this case, the subscribe request will be sent towards the publisher through other VIP routers. In order to find the most efficient path, we may need to keep track of the total number of hops (in terms of number of VIP routers).

G. The model described in steps 5/6/7 applies to data collection. For control applications, VOLTTRON team only allows control actions to be originated from the VOLTTRON instance that is directly connected to that controlled device. This decision is made to increase the robustness of the control agent and to encourage truly distributed applications to be developed.

H. Direct agent to agent communication will be supported by creation of an ephemeral topic under the topic hierarchy. Our measurements have shown repeatedly that the overhead of using the ZeroMQ message pub/sub is minimal and has zero impact on communications throughput.

In summary, by making small changes to the way VIP operates, I believe that we can significantly increase the usability of the platform and also correct the mixing of two communication platforms into VIP. VOLTTRON message bus will return to being a pub/sub messaging system going forward. Direct agent to agent communication will be supported through the message bus.

Agent VIP IDENTITY Assignment Specification

This document explains how an agent obtains it's VIP IDENTITY, how the platform sets an agent's VIP IDENTITY at startup, and what mechanisms are available to the user to set the VIP IDENTITY for any agent.

What is a VIP IDENTITY

A VIP IDENTITY is a platform instance unique identifier for agents. The IDENTITY is used to route messages from one Agent through the VOLTTRON router to the recipient Agent. The VIP IDENTITY provides a consistent, user defined, and human readable character set to build a VIP IDENTITY. VIP IDENTITIES should be composed of both upper and lowercase letters, numbers and the following special characters.

Runtime

The primary interface for obtaining a VIP IDENTITY *at runtime* is via the runtime environment of the agent. At startup the utility function *vip_main* shall check for the environment variable **AGENT_VIP_IDENTITY**. If the **AGENT_VIP_IDENTITY** environment variable is not set then the *vip_main* function will fall back to a supplied identity argument. *vip_main* will pass the appropriate identity argument to the agent constructor. If no identity is set the Agent class will create a random VIP IDENTITY using python's *uuid4* function.

An agent that inherits from the platform's base Agent class can get it's current VIP IDENTITY by retrieving the value of `self.core.identity`.

The primary use of the 'identity' argument to *vip_main* is for agent development. For development it allows agents to specify a default VIP IDENTITY when run outside the platform. As platform Agents are not started via *vip_main* they will simply receive their VIP IDENTITY via the identity argument when they are instantiated. Using the identity argument of the Agent constructor to set the VIP IDENTITY via agent configuration is no longer supported.

At runtime the platform will set the environment variable **AGENT_VIP_IDENTITY** to the value set at installation time.

Agents not based on the platform's base Agent should set their VIP IDENTITY by setting the identity of the ZMQ socket before the socket connects to the platform. If the agent fails to set it's VIP IDENTITY via the ZMQ socket it will be selected automatically by the platform. This platform chosen ID is currently not discoverable to the agent.

Agent Implementation

If an Agent has a preferred VIP IDENTITY (for example the Master Driver Agent prefers to use "platform.driver") it may specify this as a default packed value. This is done by including a file named IDENTITY containing only the desired VIP IDENTITY in ASCII plain text in the same directory at the *setup.py* file for the Agent. This will cause the packaged agent wheel to include an instruction to set the VIP IDENTITY at installation time.

This value may be overridden at packaging or installation time.

Packaging

An Agent may have it's VIP IDENTITY configured when it is packaged. The packaged value may be used by the platform to set the **AGENT_VIP_IDENTITY** environment variable for the agent process.

The packaged VIP IDENTITY may be overridden at installation time. This overrides any preferred VIP IDENTITY of the agent. This will cause the packaged agent wheel to include an instruction to set the VIP IDENTITY at installation time.

To specify the VIP IDENTITY when packaging use the `--vip-identity` option when running *volttron-pkg package*.

Installation

An agent may have it's VIP IDENTITY configured when it is installed. This overrides any VIP IDENTITY specified when the agent was packaged.

To specify the VIP IDENTITY when packaging use the `--vip-identity` option when running *volttron-ctl install*.

Installation Default VIP IDENTITY

If no VIP IDENTITY has been specified by installation time the platform will assign one automatically.

The platform uses the following template to generate a VIP IDENTITY:

```
"{agent_name}_{n}"
```

`{agent_name}` is substituted with the name of the actual agent such as `listeneragent-0.1`

`{n}` is a number to make VIP IDENTITY unique. `{n}` is set to the first unused number (starting from 1) for all installed instances of an agent. e.g. If there are 2 listener agents installed and the first (VIP IDENTITY listeneragent-0.1_1) is uninstalled leaving the second (VIP IDENTITY "listeneragent-0.1_2"), a new listener agent will receive the VIP IDENTITY "listeneragent-0.1_1" when installed. The next installed listener will receive a VIP IDENTITY of "listeneragent-0.1_3".

The `#` sign is used to prevent confusing the agent version number with the installed instance number.

If an agent is repackaged with a new version number it is treated as a new agent and the number will start again from 1.

VIP IDENTITY Conflicts During Installation

If an agent is assigned a VIP IDENTITY besides the default value given to it by the platform it is possible for VIP IDENTITY conflicts to exist between installed agents. In this case the platform rejects the installation of an agent with a conflicting VIP IDENTITY and reports an error to the user.

VIP IDENTITY Conflicts During Runtime

In the case where agents are not started through the platform (usually during development or when running standalone agents) it is possible to encounter a VIP IDENTITY conflict during runtime. In this case the first agent to use a VIP IDENTITY will function as normal. Subsequent agents will still connect to the ZMQ socket but will be silently rejected by the platform router. The router will not route any message to that Agent. Agents using the platforms base Agent will detect this automatically during the initial handshake with the platform. This condition will shutdown the Agent with an error indicating a VIP IDENTITY conflict as the most likely cause of the problem.

Auto Numbering With Non-Default VIP IDENTITYs

It is possible to use the auto numbering mechanism that the default VIP IDENTITY scheme uses. Simply include the string `{n}` somewhere in the requested VIP IDENTITY and it will be replaced with a number in the same manner as the default VIP IDENTITY is. Python *string.format()* escaping rules apply. See this question on StackOverflow.

Script Features

The *scripts/install-agent.py* script supports specifying the desired VIP IDENTITY using the `-i` (or `--vip-identity`) `<identity>` option

Security/Privacy

Currently, much like the *TAG* file in an installed agent, there is nothing to stop someone from modifying the *IDENTITY* file in the installed agent.

Constraints and Limitations

Currently there is no way for an agent based on the platform base Agent class to recover from a VIP IDENTITY conflict. This case only affects developers and a very tiny minority of users and is reported via an error message, there are currently no plans to fix it.

Design Overview

What Problems does VIP Address?

When VOLTTRON agents, controllers, or other entities needed to exchange data, they previously used the first generation pub/sub messaging mechanism and ad-hoc methods to set up direct connections. While the pub/sub messaging is easy to implement and use, it suffers from several limitations:

- It requires opening two listening sockets: one each for publishing and subscribing.
- There is no trivial way to prevent message spoofing.
- There is no trivial way to enable private messaging
- It is not ideal for peer-to-peer communications.

These limitations have severe security implications. For improved security in VOLTTRON, the communications protocol must provide a method for secure data exchange that is intuitive and simple to implement and use.

Many messaging platforms already provides many of the building blocks to implement encrypted and authenticated communications over a shared socket. They include a socket type implementing the router pattern. What remains is a protocol built on the ZeroMQ and/or RabbitMQ to provide a single connection point, secure message passing, and retain the ability for entities to come and go as they please.

VIP is VOLTTRON protocol implementation targeting the limitations above.

ZeroMQ

Why ZeroMQ?

Rather than reinvent the wheel, VIP makes use of many features already implemented in ZeroMQ, including ZAP and CurveMQ. While VIP doesn't require the use of ZAP or CurveMQ, their use substantially improves security by encrypting traffic over public networks and limiting connections to authenticated peers.

ZeroMQ also provides reliable transports with built-in framing, automatic reconnection, in-process zero-copy message passing, abstractions for underlying protocols, and so much more. While some of these features create other pain points, they are minimal compared with the effort of either reimplementing or cobbling together libraries.

VIP is a routing protocol

VIP uses the ZeroMQ router pattern. Specifically, the router binds a ROUTER socket and peers connect using a DEALER or ROUTER socket. Unless the peer is connecting a single socket to multiple routers, using the DEALER socket is easiest, but there are instances where using a ROUTER is more appropriate. One must just exercise care to include the proper address envelope to ensure proper routing.

Extensible Security

VIP makes no assumptions about the security mechanisms used. It works equally well over encrypted or unencrypted channels. Any connection-level authentication and encryption is handled by ZAP. Message-level authentication can be implemented in the protocols and services using VIP or by utilizing message properties set in ZAP replies.

ZeroMQ Compatibility

For enhanced security, VOLTTRON recommends libzmq version 4.1 or greater, however, most features of VIP are available with older versions. The following is an incomplete list of core features available with recent versions of libzmq.

- Version 3.2:
  - Basic, unauthenticated, unencrypted routing
  - Use ZMQ_ROUTER_BEHAVIOR socket option instead of ZMQ_ROUTER_MANDATORY
- Version 4.0:
  - Adds authentication and encryption via ZAP
- Version 4.1:
  - Adds message properties allowing correlating authentication tokens to messages

Message Format and Version Detection

VIP uses a simple, multi-frame format for its messages. The first one (for peers) or two (for router) frames contain the delivery address(es) and are follow immediately by the VIP signature `VIP1`. The first characters of the signature are used to match the protocol and the last character digit indicates the protocol version, which will be incremented as the protocol is revised. This allows for fail-fast behavior and backward compatibility while being simple to implement in any language supported by ZeroMQ.

Formal Specification

Architecture

VIP defines a message-based dialog between a *router* that transfers data between *peers*. The *router* and *peers* SHALL communicate using the following socket types and transports:

- The router SHALL use a ROUTER socket.
- Peers SHALL use a DEALER or ROUTER socket.
- The router SHALL bind to one or more endpoints using inproc, tcp, or ipc address types.
- Peers SHALL connect to these endpoints.
- There MAY be any number of peers.

Message Format

A routing exchange SHALL consist of a peer sending a message to the router followed by the router receiving the message and sending it to the destination peer.

Messages sent to the router by peers SHALL consist of the following message frames:

- The *recipient*, which SHALL contain the socket identity of the destination peer.
- The protocol signature, which SHALL contain the four octets "VIP1".
- The *user id*, which SHALL be an implementation-defined value.
- The *request id*, which SHALL contain an opaque binary blob.
- The *subsystem*, which SHALL contain a string.
- The *data*, which SHALL be zero or more subsystem-specific opaque frames.

Messages received from a peer by the router will automatically have a *sender* frame prepended to the message by the ROUTER socket. When the router forwards the message, the sender and recipient fields are swapped so that the *recipient* is in the first frame and the *sender* is in the second frame. The *recipient* frame is automatically stripped by the ROUTER socket during delivery. Peers using ROUTER sockets must prepend the message with an *intermediary* frame, which SHALL contain the identity of a router socket.

Messages received from the router by peers SHALL consist of the following message frames:

- The *sender*, which SHALL contain the socket identity of the source peer.
- The protocol signature, which SHALL contain the four octets "VIP1".
- The *user id*, which MAY contain a UTF-8 encoded string.
- The *request id*, which SHALL contain an opaque binary blob.
- The *subsystem*, which SHALL contain a non-empty string.
- The *data*, which SHALL be zero or more subsystem-specific opaque frames.

The various fields have these meanings:

- sender: the ZeroMQ DEALER or ROUTER identity of the sending (source) peer.
- recipient: the ZeroMQ DEALER or ROUTER identity of the recipient (destination) peer.
- intermediary: the ZeroMQ ROUTER identity of the intermediary router.

- user id: VIP authentication metadata set in the authenticator. See the discussion below for more information on this value.
- request id: the meaning of this field is defined by the sending peer. Replies SHALL echo the request id without modifying it.
- subsystem: this specifies the peer subsystem the data is intended for. The length of a subsystem name SHALL NOT exceed 255 characters and MUST only contain ASCII characters.
- data: provides the data for the given subsystem. The number of frames required is defined by each subsystem.

User ID

The value in the *user id* frame depends on the implementation and the version of ZeroMQ. If *ZAP* is used with libzmq 4.1.0 or newer, peers should send an empty string for the user id and the ZAP authenticator will replace it with an authentication token which receiving peers may use to authorize access. If ZAP is not used or a version of libzmq is used which lacks support for retrieving the user id metadata, an authentication subsystem may be used to authenticate peers. The authentication subsystem SHALL provide peers with private tokens that must be sent with each message in the user id frame and which the router will substitute with a public token before forwarding. If the message cannot be authenticated, the user id received by peers SHALL be a zero-length string.

Socket Types

Peers communicating via the router will typically use DEALER sockets and should not require additional handling. However, a DEALER peer may only connect to a single router. Peers may use ROUTER sockets to connect to multiple endpoints, but must prepend the routing ID of the destination.

When using a DEALER socket:

- A peer SHALL not send in intermediary address.
- A peer SHALL connect to a single endpoint.

When using a ROUTER socket:

- A peer SHALL prepend the intermediary routing ID of to the message frames.
- A peer MAY connect to multiple endpoints.

Routing Identities

Routing identities are set on a socket using the ZMQ_IDENTITY socket option and MUST be set on both ROUTER and DEALER sockets. The following additional requirements are placed on the use of peer identities:

- Peers SHALL set a valid identity rather than rely on automatic identity generation.
- The router MAY drop messages with automatically generated identities, which begin with the zero byte (‘0’).

A zero length identity is invalid for peers and is, therefore, unroutable. It is used instead to address the router itself.

- Peers SHALL use a zero length recipient to address the router.
- Messages sent from the router SHALL have a zero length sender address.

Error Handling

The documented default behavior of ZeroMQ ROUTER sockets when entering the mute state (when the send buffer is full) is to silently discard messages without blocking. This behavior, however, is not consistently observed. Quietly discarding messages is not the desired behavior anyway because it prevents peers from taking appropriate action to the error condition.

- Routers SHALL set the ZMQ_SNDTIMEO socket option to 0.
- Routers SHALL forward EAGAIN errors to sending peers.

It is also the default behavior of ROUTER sockets to silently drop messages addressed to unknown peers.

- Routers SHALL set the ZMQ_ROUTER_MANDATORY socket option.
- Routers SHALL forward EHOSTUNREACH errors to sending peers, unless the recipient address matches the sender.

Most subsystems are optional and some way of communicating unsupported subsystems to peers is needed.

- The error code 93, EPROTONOSUPPORT, SHALL be returned to peers to indicate unsupported or unimplemented subsystems.

The errors above are reported via the *error* subsystem. Other errors MAY be reported via the *error* subsystem, but subsystems SHOULD provide mechanisms for reporting subsystem-specific errors whenever possible.

An error message must contain the following:

- The recipient frame SHALL contain the socket identity of the original sender of the message.
- The sender frame SHALL contain the socket identity of the reporting entity, usually the router.
- The request ID SHALL be copied from the from the message which triggered the error.
- The subsystem frame SHALL be the 5 octets 'error'.
- The first data frame SHALL be a string representation of the error number.
- The second data frame SHALL contain a UTF-8 string describing the error.
- The third data frame SHALL contain the identity of the original recipient, as it may differ from the reporter.
- The fourth data frame SHALL contain the subsystem copied from the subsystem field of the offending message.

Subsystems

Peers may support any number of communications protocols or subsystems. For instance, there may be a remote procedure call (RPC) subsystem which defines its own protocol. These subsystems are outside the scope of VIP and this document with the exception of the *hello* and *ping* subsystems.

- A router SHALL implement the hello subsystem.
- All peers and routers SHALL implement the ping subsystem.

The hello Subsystem

The hello subsystem provides one simple RPC-style routine for peers to probe the router for version and identity information.

A peer hello request message must contain the following:

- The recipient frame SHALL have a zero length value.

- The request id MAY have an opaque binary value.
- The subsystem SHALL be the 5 characters "hello".
- The first data frame SHALL be the five octets 'hello' indicating the operation.

A peer hello reply message must contain the following:

- The sender frame SHALL have a zero length value.
- The request id SHALL be copied unchanged from the associated request.
- The subsystem SHALL be the 7 characters "hello".
- The first data frame SHALL be the 7 octets 'welcome'.
- The second data frame SHALL be a string containing the router version number.
- The third data frame SHALL be the router's identity blob.
- The fourth data frame SHALL be the peer's identity blob.

The hello subsystem can help a peer with the following tasks:

- Test that a connection is established.
- Discover the version of the router.
- Discover the identity of the router.
- Discover the identity of the peer.
- Discover authentication metadata.

For instance, if a peer will use a ROUTER socket for its connections, it must first know the identity of the router. The peer might first connect with a DEALER socket, issue a hello, and use the returned identity to then connect the ROUTER socket.

The ping Subsystem

The *ping* subsystem is useful for testing the presence of a peer and the integrity and latency of the connection. All endpoints, including the router, must support the ping subsystem.

A peer ping request message must contain the following:

- The recipient frame SHALL contain the identity of the endpoint to query.
- The request id MAY have an opaque binary value.
- The subsystem SHALL be the 4 characters "ping".
- The first data frame SHALL be the 4 octets 'ping'.
- There MAY be zero or more additional data frames containing opaque binary blobs.

A ping response message must contain the following:

- The sender frame SHALL contain the identity of the queried endpoint.
- The request id SHALL be copied unchanged from the associated request.
- The subsystem SHALL be the 4 characters "ping".
- The first data frame SHALL be the 4 octets 'pong'.
- The remaining data frames SHALL be copied from the ping request unchanged, starting with the second data frame.

Any data can be included in the ping and should be returned unchanged in the pong, but limited trust should be placed in that data as it is possible a peer might modify it against the direction of this specification.

Discovery

VIP does not define how to discover peers or routers. Typical options might be to hard code the router address in peers or to pass it in via the peer configuration. A well known (i.e. statically named) directory service might be used to register connected peers and allow for discovery by other peers.

Example Exchanges

These examples show the messages *as sent on the wire* as sent or received by peers using DEALER sockets. The messages received or sent by peers or routers using ROUTER sockets will have an additional address at the start. We do not show the frame sizes or flags, only frame contents.

Example of hello Request

This shows a hello request sent by a peer, with identity "alice", to a connected router, with identity "router".

```
+-+
| |                 Empty recipient frame
+-+----+
| VIP1 |            Signature frame
+-+----+
| |                 Empty user ID frame
+-+----+
| 0001 |            Request ID, for example "0001"
+------++
| hello |           Subsystem, "hello" in this case
+-------+
| hello |           Operation, "hello" in this case
+-------+
```

This example assumes a DEALER socket. If a peer uses a ROUTER socket, it SHALL prepend an additional frame containing the router identity, similar to the following example.

This shows the example request received by the router:

```
+-------+
| alice |           Sender frame, "alice" in this case
+-+-----+
| |                 Empty recipient frame
+-+----+
| VIP1 |            Signature frame
+-+----+
| |                 Empty user ID frame
+-+----+
| 0001 |            Request ID, for example "0001"
+------++
| hello |           Subsystem, "hello" in this case
+-------+
| hello |           Operation, "hello" in this case
+-------+
```

This shows an example reply sent by the router:

```
+--------+
| alice |           Recipient frame, "alice" in this case
+-+------+
| |                 Empty sender frame
+-+-----+
| VIP1 |            Signature frame
+-+-----+
| |                 Empty authentication metadata in user ID frame
+-+-----+
| 0001 |            Request ID, for example "0001"
+-------++
| hello |           Subsystem, "hello" in this case
+--------+-+
| welcome |         Operation, "welcome" in this case
+-----+---+
| 1.0 |             Version of the router
+-----+--+
| router |          Router ID, "router" in this case
+--------++
| alice |           Peer ID, "alice" in this case
+--------+
```

This shows an example reply received by the peer:

```
+-+
| |                 Empty sender frame
+-+-----+
| VIP1 |            Signature frame
+-+-----+
| |                 Empty authentication metadata in user ID frame
+-+-----+
| 0001 |            Request ID, for example "0001"
+-------++
| hello |           Subsystem, "hello" in this case
+--------+-+
| welcome |         Operation, "welcome" in this case
+-----+---+
| 1.0 |             Version of the router
+-----+--+
| router |          Router ID, "router" in this case
+--------++
| alice |           Peer ID, "alice" in this case
+--------+
```

Example of ping Subsystem

This shows a ping request sent by the peer "alice" to the peer "bob" through the router "router".

```
+------+
| bob |             Recipient frame, "bob" in this case
+------++
| VIP1 |            Signature frame
+-+-----+
| |                 Empty user ID frame
+-+-----+
```

(continues on next page)

(continued from previous page)

```
| 0002 |              Request ID, for example "0002"
+-------+
| ping |              Subsystem, "ping" in this case
+-------+
| ping |              Operation, "ping" in this case
+------+-----+
| 1422573492 |        Data, a single frame in this case (Unix timestamp)
+------------+
```

This shows the example request received by the router:

```
+--------+
| alice |             Sender frame, "alice" in this case
+-----+-+
| bob |               Recipient frame, "bob" in this case
+-----++
| VIP1 |              Signature frame
+-+----+
| |                   Empty user ID frame
+-+----+
| 0002 |              Request ID, for example "0002"
+-------+
| ping |              Subsystem, "ping" in this case
+-------+
| ping |              Operation, "ping" in this case
+------+-----+
| 1422573492 |        Data, a single frame in this case (Unix timestamp)
+------------+
```

This shows the example request forwarded by the router:

```
+-----+
| bob |               Recipient frame, "bob" in this case
+-----+-+
| alice |             Sender frame, "alice" in this case
+-------++
| VIP1 |              Signature frame
+-+----+
| |                   Empty authentication metadata in user ID frame
+-+----+
| 0002 |              Request ID, for example "0002"
+-------+
| ping |              Subsystem, "ping" in this case
+-------+
| ping |              Operation, "ping" in this case
+------+-----+
| 1422573492 |        Data, a single frame in this case (Unix timestamp)
+------------+
```

This shows the example request received by "bob":

```
+--------+
| alice |             Sender frame, "alice" in this case
+-------++
| VIP1 |              Signature frame
+-+----+
| |                   Empty authentication metadata in user ID frame
```

(continues on next page)

(continued from previous page)

```
+-+----+
| 0002 |             Request ID, for example "0002"
+------+
| ping |             Subsystem, "ping" in this case
+------+
| ping |             Operation, "ping" in this case
+------+-----+
| 1422573492 |       Data, a single frame in this case (Unix timestamp)
+------------+
```

If "bob" were using a ROUTER socket, there would be an additional frame prepended to the message containing the router identity, "router" in this case.

This shows an example reply from "bob" to "alice"

```
+-------+
| alice |            Recipient frame, "alice" in this case
+------++
| VIP1 |             Signature frame
+-+----+
| |                  Empty user ID frame
+-+----+
| 0002 |             Request ID, for example "0002"
+------+
| ping |             Subsystem, "ping" in this case
+------+
| pong |             Operation, "pong" in this case
+------+-----+
| 1422573492 |       Data, a single frame in this case (Unix timestamp)
+------------+
```

The message would make its way back through the router in a similar fashion to the request.

Reference Implementation

Reference VIP router: https://github.com/VOLTTRON/volttron/blob/master/volttron/platform/vip/router.py

Reference VIP peer: https://github.com/VOLTTRON/volttron/blob/master/volttron/platform/vip/socket.py

1.41.3 RabbitMQ Based VOLTTRON

RabbitMQ VOLTTRON uses the *Pika* library for the RabbitMQ message bus implementation. To install Pika, it is recommended to use the VOLTTRON *bootstrap.py* script:

```
python3 bootstrap.py --rabbitmq
```

Configuration

To setup a VOLTTRON instance to use the RabbitMQ message bus, we need to first configure VOLTTRON to use the RabbitMQ message library. The contents of the RabbitMQ configuration file should follow the pattern below.

Path: *$VOLTTRON_HOME/rabbitmq_config.yml*

```yaml
#host parameter is mandatory parameter. fully qualified domain name
host: mymachine.pnl.gov

# mandatory. certificate data used to create root ca certificate. Each volttron
# instance must have unique common-name for root ca certificate
certificate-data:
  country: 'US'
  state: 'Washington'
  location: 'Richland'
  organization: 'PNNL'
  organization-unit: 'VOLTTRON Team'
  # volttron1 has to be replaced with actual instance name of the VOLTTRON
  common-name: 'volttron1_root_ca'
#
# optional parameters for single instance setup
#
virtual-host: 'volttron' # defaults to volttron

# use the below four port variables if using custom rabbitmq ports
# defaults to 5672
amqp-port: '5672'

# defaults to 5671
amqp-port-ssl: '5671'

# defaults to 15672
mgmt-port: '15672'

# defaults to 15671
mgmt-port-ssl: '15671'

# defaults to true
ssl: 'true'

# defaults to ~/rabbitmq_server/rabbbitmq_server-3.7.7
rmq-home: "~/rabbitmq_server/rabbitmq_server-3.7.7"
```

Each VOLTTRON instance resides within a RabbitMQ virtual host. The name of the virtual host needs to be unique per VOLTTRON instance if there are multiple virtual instances within a single host/machine. The hostname needs to be able to resolve to a valid IP. The default port of an AMQP port without authentication is *5672* and with authentication it is *5671*. The default management HTTP port without authentication is *15672* and with authentication is *15671*. These needs to be set appropriately if the default ports are not used.

The 'ssl' flag indicates if SSL based authentication is required or not. If set to *True*, information regarding SSL certificates needs to be also provided. SSL based authentication is described in detail in Authentication And Authorization With RabbitMQ Message Bus.

To configure the VOLTTRON instance to use RabbitMQ message bus, run the following command:

```
vcfg --rabbitmq single [optional path to rabbitmq_config.yml]
```

At the end of the setup process, a RabbitMQ broker is setup to use the configuration provided. A new topic exchange for the VOLTTRON instance is created within the configured virtual host.

On platform startup, VOLTTRON checks for the type of message bus to be used. If using the RabbitMQ message bus, the RabbitMQ platform router is instantiated. The RabbitMQ platform router:

- Connects to RabbitMQ broker (with or without authentication)

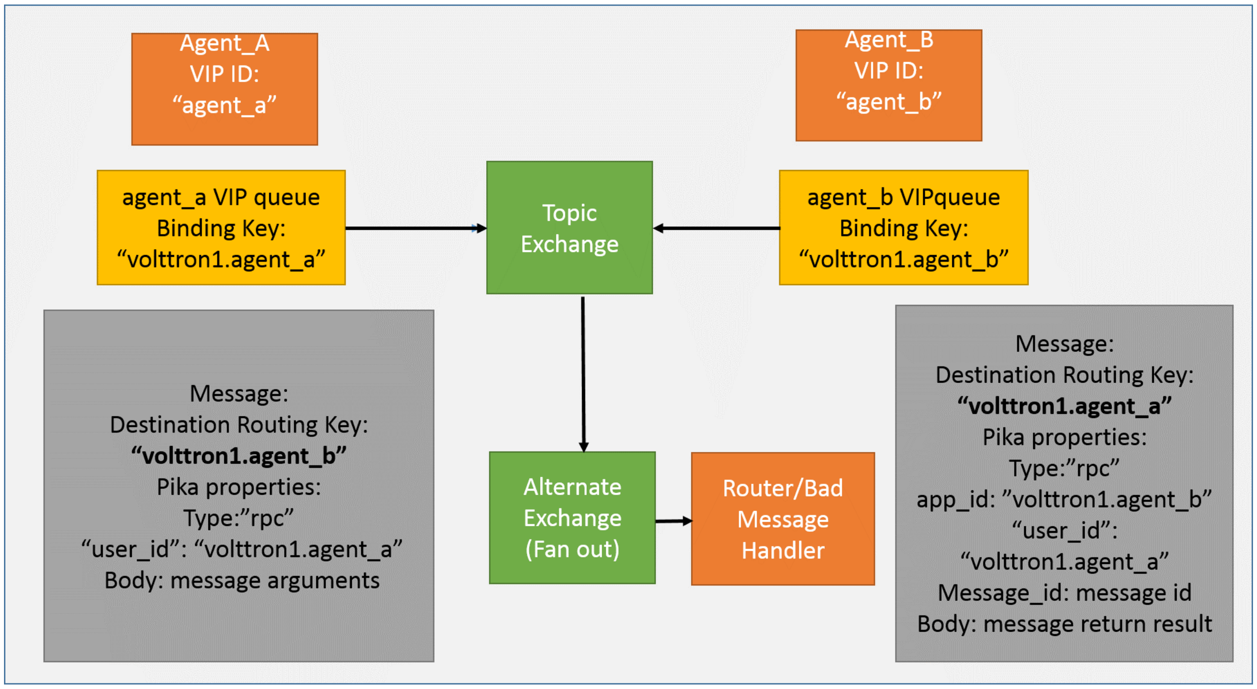
- Creates a VIP queue and binds itself to the "VOLTTRON" exchange with binding key <*instance-name*>.*router*. This binding key makes it unique across multiple VOLTTRON instances in a single machine as long as each instance has a unique instance name.
- Handles messages intended for router module such as *hello*, *peerlist*, *query* etc.
- Handles "unrouteable" messages - Messages which cannot be routed to any destination agent are captured and an error message indicating "Host Unreachable" error is sent back to the caller.
- Disconnects from the broker when the platform shuts down.

When any agent is installed and started, the Agent Core checks for the type of message bus used. If it is RabbitMQ message bus then:

- It creates a RabbitMQ user for the agent
- If SSL based authentication is enabled, client certificates for the agent is created
- Connect to the RabbitQM broker with appropriate connection parameters
- Creates a VIP queue and binds itself to the "VOLTTRON" exchange with binding key <*instance-name*>.<*agent identity*>
- Sends and receives messages using Pika library methods.
- Checks for the type of subsystem in the message packet that it receives and calls the appropriate subsystem message handler.
- Disconnects from the broker when the agent stops or platform shuts down.

RPC In RabbitMQ VOLTTRON

The agent functionality remain unchanged regardless of the underlying message bus used, meaning they can continue to use the same RPC interfaces without any change.



Consider two agents with VIP identities “agent_a” and “agent_b” connected to VOLTTRON platform with instance name “volttron1”. Agent A and B each have a VIP queue with binding key volttron1.agent_a” and “volttron1.agent_b”. Following is the sequence of operation when Agent A wants to make RPC call to Agent B:

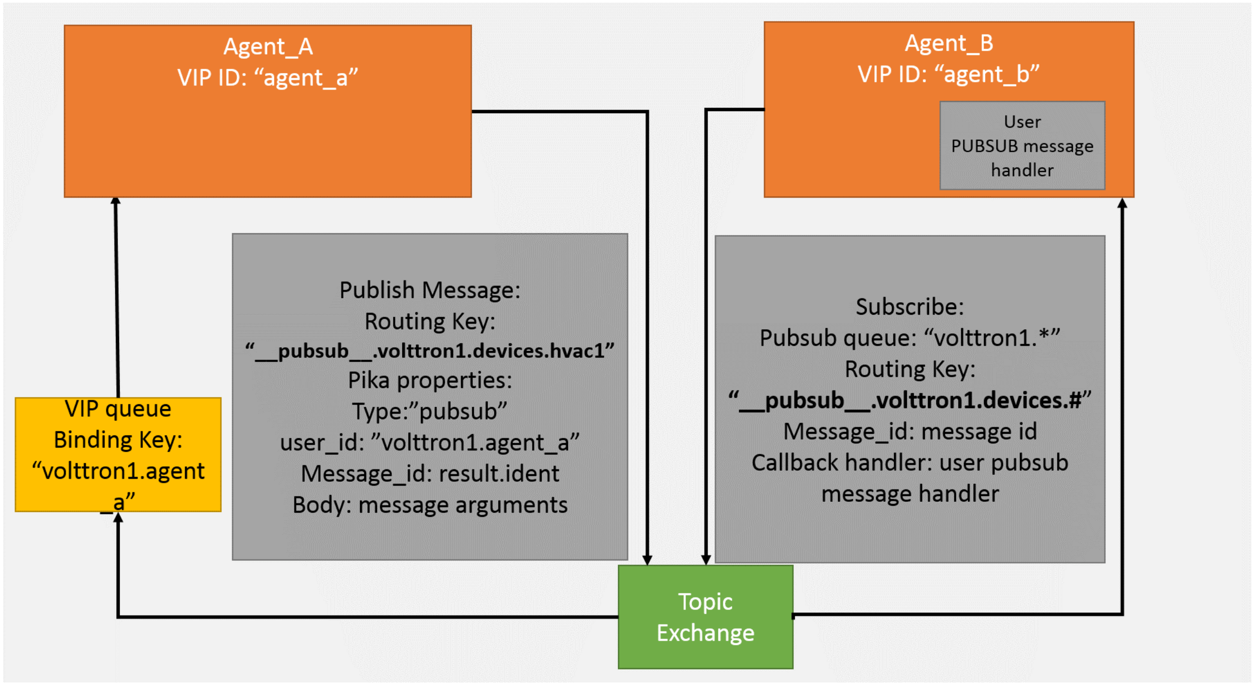
1. Agent A makes a RPC call to Agent B.

```
agent_a.vip.rpc.call("agent_b", set_point, "point_name", 2.5)
```

2. RPC subsystem wraps this call into a VIP message object and sends it to Agent B.
3. The VOLTTRON exchange routes the message to Agent B as the destination routing in the VIP message object matches with the binding key of Agent B.
4. Agent Core on Agent B receives the message, unwraps the message to find the subsystem type and calls the RPC subsystem handler.
5. RPC subsystem makes the actual RPC call *set_point()* and gets the result. It then wraps into VIP message object and sends it back to the caller.
6. The VOLTTRON exchange routes it to back to Agent A.
7. Agent Core on Agent A calls the RPC subsystem handler which in turn hands over the RPC result to Agent A application.

PUBSUB In RabbitMQ VOLTTRON

The agent functionality remains unchanged irrespective of the platform using ZeroMQ based pubsub or RabbitMQ based pubsub, i.e. agents continue to use the same PubSub interfaces and use the same topic format delimited by “/”. Since RabbitMQ expects binding key to be delimited by ‘.’, RabbitMQ PUBSUB internally replaces ‘/’ with “.”. Additionally, all agent topics are converted to `__pubsub__.<instance_name>.<remainder of topic>` to differentiate them from the main Agent VIP queue binding.



Consider two agents with VIP identities “agent_a” and “agent_b” connected to VOLTTRON platform with instance name “volttron1”. Agent A and B each have a VIP queue with binding key “volttron1.agent_a” and “volttron1.agent_b". Following is the sequence of operation when Agent A subscribes to a topic and Agent B publishes to same the topic:

1. Agent B makes subscribe call for topic "devices".

```
agent_b.vip.pubsub.subscribe("pubsub", prefix="devices", callback=self.onmessage)
```

2. Pubsub subsystem creates binding key from the topic `__pubsub__.volttron1.devices.#`
3. It creates a queue internally and binds the queue to the VOLTTRON exchange with the above binding key.
4. Agent B is publishing messages with topic: "devices/hvac1".

```
agent_b.vip.pubsub.publish("pubsub", topic="devices/hvac1", headers={}, message="foo
↪").
```

5. PubSub subsystem internally creates a VIP message object and publishes on the VOLTTRON exchange.
6. RabbitMQ broker routes the message to Agent B as routing key in the message matches with the binding key of the topic subscription.
7. The pubsub subsystem unwraps the message and calls the appropriate callback method of Agent A.

If agent wants to subscribe to topic from remote instances, it uses:

```
agent.vip.subscribe('pubsub', 'devices.hvac1', all_platforms=True)
```

It is internally set to `__pubsub__.*.<remainder of topic>`

Further Work

The Pubsub subsystem for the ZeroMQ message bus performs O(N) comparisons where N is the number of unique subscriptions. The RabbitMQ Topic Exchange was enhanced in version 2.6.0 to reduce the overhead of additional unique subscriptions to almost nothing in most cases. We speculate they are using a tree structure to store the binding keys which would reduce the search time to O(1) in most cases and O(ln) in the worst case. The VOLTTRON PubSub with ZeroMQ could be updated to match this performance scalability with some effort.

RabbitMQ Management Tool Integrated Into VOLTTRON

Some of the important native RabbitMQ control and management commands are now integrated with the :ref'volttron-ctl <Platform-Commands>' (vctl) utility. Using *volttron-ctl*'s RabbitMQ management utility, we can control and monitor the status of RabbitMQ message bus:

```
vctl rabbitmq --help
usage: vctl command [OPTIONS] ... rabbitmq [-h] [-c FILE] [--debug]
                                          [-t SECS]
                                          [--msgdebug MSGDEBUG]
                                          [--vip-address ZMQADDR]
                                          ...
subcommands:

    add-vhost           add a new virtual host
    add-user            Add a new user. User will have admin privileges
                        i.e,configure, read and write
    add-exchange        add a new exchange
    add-queue           add a new queue
    list-vhosts         List virtual hosts
```

(continues on next page)

(continued from previous page)

```
    list-users          List users
    list-user-properties
                        List users
    list-exchanges      add a new user
    list-exchange-properties
                        list exchanges with properties
    list-queues         list all queues
    list-queue-properties
                        list queues with properties
    list-bindings       list all bindings with exchange
    list-federation-parameters
                        list all federation parameters
    list-shovel-parameters
                        list all shovel parameters
    list-policies       list all policies
    remove-vhosts       Remove virtual host/s
    remove-users        Remove virtual user/s
    remove-exchanges    Remove exchange/s
    remove-queues       Remove queue/s
    remove-federation-parameters
                        Remove federation parameter
    remove-shovel-parameters
                        Remove shovel parameter
    remove-policies     Remove policy
```

For information about using RabbitMQ in multi-platform deployments, view the *docs*

RabbitMQ Overview

Note: Some of the RabbitMQ summary/overview documentation and supporting images added here are taken from the RabbitMQ official documentation.

RabbitMQ is the most popular messaging library with over 35,000 production deployments. It is highly scalable, easy to deploy, runs on many operating systems and cloud environments. It supports many kinds of distributed deployment methodologies such as clusters, federation and shovels.

RabbitMQ uses *Advanced Message Queueing Protocol* (AMQP) and works on the basic producer consumer model. A consumer is a program that consumes/receives messages and producer is a program that sends the messages. Following are some important definitions that we need to know before we proceed.

- Queue - Queues can be considered like a post box that stores messages until consumed by the consumer. Each consumer must create a queue to receives messages that it is interested in receiving. We can set properties to the queue during it's declaration. The queue properties are:
  - Name - Name of the queue
  - Durable - Flag to indicate if the queue should survive broker restart.
  - Exclusive - Used only for one connection and it will be removed when connection is closed.
  - Auto-queue - Flag to indicate if auto-delete is needed. The queue is deleted when last consumer unsubscribes from it.
  - Arguments - Optional, can be used to set message TTL (Time To Live), queue limit etc.

- Bindings - Consumers bind the queue to an exchange with binding keys or routing patterns. Producers send messages and associate them with a routing key. Messages are routed to one or many queues based on a pattern matching between a message routing key and binding key.
- Exchanges - Exchanges are entities that are responsible for routing messages to the queues based on the routing pattern/binding key used. They look at the routing key in the message when deciding how to route messages to queues. There are different types of exchanges and one must choose the type of exchange depending on the application design requirements

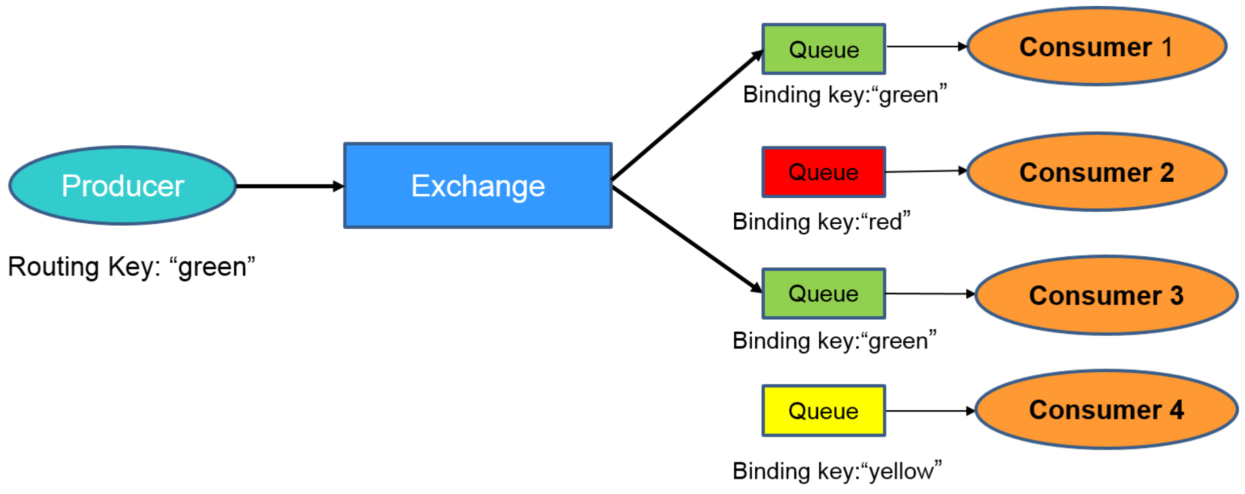
  1. Fanout - It blindly broadcasts the message it receives to all the queues it knows.
  2. Direct - Here, the message is routed to a queue if the routing key of the message exactly matches the binding key of the queue.
  3. Topic - Here, the message is routed to a queue based on pattern matching of the routing key with the binding key. The binding key and the routing key pattern must be a list of words delimited by dots, for example, "car.subaru.outback" or "car.subaru.*", "car.#". A message sent with a particular routing key will be delivered to all the queues that are bound with a matching binding key with some special rules as

     '*' (star) - can match exactly one word in that position. '#' (hash) - can match zero or more words
  4. Headers - If we need more complex matching then we can add a header to the message with all the attributes set to the values that need to be matched. The message is considered matching if the values of the attributes in the header is equal to that of the binding. The Header exchange ignores the routing key.

  We can set some properties of the exchange during it's declaration.

  – Name - Name of the exchange
  – Durable - Flag to indicate if the exchange should survive broker restart.
  – Auto-delete - Flag indicates if auto-delete is needed. If set to true, the exchange is deleted when the last queue is unbound from it.
  – Arguments - Optional, used by plugins and broker-specific features

Lets use an example to understand how they all fit together. Consider an example where there are four consumers (Consumer 1 - 4) interested in receiving messages matching the pattern "green", "red" or "yellow". In this example, we are using a direct exchange that will route the messages to the queues only when there is an exact match of the routing key of the message with the binding key of the queues. Each of the consumers declare a queue and bind the queue to the exchange with a binding key of interest. Lastly, we have a producer that is continuously sending messages to exchange with routing key "green". The exchange will check for an exact match and route the messages to only Consumer 1 and Consumer 3.



For more information about queues, bindings, exchanges, please refer to the RabbitMQ tutorial.

Authentication in RabbitMQ

By default RabbitMQ supports SASL PLAIN authentication with username and password. RabbitMQ supports other SASL authentication mechanisms using plugins. In VOLTTRON we use one such external plugin based on x509 certificates (https://github.com/rabbitmq/rabbitmq-auth-mechanism-ssl). This authentication is based on a technique called public key cryptography which consists of a key pair - a public key and a private key. Data that has been encrypted with a public key can only be decrypted with the corresponding private key and vice versa. The owner of key pair makes the public key available and keeps the private confidential. To send a secure data to a receiver, a sender encrypts the data with the receiver's public key. Since only the receiver has access to his own private key only the receiver can decrypted. This ensures that others, even if they can get access to the encrypted data, cannot decrypt it. This is how public key cryptography achieves confidentiality.

A digital certificate is a digital file that is used to prove ownership of a public key. Certificates act like identification cards for the owner/entity. Certificates are therefore crucial to determine that a sender is using the right public key to encrypt the data in the first place. Digital Certificates are issued by Certification Authorities(CA). Certification Authorities fulfill the role of the *Trusted Third Party* by accepting Certificate applications from entities, authenticating applications, issuing Certificates and maintaining status information about the Certificates issued. Each CA has its own public private key pair and its public key certificate is called a root CA certificate. The CA attests to the identity of a Certificate applicant when it signs the Digital Certificate using its private key.

In x509 based authentication, a signed certificate is presented instead of username/password for authentication and if the server recognizes the the signer of the certificate as a trusted CA, accepts and allows the connection. Each server/system can maintain its own list of trusted CAs (i.e. list of public certificates of CAs). Certificates signed by any of the trusted CA would be considered trusted. Certificates can also be signed by intermediate CAs that are in turn signed by a trusted.

This section only provides a brief overview about the SSL based authentication. Please refer to the vast material available online for detailed description. Some useful links to start:

- https://en.wikipedia.org/wiki/Public-key_cryptography
- https://robertheaton.com/2014/03/27/how-does-https-actually-work/

Management Plugin

The RabbitMQ-management plugin provides an HTTP-based API for management and monitoring of RabbitMQ nodes and clusters, along with a browser-based UI and a command line tool, *rabbitmqadmin*. The management interface allows you to:

- Create, Monitor the status and delete resources such as virtual hosts, users, exchanges, queues etc.
- Monitor queue length, message rates and connection information and more
- Manage users and add permissions (read, write and configure) to use the resources
- Manage policies and runtime parameters
- Send and receive messages (for trouble shooting)

For more detailed information about the management plugin, please refer to RabbitMQ documentation on the Management Plugin.

Deployments

The *platform installation* docs describe performing first time setup for single machine RabbitMQ deployments.

See the *multi-platform RabbitMQ* docs for setting up shovel or federation in multi-platform RabbitMQ deployments.

Message Bus Plugin Framework

The message bus plugin framework aims to decouple the VOLTTRON specific code from the message bus implementation without compromising the existing features of the platform. The concept of the plugin framework is similar to that used in historian or driver framework i.e, we should be easily able to support multiple message buses and be able to use any of them by following few installation and setup steps.

Message Bus Refactor

It consists of five components

1. New connection class per message bus
2. Extensions to platform router functionality
3. Extensions to core agent functionality
4. A proxy agent for each message bus to support backward compatibility
5. Authentication related changes

Connection class

A connection class that has methods to handle

1. Connection to new message bus.
2. Set properties such as message transmission rate, send/receive buffer sizes, open socket limits etc.
3. Send/receive messages from the underlying layer.
4. Error handling functionality.
5. Disconnect from the message bus

Platform Level Changes

A new message bus flag is introduced to indicate the type of message bus used by the platform. If no message bus flag is added in the platform config file, the platform uses default ZeroMQ based message bus.

Path of the config: $VOLTTRON_HOME/config

```
[volttron]
vip-address = tcp://127.0.0.1:22916
instance-name = volttron1
message-bus = rmq
```

Please note, the valid message types are 'zmq' and 'rmq'.

On startup, platform checks for the type of message bus and creates appropriate router module. Please note, ZeroMQ router functionality remains unchanged. However, a new router module with limited functionality is added for RabbitMQ message bus. The actual routing of messages is handed over to the RabbitMQ broker and router module will only handle some of the necessary subsystem messages such as "hello", "peerlist", "query" etc. If a new message bus needs to be added then the complexity of the router module depends on whether the messaging library uses a broker based or broker less (as in case of ZeroMQ) protocol.

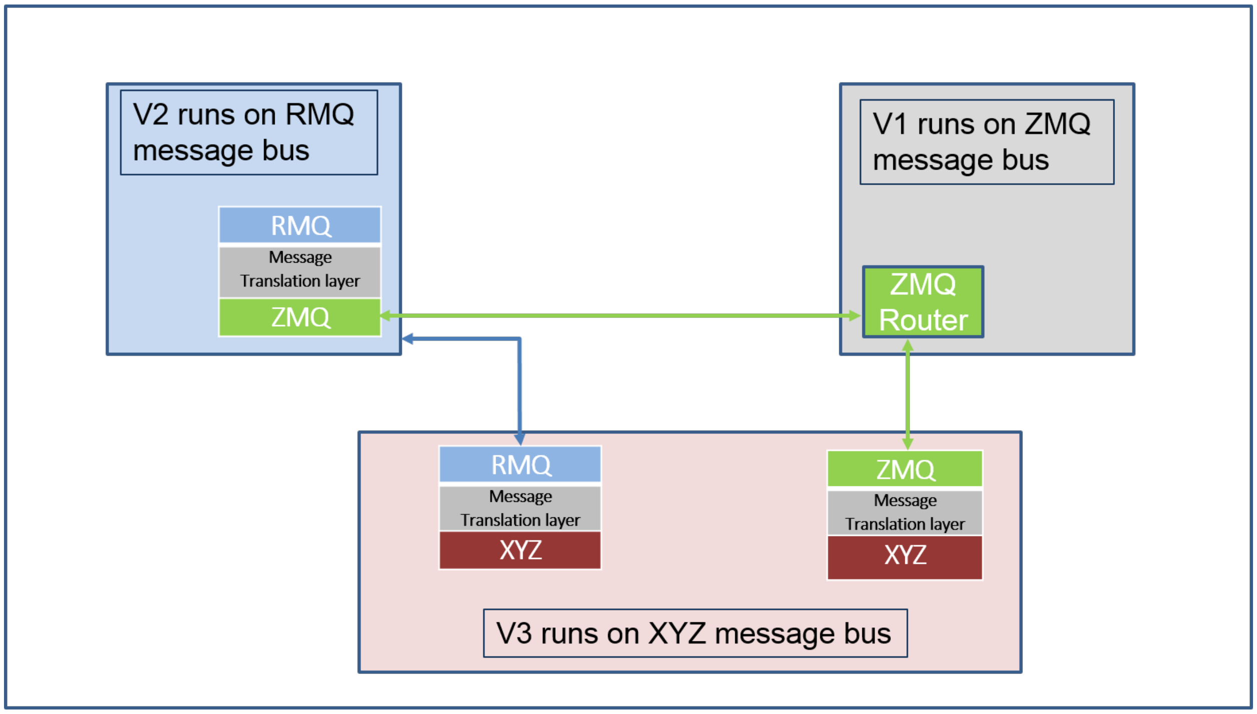
Agent Core Changes

The application specific code of the agent remains unchanged. The agent core functionality is modified to check the type of message bus and connect to and use the appropriate message bus. On startup, the agent Core checks the type of message bus, connects to appropriate message bus and routes messages to appropriate subsystem. All subsystem messages are encapsulated inside a message bus agnostic VIP message object. If a new message bus needs to be added, then we would have to extend the Agent Core to connect to new message bus.

Compatibility Between VOLTTRON Instances Running On Different Message Buses

All the agents connected to local platform uses the same message bus that the platform is connected to. But if we need agents running on different platforms with different message buses to communicate with each other then we need some kind of proxy entity or bridge that establishes the connection, handles the message routing and performs the message translation between the different message formats. To achieve that, we have a proxy agent that acts as a bridge between the local message bus and remote message bus. The role of the proxy agent is to

- Maintain connections to internal and external message bus.
- Route messages from internal to external platform.
- Route messages from external to internal platform.


V2 runs on RMQ message bus
RMQ
Message Translation layer
ZMQ
V1 runs on ZMQ message bus
ZMQ Router
RMQ
Message Translation layer
XYZ
ZMQ
Message Translation layer
XYZ
V3 runs on XYZ message bus

The above figure shows three VOLTTRON instances with V1 connected to ZMQ message bus, V2 connected to RMQ message bus and V3 connected to XYZ (some message bus of the future) and all three want to connect to each other. Then V2 and V3 will have proxy agents that get connected to the local bus and to the remote bus and forward messages from one to another.

Authentication And Authorization With RabbitMQ Message Bus

Authentication In RabbitMQ VOLTTRON

RabbitMQ VOLTTRON uses SSL based authentication, rather than the default username and password authentication. VOLTTRON adds SSL based configuration entries into the *rabbitmq.conf* file during the setup process. The necessary SSL configurations can be seen by running the following command:

```
cat ~/rabbitmq_server/rabbitmq_server-3.7.7/etc/rabbitmq/rabbitmq.conf
```

The configurations required to enable SSL:

```
listeners.ssl.default = 5671
ssl_options.cacertfile = VOLTTRON_HOME/certificates/certs/volttron1-trusted-cas.crt
ssl_options.certfile = VOLTTRON_HOME/certificates/certs/volttron1-server.crt
ssl_options.keyfile = VOLTTRON_HOME/certificates/private/volttron1-server.pem
ssl_options.verify = verify_peer
ssl_options.fail_if_no_peer_cert = true
```

Parameter explanations

- listeners.ssl.default: port for listening for SSL connections
- ssl_options.cacertfile: path to trusted Certificate Authorities (CA)
- ssl_options.certfile: path to server public certificate
- ssl_options.keyfile: path to server's private key
- ssl_options.verify: whether verification is enabled
- ssl_options.fail_if_no_peer_cert: upon client's failure to provide certificate, SSL connection either rejected (true) or accepted (false)
- auth_mechanisms.1: type of authentication mechanism. EXTERNAL means SSL authentication is used

SSL in RabbitMQ VOLTTRON

To configure RabbitMQ-VOLTTRON to use SSL based authentication, we need to add SSL configuration in rabbitmq_config.yml.

```
# mandatory. fully qualified domain name for the system
host: mymachine.pnl.gov

# mandatory. certificate data used to create root ca certificate. Each volttron
# instance must have unique common-name for root ca certificate
certificate-data:
  country: 'US'
  state: 'Washington'
  location: 'Richland'
  organization: 'PNNL'
  organization-unit: 'VOLTTRON Team'
  # volttron1 has to be replaced with actual instance name of the VOLTTRON instance
  common-name: 'volttron1_root_ca'

virtual-host: 'volttron' # defaults to volttron

# use the below four port variables if using custom rabbitmq ports
```

(continues on next page)

(continued from previous page)

```
# defaults to 5672
amqp-port: '5672'

# defaults to 5671
amqp-port-ssl: '5671'

# defaults to 15672
mgmt-port: '15672'

# defaults to 15671
mgmt-port-ssl: '15671'

# defaults to true
ssl: 'true'

# defaults to ~/rabbitmq_server/rabbbitmq_server-3.7.7
rmq-home: "~/rabbitmq_server/rabbitmq_server-3.7.7"
```

The parameters of interest for SSL based configuration are

- certificate-data: subject information needed to create certificates
- ssl: Flag set to 'true' for SSL based authentication
- amqp-port-ssl: Port number for SSL connection (defaults to 5671)
- mgmt-port-ssl: Port number for HTTPS management connection (defaults to 15671)

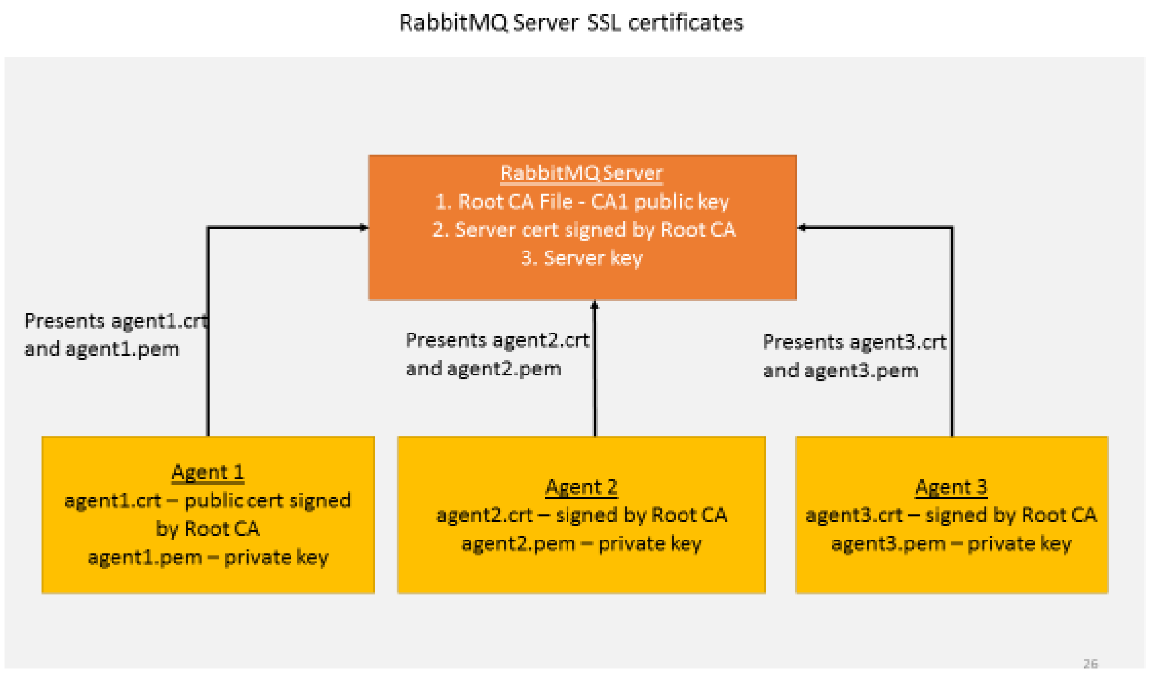
We can then configure the VOLTTRON instance to use SSL based authentication with the below command:

> vcfg –rabbitmq single <optional path to rabbitmq_config.yml>

When one creates a single instance of RabbitMQ, the following is created / re-created in the VOLTTRON_HOME/certificates directory:

- Public and private certificates of root Certificate Authority (CA)
- Public and private (automatically signed by the CA) server certificates needed by RabbitMQ broker
- Admin certificate for the RabbitMQ instance
- Public and private (automatically signed by the CA) certificates for VOLTTRON platform service agents.
- Trusted CA certificate

The public files can be found at `VOLTTRON_HOME/certificates/certs` and the private files can be found at `VOLTTRON_HOME/certificates/private`. The *trusted-cas.crt* file is used to store the root CAs of all VOLTTRON instances that the RabbitMQ server has to connected to. The trusted CA is only created once, but can be updated. Initially, the trusted CA is a copy of the the root CA file, but when an external VOLTTRON instance needs to be connected to an instance, the external VOLTTRON instance's root CA will be appended to this file in order for the RabbitMQ broker to trust the new connection.


RabbitMQ Server SSL certificates
RabbitMQ Server
1. Root CA File - CA1 public key
2. Server cert signed by Root CA
3. Server key
Presents agent1.crt and agent1.pem
Presents agent2.crt and agent2.pem
Presents agent3.crt and agent3.pem
Agent 1
agent1.crt – public cert signed by Root CA
agent1.pem – private key
Agent 2
agent2.crt – signed by Root CA
agent2.pem – private key
Agent 3
agent3.crt – signed by Root CA
agent3.pem – private key
26

Every RabbitMQ has a single self signed root ca and server certificate signed by the root CA. This is created during VOLTTRON setup and the RabbitMQ server is configured and started with these two certificates. Every time an agent is started, the platform automatically creates a pair of public-private certificates for that agent that is signed by the same root CA. When an agent communicates with the RabbitMQ message bus it presents it's public certificate and private key to the server and the server validates if it is signed by a root CA it trusts – ie., the root certificate it was started with. Since there is only a single root CA for one VOLTTRON instance, all the agents in this instance can communicate with the message bus over SSL.

For information about using SSL with multi-platform RabbitMQ deployments, view the *docs*

Authorization in RabbitMQ VOLTTRON

To be implemented in VOLTTRON at a later date.

For more detailed information about access control, please refer to RabbitMQ documentation Access Control.

1.41.4 Multi-Platform Communication

To connect to remote VOLTTRON platforms, we would need platform discovery information of the remote platforms. This information contains the platform name, VIP address and *serverkey* of the remote platforms and we need to provide this as part of multi-platform configuration.

Configuration

The configuration and authentication for multi-platform connection can be setup either manually or by running the platforms in set up mode. Both the setups are described below.

Setup Mode For Automatic Authentication

Note: It is necessary for each platform to have a web server if running in setup mode.

For ease of use and to support multi-scale deployment, the process of obtaining the platform discovery information and authenticating the new platform connection is automated. We can now bypass the manual process of adding auth keys (i.e., either by using the *volttron-ctl* utility or directly updating the *auth.json* config file).

A config file containing list of web addresses (one for each platform) need to be made available in *VOLTTRON_HOME* directory.

Name of the file: *external_address.json*

Directory path: Each platform's VOLTTRON_HOME directory.

For example: */home/volttron/.volttron1*

Contents of the file:

```
[
"http://<ip1>:<port1>",
"http://<ip2>:<port2>",
"http://<ip3>:<port3>",
 ......
]
```

We then start each VOLTTRON platform with setup mode option in this way.

```
volttron -vv -l volttron.log --setup-mode&
```

Each platform will obtain the platform discovery information of the remote platform that it is trying to connect through a HTTP discovery request and store the information in a configuration file (*$VOLTTRON_HOME/external_platform_discovery.json*). It will then use the VIP address and *serverkey* to connect to the remote platform. The remote platform shall authenticate the new connection and store the auth keys (public key) of the connecting platform for future use.

The platform discovery information will be stored in *VOLTTRON_HOME* directory and looks like below:

Name of config file: *external_platform_discovery.json*

Contents of the file:

```
{"<platform1 name>": {"vip-address":"tcp://<ip1>:<vip port1>",
                      "instance-name":"<platform1 name>",
                      "serverkey":"<serverkey1>"
                      },
 "<platform2 name>": {"vip-address":"tcp://<ip2>:<vip port2>",
                      "instance-name":"<platform2 name>",
                      "serverkey":"<serverkey2>"
                      },
 "<platform3 name>": {"vip-address":"tcp://<ip3>:<vip port3>",
                      "instance-name":"<platform3 name>",
                      "serverkey":"<serverkey3>"
                      },
  ......
}
```

Each platform will use this information for future connections.

Once the keys have been exchanged and stored in the auth module, we can restart all the VOLTTRON platforms in normal mode.

```
./stop-volttron
./start-volttron
```

Manual Configuration of External Platform Information

Platform discovery configuration file can also be built manually and it needs to be added inside *VOLTTRON_HOME* directory of each platform.

Name of config file: *external_platform_discovery.json*

Contents of the file:

```
{"<platform1 name>": {"vip-address":"tcp://<ip1>:<vip port1>",
                      "instance-name":"<platform1 nam>",
                      "serverkey":"<serverkey1>"
                      },
 "<platform2 name>": {"vip-address":"tcp://<ip2>:<vip port2>",
                      "instance-name":"<platform2 name>",
                      "serverkey":"<serverkey2>"
                      },
 "<platform3 name>": {"vip-address":"tcp://<ip3>:<vip port3>",
                      "instance-name":"<platform3 name>",
                      "serverkey":"<serverkey3>"
                      },
 ......
}
```

With this configuration, platforms can be started in normal mode.

```
./start-volttron
```

For external platform connections to be authenticated, we would need to add the credentials of the connecting platforms in each platform using the *volttron-ctl auth* utility. For more details *Agent authentication walk-through*.

See also:

Multi-Platform Walk-through

PubSub Communication Between Remote Platforms

This document describes pubsub communication between different platforms. The goal of this specification is to improve forward historians forwarding local PubSub messages to remote platforms. Agents interested in receiving PubSub messages from external platforms will not need to have a forward historian running on the source platform to forward PubSub messages to the interested destination platforms; The VIP router will now do all the work. It shall use the Routing Service to internally manage connections with external VOLTTRON platforms and use the PubSubService for the actual inter-platform PubSub communication.

For future:

This specification will need to be extended to support PubSub communication between platforms that are multiple hops away. The VIP router of each platform shall need to maintain a routing table and use it to forward pubsub messages to subscribed platforms that are multiple hops away. The routing table shall contain shortest path to each destination platform.

Functional Capabilities

1. Each VOLTTRON platform shall have a list of other VOLTTRON platforms that it has to connect to in a config file.
2. Routing Service of each platform connects to other platforms on startup.
3. The Routing Service in each platform is responsible for connecting to (and also initiating reconnection if required), monitoring and disconnecting from each external platform. The function of the Routing Service is explained in detail in the Routing Service section.
4. Platform to platform PubSub communication shall be using VIP protocol with the subsystem frame set to "pubsub".
5. The PubSubService of each VOLTTRON platform shall maintain a list of local and external subscriptions.
6. Each VIP router sends its list of external subscriptions to other connected platforms in the following cases:
   a. On startup
   b. When a new subscription is added
   c. When an existing subscription is removed
   d. When a new platform gets connected
7. When a remote platform disconnection is detected, all stale subscriptions related to that platform shall be removed.
8. Whenever an agent publishes a message to a specific topic, the PubSubService on the local platform first checks the topic against its list of local subscriptions. If a local subscription exists, it sends the publish message to corresponding local subscribers.
9. The PubSubService shall also check the topic against list of external subscriptions. If an external subscription exists, it shall use the Routing Service to send the publish message to the corresponding external platform.
10. Whenever a router receives messages from other platform, it shall check the destination platform in the incoming message.
    a. If the destination platform is the local platform, it hand overs the publish message to the PubSubService which checks the topic against list of external subscriptions. If an external subscription matches, the PubSubService forwards the message to all the local subscribers subscribed to that topic.
    b. If the destination platform is not the local platform, it discards the message.

Routing Service

1. The Routing Service shall maintain connection status (CONNECTING, CONNECTED, DISCONNECTED etc.) for each external platform.
2. In order to establish connection with an external VOLTTRON platform, the server key of the remote platform is needed. The Routing Service shall connect to an external platform once it obtains the server key for that platform from the KeyDiscoveryService.
3. The Routing Service shall exchange "hello"/"welcome" handshake messages with the newly connected remote platform to confirm the connection. It shall use VIP protocol with the subsystem frame set to "routing_table" for the handshake messages.
4. Routing Service shall monitor the connection status and inform the PubSubService whenever a remote platform gets connected/disconnected.

For Future:

1. Each VIP router shall exchange its routing table with its connected platforms on startup and whenever a new platform gets connected or disconnected.
2. The router shall go through each entry in the routing table that it received from other platforms and calculate the shortest, most stable path to each remote platform. It then sends the updated routing table to other platforms for adjustments in the forwarding paths (in their local routing table) if any.
3. Whenever a VIP router detects a new connection, it adds an entry into the routing table and sends updated routing table to its neighboring platforms. Each router in the other platforms shall update and re-calculate the forwarding paths in its local routing table and forward to rest of the platforms.
4. Similarly, whenever a VIP router detects a remote platform disconnection, it deletes the entry in the routing table for that platform and forwards the routing table to other platforms to do the same.

KeyDiscovery Service

1. Each platform tries to obtain the platform discovery information - platform name, VIP address and server key of remote VOLTTRON platforms through HTTP discovery service at startup.
2. If unsuccessful, it shall make regular attempts to obtain discovery information until successful.
3. The platform discovery information shall then be sent to the Routing Service using VIP protocol with subsystem frame set to "routing_table".

Messages for Routing Service

Below are example messages that are applicable to the Routing Service.

- Message sent by KeyDiscovery Service containing the platform discovery information (platform name, VIP address and server key) of a remote platform

```
+-+
| |                                 Empty recipient frame
+-+----+
| VIP1 |                            Signature frame
+-+----+
| |                                 Empty user ID frame
+-+----+
| 0001 |                            Request ID, for example "0001"
+----------------+
| routing_table |                   Subsystem, "routing_table"
+----------------+----------------+
| normalmode_platform_connection | Type of operation, "normalmode_
↪platform_connection"
+---------------------------------+
| platform discovery information |
| of external platform           | platform name, VIP address and server␣
↪key of external platform
+---------------------------------+
| platform name        | Remote platform for which the server key belongs␣
↪to.
+---------------------+
```

Handshake messages between two newly connected external VOLTTRON platform to confirm successful connection.

- Message from initiating platform

```
+-+
| |                         Empty recipient frame
+-+----+
| VIP1 |                    Signature frame
+-+----+
| |                         Empty user ID frame
+-+----+
| 0001 |                    Request ID, for example "0001"
+---------------++
| routing_table |           Subsystem, "routing_table"
+----------------+
| hello   |                 Operation, "hello"
+---------+
| hello   |                 Hello handshake request frame
+---------+-------+
| platform name |           Platform initiating a "hello"
+----------------+
```

- Reply message from the destination platform

```
+-+
| |                         Empty recipient frame
+-+----+
| VIP1 |                    Signature frame
+-+----+
| |                         Empty user ID frame
+-+----+
| 0001 |                    Request ID, for example "0001"
+---------------++
| routing_table |           Subsystem, "routing_table"
+---------+-------+
| hello   |                 Operation, "hello"
+---------++
| welcome |                 Welcome handshake reply frame
+---------+------+
| platform name |           Platform sending reply to "hello"
+----------------+
```

Messages for PubSub communication

The VIP routers of each platform shall send PubSub messages between platforms using VIP protocol message semantics. Below is an example of external subscription list message sent by VOLTTRON platform *V1* router to VOLTTRON platform *V2*.

```
+-+
| |                   Empty recipient frame
+-+----+
| VIP1 |              Signature frame
+-+---------+
|V1 user id |         Empty user ID frame
+-+---------+
| 0001 |              Request ID, for example "0001"
+-------++
| pubsub |            Subsystem, "pubsub"
+-------------+-+
```

(continues on next page)

(continued from previous page)

```
| external_list |   Operation, "external_list" in this case
+---------------+
| List of       |
| subscriptions |   Subscriptions dictionary consisting of VOLTTRON platform id and␣
↪list of topics as
+---------------+   key - value pairings, for example: { "V1": ["devices/rtu3"]}
```

This shows an example of an external publish message sent by the router of VOLTTRON platform *V2* to VOLTTRON platform *V1*.

```
+-+
| |                       Empty recipient frame
+-+----+
| VIP1 |                  Signature frame
+-+---------+
|V1 user id |             Empty user ID frame
+-+---------+
| 0001 |                  Request ID, for example "0001"
+-------++
| pubsub |                Subsystem, "pubsub"
+------------------+
| external_publish |      Operation, "external_publish" in this case
+------------------+
| topic            |      Message topic
+------------------+
| publish message  |      Actual publish message frame
+------------------+
```

API

Methods for Routing Service

- *external_route( )* - This method receives message frames from external platforms, checks the subsystem frame and redirects to appropriate subsystem (routing table, pubsub) handler. It shall run within a separate thread and get executed whenever there is a new incoming message from other platforms.
- *setup( )* - This method initiates socket connections with all the external VOLTTRON platforms configured in the config file. It also starts monitor thread to monitor connections with external platforms.
- *handle_subsystem(frames)* - Routing Service subsytem handler to handle serverkey message from KeyDiscoveryService and "hello/welcome" handshake message from external platforms.
- *send_external(instance_name, frames)* - This method sends input message to specified VOLTTRON platform/instance.
- *register(type, handler)* - Register method for PubSubService to register for connection and disconnection events.
- *disconnect_external_instances(instance_name)* - Disconnect from specified VOLTTRON platform.
- *close_external_connections( )* - Disconnect from all external VOLTTRON platforms.
- *get_connected_platforms( )* - Return list of connected platforms.

Methods for PubSubService

- *external_platform_add(instance_name)* - Send external subscription list to newly connected external VOLTTRON platform.
- *external_platform_drop(instance_name)* - Remove all subscriptions for the specified VOLTTRON platform
- *update_external_subscriptions(frames)* - Store/Update list of external subscriptions as per the subscription list provided in the message frame.
- *_distribute_external(frames)* - Publish the message all the external platforms that have subscribed to the topic. It uses send_external_pubsub_message() of router to send out the message.
- *external_to_local_publish(frames)* - This method retrieves actual message from the message frame, checks the message topic against list of external subscriptions and sends the message to corresponding subscribed agents.

Methods for agent pubsub subsystem

To subscribe to topics from a remote platform, the subscribing agent has to add an additional input parameter - `all_platforms` to the pubsub subscribe method.

- *subscribe(peer, prefix, callback, bus='', all_platforms=False)* - The existing 'subscribe' method is modified to include optional keyword argument - 'all_platforms'. If 'all_platforms' is set to True, the agent is subscribing to topic from local publisher and from external platform publishers.

```
self.vip.pubsub.subscribe('pubsub', 'foo', self.on_match, all_platforms=True)
```

There is no change in the publish method pf PubSub subsystem. If all the configurations are correct and the publisher agent on the remote platform is publishing message to topic=''foo'', then the subscriber agent will start receiving those messages.

Multi-Platform RPC Communication

Multi-Platform RPC communication allows an agent on one platform to make RPC call on an agent in another platform without having to setup connection to the remote platform directly. The connection will be internally managed by the VOLTTRON platform router module. Please refer here *Multi-Platform Communication Setup*) for more details regarding setting up of Multi-Platform connections.

Calling External Platform RPC Method

If an agent in one platform wants to use an exported RPC method of an agent in another platform, it has to provide the platform name of the remote platform when using RPC subsystem call/notify method.

Here is an example:

```
self.vip.rpc.call(peer, 'say_hello', 'Bob', external_platform='platform2').get()
self.vip.rpc.notify(peer, 'ready', external_platform='platform2')
```

Here, 'platform2' is the platform name of the remote platform.

Distributed RabbitMQ Brokers

RabbitMQ allows multiple distributed RabbitMQ brokers to be connected in three different ways - with clustering, with federation and using shovel. We take advantage of these built-in plugins for multi-platform VOLTTRON communication. For more information about the differences between clustering, federation, and shovel, please refer to the RabbitMQ documentation on Distributed RabbitMQ brokers.

Clustering

Clustering connects multiple brokers residing in multiple machines to form a single logical broker. It is used in applications where tight coupling is necessary i.e, where each node shares the data and knows the state of all other nodes in the cluster. A new node can connect to the cluster through a peer discovery mechanism if configured to do so in the RabbitMQ config file. For all the nodes to be connected together in a cluster, it is necessary for them to share the same Erlang cookie and be reachable through it's DNS hostname. A client can connect to any one of the nodes in the cluster and perform any operation (to send/receive messages from other nodes etc.), the nodes will route the operation internally. In case of a node failure, clients should be able to reconnect to a different node, recover their topology and continue operation.

Note: This feature is not integrated into VOLTTRON, but we hope to support it in the future. For more detailed information about clustering, please refer to RabbitMQ documentation on the Clustering plugin.

Federation

Federation plugin is used in applications that does not require as much of tight coupling as clustering. Federation has several useful features:

- Loose coupling - The federation plugin can transmit messages between brokers (or clusters) in different administrative domains:
  - they may have different users and virtual hosts;
  - they may run on different versions of RabbitMQ and Erlang.
- WAN friendliness - They can tolerate network intermittent connectivity.
- Specificity - Not everything needs to be federated (made available to other brokers); There can be local-only components.
- Scalability - Federation does not require O(n2) connections for *n* brokers, so it scales better.

The federation plugin allows you to make exchanges and queues *federated*. A federated exchange or queue can receive messages from one or more upstreams (remote exchanges and queues on other brokers). A federated exchange can route messages published upstream to a local queue. A federated queue lets a local consumer receive messages from an upstream queue.

Before we move forward, let's define upstream and downstream servers.

- Upstream server - The node that is publishing some message of interest
- Downstream server - The node connected to a different broker that wants to receive messages from the upstream server

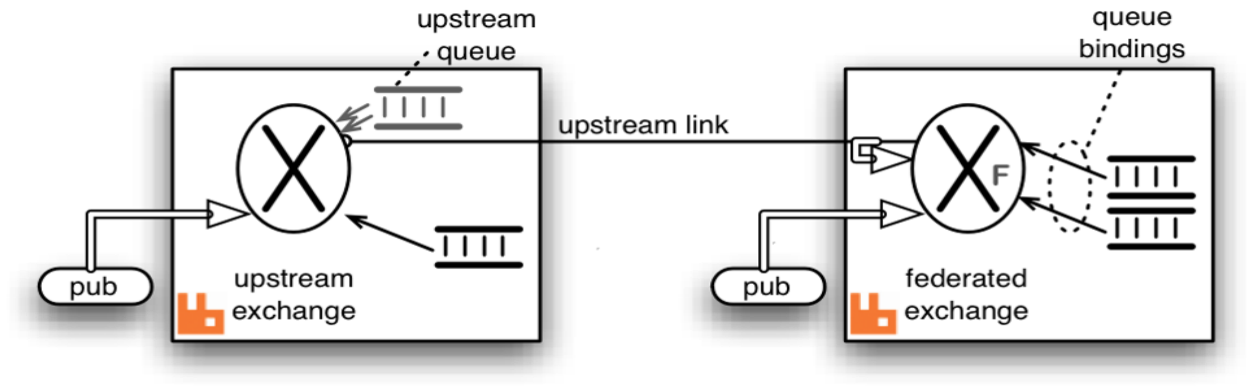
A federation link needs to be established from downstream server to the upstream server. The data flows in single direction from upstream server to downstream server. For bi-directional data flow, we would need to create federation links on both the nodes.

We can receive messages from upstream server to downstream server by either making an exchange or a queue *federated*.

For more detailed information about federation, please refer to RabbitMQ documentation Federation plugin.

Federated Exchange

When we make an exchange on the downstream server *federated*, the messages published to the upstream exchanges are copied to the federated exchange, as though they were published directly to it.



The above figure explains message transfer using federated exchange. The box on the right acts as the downstream server and the box on the left acts as the upstream server. A federation/upstream link is established between the downstream server and the upstream server by using the federation management plugin.

An exchange on the downstream server is made *federated* using federation policy configuration. The federated exchange only receives the messages for which it has subscribed. An upstream queue is created on the upstream server with a binding key same as subscription made on the federated exchange. For example, if an upstream server is publishing messages with binding key "foo" and a client on the downstream server is interested in receiving messages of the binding key "foo", then it creates a queue and binds the queue to the federated with the same binding key. This binding is sent to the upstream and the upstream queue binds to the upstream exchange with that key.

Publications to either exchange may be received by queues bound to the federated exchange, but publications directly to the federated exchange cannot be received by queues bound to the upstream exchange.

For more information about federated exchanges and different federation topologies, please read about Federated Exchanges.

Federated Queue

Federated queue provides a way of balancing load of a single queue across nodes or clusters. A federated queue lets a local consumer receive messages from an upstream queue. A typical use would be to have the same "logical" queue distributed over many brokers. Such a logical distributed queue is capable of having higher capacity than a single queue. A federated queue links to other upstream queues.

A federation or upstream link needs to be created like before and a federated queue needs to be setup on the downstream server using federation policy configuration. The federated queue will only retrieve messages when it has run out of messages locally, it has consumers that need messages, and the upstream queue has "spare" messages that are not being consumed.

For more information about federated queues, please read about Federated Queues.

Shovel

The Shovel plugin allows you to reliably and continually move messages from a source in one broker to destination in another broker. A shovel behaves like a well-written client application in that it:

- connects to it's source and destination broker
- consumes messages from the source queue
- re-publishes messages to the destination if the messages match the routing key.

The Shovel plugin uses an Erlang client under the hood. In the case of shovel, apart from configuring the hostname, port and virtual host of the remote node, we will also have to provide a list of routing keys that we want to forward to the remote node. The primary advantages of shovels are:

- Loose coupling - A shovel can move messages between brokers (or clusters) in different administrative domains: * they may have different users and virtual hosts; * they may run on different versions of RabbitMQ and Erlang.
- WAN friendliness - They can tolerate network intermittent connectivity.

Shovels are also useful in cases where one of the nodes is behind NAT. We can setup shovel on the node behind NAT to forward messages to the node outside NAT. Shovels do not allow you to adapt to subscriptions like a federation link and we need to a create a new shovel per subscription.

For more detailed information about shovel, please refer to RabbitMQ documentation on the Shovel plugin.

Agent communication to Remote RabbitMQ instance

Communication between two RabbitMQ based VOLTTRON instances must be done using SSL certificate based authentication. Non SSL based authentication will not be supported for communication to remote RabbitMQ based VOLTTRON instances. A VOLTTORN instance that wants to communicate with a remote instance should first request a SSL certificate that is signed by the remote instance. To facilitate this process there will be a web based server API for requesting, listing, approving and denying certificate requests. This api will be exposed via the MasterWebService and will be available to any RabbitMQ based VOLTTRON instance with SSL enabled. This API will be tested and used in the following agents:

- ForwarderAgent
- DataPuller
- VolttronCentralPlatform

For the following document we will assume we have two instances a local-instance and remote-volttron-instance. The remote-volttron-instance will be configured to allow certificate requests to be sent to it from the local-instance. A remote-agent running in local-instance will attempt to establish a connection to the remote-volttron-instance

Configuration

Both volttron-server and volttron-client must be configured for RabbitMQ message bus with SSL using the step described at *Installing Volttron*.

In addition the remote-volttron-instance configuration file must have a https bind-web-address specified in the instance config file. Below is an example config file with bind-web-address. Restart volttron after editing the config file

```
[volttron]
message-bus = rmq
vip-address = tcp://127.0.0.1:22916
```

(continues on next page)

(continued from previous page)

```
bind-web-address = https://volttron1:8443
instance-name = volttron1
```

By default the *bind-web-address* parameter will use the MasterWebService agent's certificate and private key. Both private and public key are necessary in order to bind the port to the socket for incoming connections. This key pair is auto generated for RabbitMQ based VOLTTRON at the time of platform startup. Users can provide a different certificate and private key to be used for the bind-web-address by specifying web-ssl-cert and web-ssl-key in the config file. Below is an example config file with the additional entries

```
[volttron]
message-bus = rmq
vip-address = tcp://127.0.0.1:22916
bind-web-address = https://volttron1:8443
instance-name = volttron1
web-ssl-cert = /path/to/cert/cert.pem
web-ssl-key = /path/to/cert/key.pem
```

Note:

- The */etc/hosts* file should be modified in order for the dns name to be used for the bound address.

remote-agent on local-instance

The *auth* subsystem of the volttron architecture is how a remote-agent on local instance will connect to the remote volttron instance.

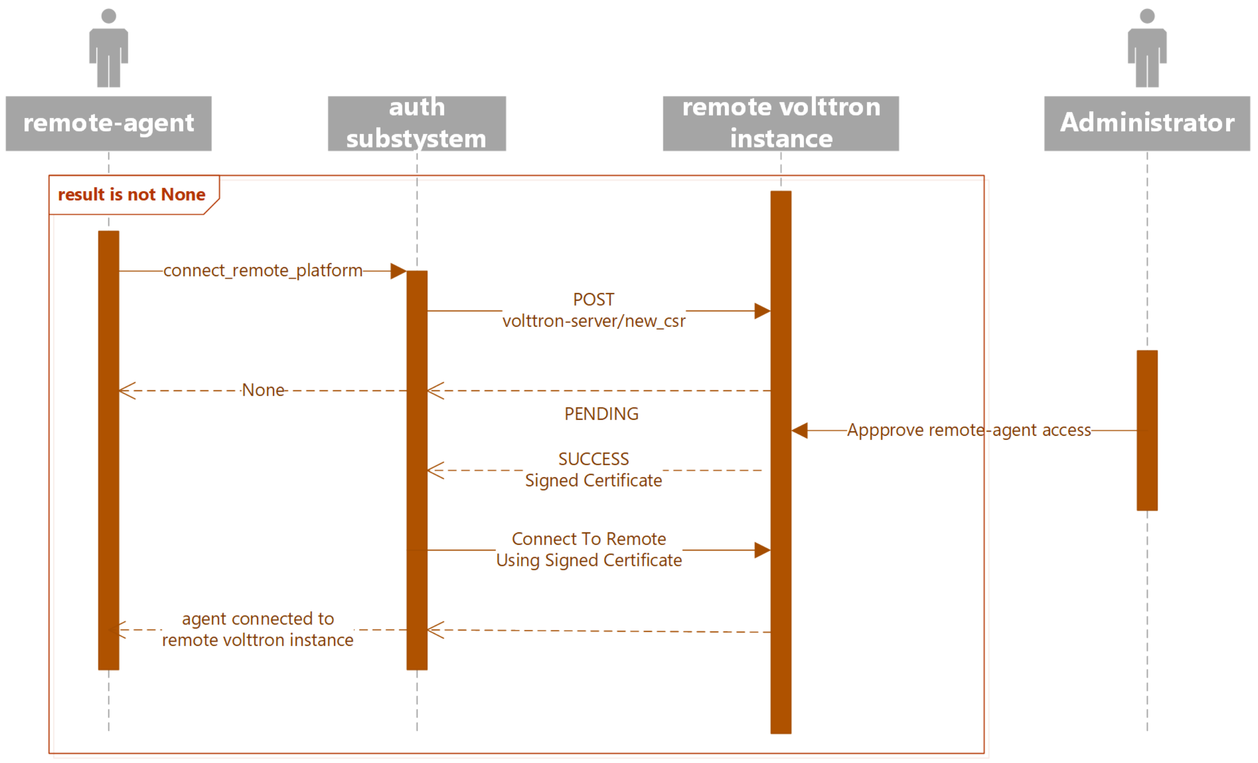
The following is a code snippet from the remote-agent to connect to the remote volttron instance.

```
...
value = self.vip.auth.connect_remote_platform(address)
```

The above function call will return an agent that connects to the remote instance only after the request is approved by an administrator of the remote instance. It is up to the agent to repeat calling *connect_remote_platform* periodically until an agent object is obtained.

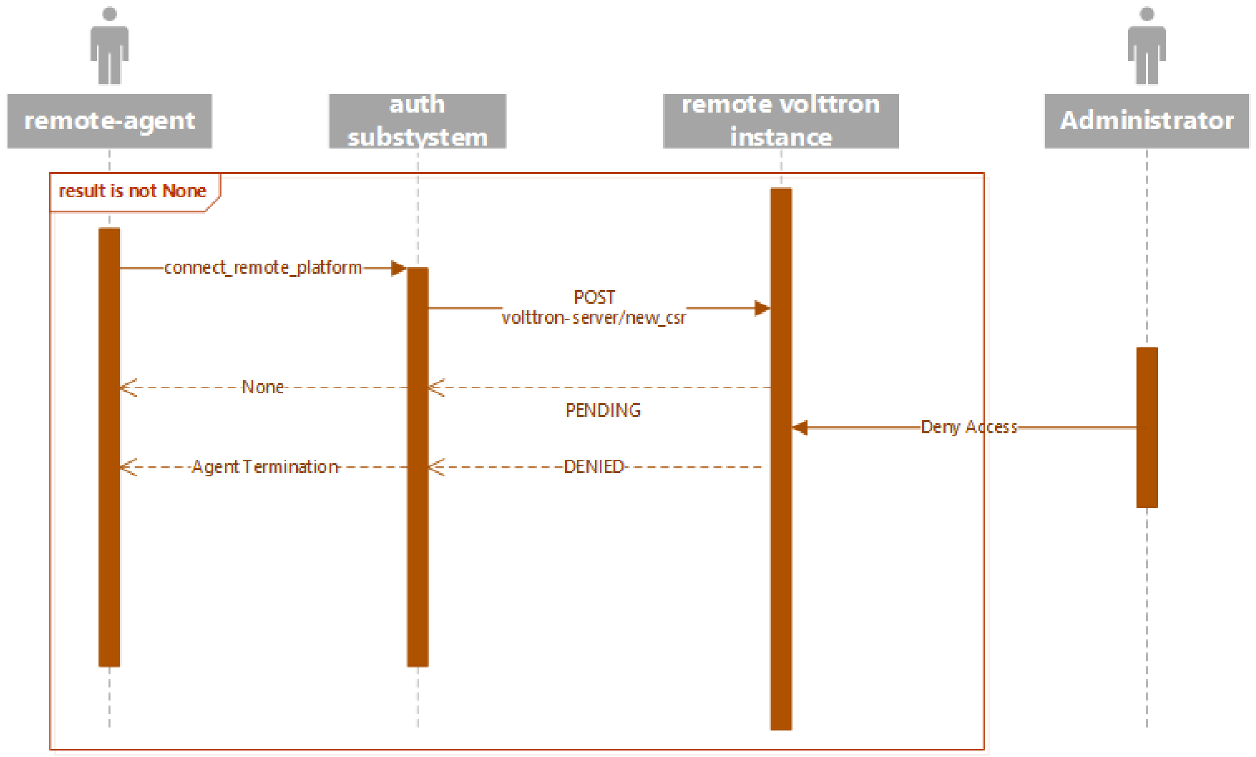
Approving a CSR Request

The following diagram shows the sequence of events when an access request is approved by the administrator of remote volttron instance. In this case, the volttron-client agent will get a Agent object that is connected to the remote instance. The diagram shows the client agent repeating the call to connect_remote_platform until the return value is not None.


remote-agent
auth substystem
remote volttron instance
Administrator
result is not None
connect_remote_platform
POST volttron-server/new_csr
None
PENDING
Appprove remote-agent access
SUCCESS Signed Certificate
Connect To Remote Using Signed Certificate
agent connected to remote volttron instance

Denying a CSR Request

The following diagram shows the sequence of events when an access request is denied by the administrator. The client agent repeats the call to connect_remote_platform until the return value is not None. When the remote instance's administrator denies a access request, the auth subsystem will raise an alert and shutdown the agent.



Follow walk-through in *Multi-Platform Multi-Bus Walk-through* for setting up different combinations of multi-bus multi-platform setup using CSR.

1.42 VOLTTRON Control

The base platform functionality focuses on the agent lifecycle, management of the platform itself, and security. This section describes how to use the commands included with VOLTTRON to configure and control the platform, agents and drivers.

1.42.1 Platform Commands

VOLTTRON files for a platform instance are stored under a single directory known as the VOLTTRON home. This home directory is set via the *VOLTTRON_HOME* environment variable and defaults to `~/.volttron`. Multiple instances of the platform may exist under the same account on a system by setting the *VOLTTRON_HOME* environment variable appropriately before executing VOLTTRON commands.

VOLTTRON's configuration file uses a modified INI format where section names are command names for which the settings in the section apply. Settings before the first section are considered global and will be used by all commands for which the settings are valid. Settings keys are long options (with or without the opening "–") and are followed by a colon (`:`) or equal (=) and then the value. Boolean options need not include the separator or value, but may specify a value of `1`, `yes`, or `true` for *true* or `0`, `no`, or `false` for *false*.

It is best practice to use the *vcfg command* prior to starting VOLTTRON for the first time to populate the configuration file for your deployment. If VOLTTRON is started without having run *vcfg*, a default config will be created in *$VOLTTRON_HOME/config*. The following is an example configuration after running *vcfg*:

where **message-bus** - Indicates message bus to be used. Valid values are `zmq` and `rmq` **instance-name** - Name of the VOLTTRON instance. This has to be unique if multiple instances need to be connected together **vip-address** - VIP address of the VOLTTRON instance. It contains the IP address and port number (default port number is 22916)

bind-web-address - Optional parameter, only needed if VOLTTRON instance needs a web interface **volttron-central-address** - Optional parameter. Web address of VOLTTRON Central agent

Note:

```
env/bin/volttron -c <config> -l volttron.log &
```

Below is a compendium of commands which can be used to operate the VOLTTRON Platform from the command line interface.

VOLTTRON Platform Command

The main VOLTTRON platform command is `volttron`, however this command is seldom run as-is. In most cases the user will want to run the platform in the background. In a limited number of cases, the user will wish to enable verbose logging. A typical command to start the platform is:

Note:

- All commands and sub-commands have help available with `-h` or `--help`
- Additional configuration files may be specified with `-c` or `-config`
- To specify a log file, use `-l` or `--log`
- The ampersand (`&`) can be added to then end of the command to run the platform in the background, freeing the open shell to be used for additional commands.

```
volttron -vv -l volttron.log &
```

volttron Optional Arguments

- **-c FILE, –config FILE** - Start the platform using the configuration from the provided FILE
- **-l FILE, –log FILE** - send log output to FILE instead of standard output/error
- **-L FILE, –log-config FILE** - Use the configuration from FILE for VOLTTRON platform logging
- **–log-level LOGGER:LEVEL** - override default logger logging level (*INFO*, *DEBUG*, *WARNING*, *ERROR*, *CRITICAL*, *NOTSET*)
- **–monitor** - monitor and log connections (implies verbose logging mode `-v`)
- **-q, –quiet** - decrease logger verboseness; may be used multiple times to further reduce logging (i.e. `-qq`)
- **-v, –verbose** - increase logger verboseness; may be used multiple times (i.e. `-vv`)
- **–verboseness LEVEL** - set logger verboseness level
- **-h, –help** - show this help message and exit
- **–version** - show program's version number and exit
- **–message-bus MESSAGE_BUS** - set message bus to be used. valid values are `zmq` and `rmq`

Note: Visit the Python 3 logging documentation for more information about logging and verboseness levels.

Agent Options

- **–autostart** - automatically start enabled agents and services after platform startup
- **–vip-address ZMQADDR** - ZeroMQ URL to bind for VIP connections
- **–vip-local-address ZMQADDR** - ZeroMQ URL to bind for local agent VIP connections
- **–bind-web-address BINDWEBADDR** - Bind a web server to the specified ip:port passed
- **–web-ca-cert CAFILE** - If using self-signed certificates, this variable will be set globally to allow requests to be able to correctly reach the webserver without having to specify verify in all calls.
- **–web-secret-key WEB_SECRET_KEY** - Secret key to be used instead of HTTPS based authentication.
- **–web-ssl-key KEYFILE** - SSL key file for using https with the VOLTTRON server
- **–web-ssl-cert CERTFILE** - SSL certificate file for using https with the VOLTTRON server
- **–volttron-central-address VOLTTRON_CENTRAL_ADDRESS** - The web address of a VOLTTRON Central install instance.
- **–volttron-central-serverkey VOLTTRON_CENTRAL_SERVERKEY** - The server key of the VOLTTRON Central being connected to.
- **–instance-name INSTANCE_NAME** - The name of the instance that will be reported to VOLTTRON Central.
- **–msgdebug** - Route all messages to an instance of the MessageDebug agent while debugging.
- **–setup-mode** - Setup mode flag for setting up authorization of external platforms.
- **–volttron-central-rmq-address VOLTTRON_CENTRAL_RMQ_ADDRESS** - The AMQP address of a VOLTTRON Central install instance
- **–agent-monitor-frequency AGENT_MONITOR_FREQUENCY** - How often should the platform check for crashed agents and attempt to restart. Units=seconds. Default=600
- **–secure-agent-users SECURE_AGENT_USERS** - Require that agents run with their own users (this requires running scripts/secure_user_permissions.sh as sudo)

Warning: Certain options alter some basic behaviors of the platform, such as *–secure-agent-users* which causes the platform to run each agent using its own Unix user to spawn the process. Please view the documentation for each feature to understand its implications before choosing to run the platform in that fashion.

volttron-ctl Commands

volttron-ctl is used to issue commands to the platform from the command line. Through *volttron-ctl* it is possible to install and removed agents, start and stop agents, manage the configuration store, get the platform status, and shutdown the platform.

In more recent versions of VOLTTRON, the commands *vctl*, *vpkg*, and *vcfg* have been added to be used as a stand-in for *volttron-ctl*, *volttron-pkg*, and *volttron-cfg* in the CLI. The VOLTTRON documentation will often use this convention.

Warning: *vctl* creates a special temporary agent to communicate with the platform with a specific VIP IDENTITY, thus multiple instances of *vctl* cannot run at the same time. Attempting to do so will result in a conflicting identity error.

Use *vctl* with one or more of the following arguments, or below sub-commands:

vctl Optional Arguments

- **-c FILE, –config FILE** - Start the platform using the configuration from the provided FILE
- **–debug** - show tracebacks for errors rather than a brief message
- **-t SECS, –timeout SECS** - timeout in seconds for remote calls (default: 60)
- **–msgdebug MSGDEBUG** - route all messages to an agent while debugging
- **–vip-address ZMQADDR** - ZeroMQ URL to bind for VIP connections
- **-l FILE, –log FILE** - send log output to FILE instead of standard output/error
- **-L FILE, –log-config FILE** - Use the configuration from FILE for VOLTTRON platform logging
- **-q, –quiet** - decrease logger verboseness; may be used multiple times to further reduce logging (i.e. `-qq`)
- **-v, –verbose** - increase logger verboseness; may be used multiple times (i.e. `-vv`)
- **–verboseness LEVEL** - set logger verboseness level (this level is a numeric level co
- **–json** - format output to json
- **-h, –help** - show this help message and exit

Commands

- **install** - install an agent from wheel

  Note: Does *NOT* package agents similarly to the *scripts/install-agent.py* script; installs agents from wheel files only

- **tag AGENT TAG** - set, show, or remove agent tag for a particular agent
- **remove AGENT** - disconnect specified agent from the platform and remove its installed agent package from *VOLTTRON_HOME*
- **peerlist** - list the peers connected to the platform
- **list** - list installed agents
- **status** - show status of installed agents
- **health AGENT** - show agent health as JSON
- **clear** - clear status of defunct agents
- **enable AGENT** - enable agent to start automatically
- **disable AGENT** - prevent agent from start automatically
- **start AGENT** - start installed agent
- **stop AGENT** - stop agent
- **restart AGENT** - restart agent
- **run PATH** - start any agent by path
- **upgrade AGENT WHEEL** - upgrade agent from wheel file

Note: Does *NOT* upgrade agents from the agent's code directory, requires agent wheel file.

- **rpc** - rpc controls
- **certs OPTIONS** - manage certificate creation
- **auth OPTIONS** - manage authorization entries and encryption keys
- **config OPTIONS** - manage the platform configuration store
- **shutdown** - stop all agents (providing the *–platform* optional argument causes the platform to be shutdown)
- **send WHEEL** - send agent and start on a remote platform
- **stats** - manage router message statistics tracking
- **rabbitmq OPTIONS** - manage rabbitmq

Note: For each command with *OPTIONS* in the description, additional options are required to make use of the command. For each, please visit the corresponding section of documentation.

- *Auth*
- *Certs*
- *Config*
- *RPC*

Note: Visit the Python 3 logging documentation for more information about logging and verboseness levels.

vctl auth Subcommands

- **add** - add new authentication record
- **add-group** - associate a group name with a set of roles
- **add-known-host** - add server public key to known-hosts file
- **add-role** - associate a role name with a set of capabilities
- **keypair** - generate CurveMQ keys for encrypting VIP connections
- **list** - list authentication records
- **list-groups** - show list of group names and their sets of roles
- **list-known-hosts** - list entries from known-hosts file
- **list-roles** - show list of role names and their sets of capabilities
- **publickey** - show public key for each agent
- **remove** - removes one or more authentication records by indices
- **remove-group** - disassociate a group name from a set of roles
- **remove-known-host** - remove entry from known-hosts file
- **remove-role** - disassociate a role name from a set of capabilities

- **serverkey** - show the serverkey for the instance
- **update** - updates one authentication record by index
- **update-group** - update group to include (or remove) given roles
- **update-role** - update role to include (or remove) given capabilities

vctl certs Subcommands

- **create-ssl-keypair** - create a SSL keypair
- **export-pkcs12** - create a PKCS12 encoded file containing private and public key from an agent. This function is may also be used to create a Java key store using a p12 file.

vctl config Subcommands

- **store AGENT CONFIG_NAME CONFIG PATH** - store a configuration file in agent's config store (parses JSON by default, use *–csv* for CSV files)
- **edit AGENT CONFIG_NAME** - edit a configuration. (opens nano by default, respects EDITOR env variable)
- **delete AGENT CONFIG_NAME** - delete a configuration from agent's config store (*–all* removes all configs for the agent)
- **list AGENT** - list stores or configurations in a store
- **get AGENT CONFIG_NAME** - get the contents of a configuration

vctl rpc Subcommands

- **code** - shows how to use RPC call in other agents
- **list** - lists all agents and their RPC methods

vpkg Commands

vpkg is the VOLTTRON command used to manage agent packages (code directories and wheel files) including creating initializing new agent code directories, creating agent wheels, etc.

vpkg Optional Arguments

- **-h, –help** - show this help message and exit
- **-l FILE, –log FILE** - send log output to FILE instead of standard output/error
- **-L FILE, –log-config FILE** - Use the configuration from FILE for VOLTTRON platform logging
- **-q, –quiet** - decrease logger verboseness; may be used multiple times to further reduce logging (i.e. `-qq`)
- **-v, –verbose** - increase logger verboseness; may be used multiple times (i.e. `-vv`)
- **–verboseness LEVEL** - set logger verboseness level

Subcommands

- **package** - Create agent package (whl) from a directory
- **init** - Create new agent code package from a template. Will prompt for additional metadata.
- **repackage** - Creates agent package from a currently installed agent.
- **configure** - Add a configuration file to an agent package

volttron-cfg Commands

volttron-cfg (*vcfg*) is a tool aimed at making it easier to get up and running with VOLTTRON and a handful of agents. Running the tool without any arguments will start a *wizard* with a walk through for setting up instance configuration options and available agents. If only individual agents need to be configured they can be listed at the command line.

Note: For a detailed description of the VOLTTRON configuration file and *vcfg* wizard, as well as example usage, view the *platform configuration* docs.

vcfg Optional Arguments

- **-h, –help** - show this help message and exit
- **-v, –verbose** - increase logger verboseness; may be used multiple times (i.e. `-vv`)
- **–vhome VHOME** Path to volttron home
- **–instance-name INSTANCE_NAME** Name of this volttron instance
- **–list-agents** - list configurable agents

```
Agents available to configure:
   listener
   master_driver
   platform_historian
   vc
   vcp
```

- **–agent AGENT [AGENT ...]** - configure listed agents
- **–rabbitmq RABBITMQ [RABBITMQ ...]** - Configure RabbitMQ for single instance, federation, or shovel either based on configuration file in YML format or providing details when prompted. Usage:

```
vcfg --rabbitmq single|federation|shovel [rabbitmq config file]
```

- **–secure-agent-users** Require that agents run with their own users (this requires running scripts/secure_user_permissions.sh as sudo)

Warning: The secure agent users significantly changes the operation of agents on the platform, please read the *secure agent users* documentation before using this feature.

1.42.2 Agent Control Commands

The VOLTTRON platform has several commands for controlling the lifecycle of agents. This page discusses how to use them, for details of operation please see *Platform Configuration*

Note: These examples assume the VOLTTRON environment has been activated

```
. env/bin/activate
```

If not activating the VOLTTRON virtual environment, add "bin/" to all commands

Agent Packaging

The *vpkg* command is used for packaging and configuring agents. It is not necessary to have the platform running to use this command. The platform uses Python Wheel for its packaging and follows the Wheel naming convention.

To create an agent package, call:

```
vpkg <Agent Dir>
```

For instance: `vpkg package examples/ListenerAgent`

The `package` command uses the *setup.py* in the agent directory to create the package. The name and version number portion of the Wheel filename come from this. The resulting wheels are created at *~/.volttron/packaged*. For example: `~/.volttron/packaged/listeneragent-3.0-py2-none-any.whl`.

Agent Configuration

Agent packages are configured with:

```
vpkg configure <AgentPackage> <ConfigFile>
```

It is suggested that this file use JSON formatting but the agent can be written to interpret any format it requires. The configuration of a particular agent is opaque to the VOLTTRON platform. The location of the agent config file is passed as an environmental variable *AGENT_CONFIG* which the provided utilities read in and pass to the agent.

An example config file passing in some parameters:

```
{

    "agentid": "listener1",
    "message": "hello"
}
```

Agent Installation and Removal

Agents are installed into the platform using:

```
vctl install <package>
```

When agents are installed onto a platform, it creates a uuid for that instance of an agent. This allows multiple instances of the same agent package to be installed on the platform.

This allows the user to refer to the agent with `--tag <tag>` instead of the uuid when issuing commands. This tag can also distinguish instances of an agent from each other.

A stopped agent can be removed with:

- `vctl remove <AGENT_UUID>`
- `vctl remove --tag <AGENT_TAG>`
- `vctl remove --name <AGENT_NAME>`

Tagging Agents

Agents can be tagged as they are installed with:

`vctl install <TAG>=<AGENT_PACKAGE>`

Agents can be tagged after installation with:

`vctl tag <AGENT_UUID> <TAG>`

Agents can be "tagged" to provide a meaningful user defined way to reference the agent instead of the uuid or the name. This allows users to differentiate between instances of agents which use the same codebase but are configured differently.

Example

A user installs two instances of the Listener Agent, tagged with *listen1* and *listen2* respectively:

```
python scripts/install-agent.py -s examples/ListenerAgent --tag listener1
python scripts/install-agent.py -s examples/ListenerAgent --tag listener2
```

`vctl status` displays:

```
  AGENT              IDENTITY             TAG       STATUS          HEALTH
a listeneragent-3.3 listeneragent-3.3_2 listener2
6 listeneragent-3.3 listeneragent-3.3_1 listener1
```

Commands which operate off an agent's UUID can optionally operate off the tag by using "–tag ". This can use wildcards to catch multiple agents at once. For example, `vctl start --tag listener*` will start both *listener1* and *listener2*.

Warning: Removal by tag and name potentially allows multiple agents to be removed at once and should be used with caution. A "-f" option is required to delete more than one agent at a time.

Agent Control

Starting and Stopping an Agent

Agent that are installed in the platform can be launched with the *start* command. By default this operates off the agent's UUID but can be used with `--tag` or `--name` to launch agents by those attributes.

This can allow multiple agents to be started at once. For instance: `vctl start --name myagent-0.1` would start all instances of that agent regardless of their uuid, tag, or configuration information.

After an agent is started, it will show up in *Agent Status* as "running" with a process id.

Similarly, `volttron-ctl stop <UUID>` can also operate off the tag and name of agent(s). After an agent is stopped, it will show an exit code of 0 in *Agent Status*

Running an agent

For testing purposes, an agent package not installed in the platform can be run by using:

```
vctl run <PACKAGE>
```

Agent Status

`vctl list` shows the agents which have been installed on the platform along with their uuid, associated *tag* and *priority*.

- *uuid* is the first column of the display and is displayed as the shorted unique portion. Using this portion, agents can be started, stopped, removed, etc.
- *AGENT* is the "name" of this agent based on the name of the wheel file which was installed. Agents can be controlled with this using `--name`.

  Note: If multiple instances of a wheel are installed they will all have the same name and can be controlled as a group.

- IDENTITY is the VIP platform identity assigned to the agent which can be used to make RPC calls, etc. with the platform
- *TAG* is a user provided tag which makes it simpler to track and refer to agents. `--tag <tag>` can used in most agent control commands instead of the UUID to control that agent or multiple agents with a pattern.
- PRI is the priority for agents which have been "enabled" using the `vctl enable` command. When enabled, agents will be automatically started in priority order along with the platform.

```
  AGENT              IDENTITY             TAG          PRI
a listeneragent-3.3 listeneragent-3.3_2 listener2
6 listeneragent-3.3 listeneragent-3.3_1 listener1
```

The `vctl status` command shows the list of installed agents and whether they are running or have exited.

```
  AGENT              IDENTITY             TAG       STATUS          HEALTH
a listeneragent-3.3 listeneragent-3.3_2 listener2 running [12872] GOOD
6 listeneragent-3.3 listeneragent-3.3_1 listener1 running [12873] GOOD
```

- *AGENT*, *IDENTITY* and *TAG* are the same as in the `vctl list` command
- *STATUS* is the current condition of the agent. If the agent is currently executing, it has "running" and the process id of the agent. If the agent is not running, the exit code is shown.
- *HEALTH* represents the current state of the agent. *GOOD* health is displayed while the agent is operating as expected. If an agent enters an error state the health will display as *BAD*

Agent Autostart

An agent can be setup to start when the platform is started with the *enable* command. This command also allows a priority to be set (0-100, default 50) so that agents can be started after any dependencies. This command can also be used with the `--tag` or `--name` options.

```
vctl enable <AGENT_UUID> <PRIORITY>
```

1.42.3 Authentication Commands

All authentication sub-commands can be viewed by entering following command.

```
vctl auth --help
```

```
optional arguments:
-h, --help            show this help message and exit
-c FILE, --config FILE
                        read configuration from FILE
--debug                     show tracbacks for errors rather than a brief message
-t SECS, --timeout SECS
                        timeout in seconds for remote calls (default: 30)
--vip-address ZMQADDR
                        ZeroMQ URL to bind for VIP connections
--keystore-file FILE        use keystore from FILE
--known-hosts-file FILE
                        get known-host server keys from FILE

subcommands:
    add                 add new authentication record
    add-group           associate a group name with a set of roles
    add-known-host      add server public key to known-hosts file
    add-role            associate a role name with a set of capabilities
    keypair             generate CurveMQ keys for encrypting VIP connections
    list                list authentication records
    list-groups         show list of group names and their sets of roles
    list-known-hosts    list entries from known-hosts file
    list-roles          show list of role names and their sets of capabilities
    publickey           show public key for each agent
    remove              removes one or more authentication records by indices
    remove-group        disassociate a group name from a set of roles
    remove-known-host   remove entry from known-hosts file
    remove-role         disassociate a role name from a set of capabilities
    serverkey           show the serverkey for the instance
    update              updates one authentication record by index
    update-group        update group to include (or remove) given roles
    update-role         update role to include (or remove) given capabilities
```

Authentication record

An authentication record consist of following parameters

```
domain []:
address []: Either a single agent identity or an array of agents identities
user_id []: Arbitrary string to identify the agent
```

(continues on next page)

(continued from previous page)

```
capabilities (delimit multiple entries with comma) []: Array of strings referring to␣
↪authorized capabilities defined by exported RPC methods
roles (delimit multiple entries with comma) []:
groups (delimit multiple entries with comma) []:
mechanism [CURVE]:
credentials []: Public key string for the agent
comments []:
enabled [True]:
```

For more details on how to create authentication record, please see section *Agent Authentication*

How to authenticate an agent to communicate with VOLTTRON platform

An administrator can allow an agent to communicate with VOLTTRON platform by creating an authentication record for that agent. An authentication record is created by using `vctl auth add` command and entering values to asked arguments.

```
vctl auth add

    domain []:
    address []:
    user_id []:
    capabilities (delimit multiple entries with comma) []:
    roles (delimit multiple entries with comma) []:
    groups (delimit multiple entries with comma) []:
    mechanism [CURVE]:
    credentials []:
    comments []:
    enabled [True]:
```

The listed fields can also be specified on the command line:

```
vctl auth add --user_id bob --credentials ABCD...
```

If any field is specified on the command line, then the interactive menu will not be used.

The simplest way of creating an authentication record is by entering the user_id and credential values. User_id is a arbitrary string for VOLTTRON to identify the agent. Credential is the encoded public key string for the agent. Create a public/private key pair for the agent and enter encoded public key for credential parameter.

```
vctl auth add

    domain []:
    address []:
    user_id []: my-test-agent
    capabilities (delimit multiple entries with comma) []:
    roles (delimit multiple entries with comma) []:
    groups (delimit multiple entries with comma) []:
    mechanism [CURVE]:
    credentials []: encoded-public-key-for-my-test-agent
    comments []:
    enabled [True]:
```

In next sections, we will discuss each parameter, its purpose and what all values it can take.

Domain:

Domain is the name assigned to locally bound address. Domain parameter is currently not being used in VOLTTRON and is placeholder for future implementation.

Address:

By specifying address, administrator can allow an agent to connect with VOLTTRON only if that agent is running on that address. Address parameter can take a string representing an IP addresses. It can also take a regular expression representing a range of IP addresses.

```
address []: 192.168.111.1
address []: /192.168.*/
```

User_id:

User_id can be any arbitrary string that is used to identify the agent by the platform. If a regular expression is used for address or credential to combine agents in an authentication record then all those agents will be identified by this user_id. It is primarily used for identifying agents during logging.

Capabilities:

Capability is an arbitrary string used by an agent to describe its exported RPC method. It is used to limit the access to that RPC method to only those agents who have that capailbity listed in their authentication record.

If administrator wants to authorize an agent to access an exported RPC method with capability of another agent, the administrator can list that capability string in this parameter. Capability parameter takes an string or an array of strings or a string representation of dictionary listing all the capabilities this agent is authorized to access. Listing capabilities here will allow this agent to access corresponding exported RPC methods of other agents.

For example, if there is an AgentA with capability enables exported RPC method and AgentB needs to access that method then AgentA's code and AgentB's authentication record would be as follow:

AgentA's capability enabled exported RPC method:

```
@RPC.export
@RPC.allow('can_call_bar')
def bar(self):
   return 'If you can see this, then you have the required capabilities'
```

AgentB's authentication record to access bar method:

```
volttron-ctl auth add

    domain []:
    address []:
    user_id []: agent-b
    capabilities (delimit multiple entries with comma) []: can_call_bar
    roles (delimit multiple entries with comma) []:
    groups (delimit multiple entries with comma) []:
    mechanism [NULL]: CURVE
    credentials []: encoded-public-key-for-agent-b
    comments []:
    enabled [True]:
```

Similarly, capability parameter can take an array of string:

```
capabilities (delimit multiple entries with comma) []: can_call_bar
capabilities (delimit multiple entries with comma) []: can_call_method1, can_call_
↪method2
```

Capabilities can also be used to restrict access to a rpc method only with certain parameter values. For example, if AgentA exposes a method bar which accepts parameter x

AgentA's capability enabled exported RPC method:

```
@RPC.export
@RPC.allow('can_call_bar')
def bar(self, x):
   return 'If you can see this, then you have the required capabilities'
```

You can restrict access to AgentA's bar method to AgentB with x=1. To add this auth entry use the vctl auth add command as show below

```
vctl auth add --capabilities '{"test1_cap2":{"x":1}}' --user_id AgentB --credential␣
↪vELQORgWOUcXo69DsSmHiCCLesJPa4-CtVfvoNHwIR0
```

auth.json file entry for the above command would be

```
{
  "domain": null,
  "user_id": "AgentB",
  "roles": [],
  "enabled": true,
  "mechanism": "CURVE",
  "capabilities": {
    "test1_cap2": {
      "x": 1
    }
  },
  "groups": [],
  "address": null,
  "credentials": "vELQORgWOUcXo69DsSmHiCCLesJPa4-CtVfvoNHwIR0",
  "comments": null
}
```

Parameter values can also be regular expressions

```
(volttron)volttron@volttron1:~/git/myvolttron$ vctl auth add
domain []:
address []:
user_id []:
capabilities (delimit multiple entries with comma) []: {'test1_cap2':{'x':'/.*'}}
roles (delimit multiple entries with comma) []:
groups (delimit multiple entries with comma) []:
mechanism [CURVE]:
credentials []: vELQORgWOUcXo69DsSmHiCCLesJPa4-CtVfvoNHwIR0
comments []:
enabled [True]:
added entry domain=None, address=None, mechanism='CURVE', credentials=u
↪'vELQORgWOUcXo69DsSmHiCCLesJPa4-CtVfvoNHwIR0', user_id='b22e041d-ec21-4f78-b32e-
↪ab7138c22373'
```

auth.json file entry for the above command would be:

```
{
  "domain": null,
  "user_id": "90f8ef35-4407-49d8-8863-4220e95974c7",
  "roles": [],
  "enabled": true,
  "mechanism": "CURVE",
  "capabilities": {
    "test1_cap2": {
      "x": "/.*"
    }
  },
  "groups": [],
  "address": null,
  "credentials": "vELQORgWOUcXo69DsSmHiCCLesJPa4-CtVfvoNHwIR0",
  "comments": null
}
```

Roles:

A role is a name for a set of capabilities. Roles can be used to grant an agent multiple capabilities without listing each capability in the in the agent's authorization entry. Capabilities can be fully utilized without roles. Roles are purely for organizing sets of capabilities.

Roles can be viewed and edited with the following commands:

- `vctl auth add-role`
- `vctl auth list-roles`
- `vctl auth remove-role`
- `vctl auth updated-role`

For example, suppose agents protect certain methods with the following capabilites: `READ_BUILDING_A_TEMP`, `SET_BUILDING_A_TEMP`, `READ_BUILDLING_B_TEMP`, and `SET_BUILDING_B_TEMP`.

These capabilities can be organized into various roles:

```
vctl auth add-role TEMP_READER READ_BUILDING_A_TEMP READ_BUILDLING_B_TEMP
vctl auth add-role BUILDING_A_ADMIN READ_BUILDING_A_TEMP SET_BUILDING_A_TEMP
vctl auth add-role BUILDING_B_ADMIN READ_BUILDING_B_TEMP SET_BUILDING_B_TEMP
```

To view these roles run `vctl auth list-roles`:

```
ROLE              CAPABILITIES
----              ------------
BUILDING_A_ADMIN  ['READ_BUILDING_A_TEMP', 'SET_BUILDING_A_TEMP']
BUILDING_B_ADMIN  ['READ_BUILDING_B_TEMP', 'SET_BUILDING_B_TEMP']
TEMP_READER       ['READ_BUILDING_A_TEMP', 'READ_BUILDLING_B_TEMP']
```

With this configuration, adding the `BUILDING_A_ADMIN` role to an agent's authorization entry implicitly grants that agent the `READ_BUILDING_A_TEMP` and `SET_BUILDING_A_TEMP` capabilities.

To add a new capabilities to an existing role:

```
vctl auth update-role BUILDING_A_ADMIN CLEAR_ALARM TRIGGER_ALARM
```

To remove a capability from a role:

```
vctl auth update-role BUILDING_A_ADMIN TRIGGER_ALARM --remove
```

Groups:

Groups provide one more layer of *grouping*. A group is a named set of roles. Like roles, groups are optional and are meant to help with organization.

Groups can be viewed and edited with the following commands:

- `vctl auth add-group`
- `vctl auth list-groups`
- `vctl auth remove-group`
- `vctl auth updated-group`

These commands behave the same as the *role* commands. For example, to further organize the capabilities in the previous section, one could create create an `ALL_BUILDING_ADMIN` group:

```
vctl auth add-group ALL_BUILDING_ADMIN BUILDING_A_ADMIN BUILDING_B_ADMIN
```

With this configuration, agents in the `ALL_BUILDING_ADMIN` group would implicity have the `BUILDING_A_ADMIN` and `BUILDING_B_ADMIN` roles. This means such agents would implicity be granted the following capabilities: `READ_BUILDING_A_TEMP`, `SET_BUILDING_A_TEMP`, `READ_BUILDLING_B_TEMP`, and `SET_BUILDING_B_TEMP`.

Mechanism:

Mechanism is the authentication method by which the agent will communicate with VOLTTRON platform. Currently VOLTTRON uses only CURVE mechanism to authenticate agents.

Credentials:

The credentials field must be an CURVE encoded public key (see *volttron.platform.vip.socket.encode_key* for method to encode public key).

```
credentials []: encoded-public-key-for-agent
```

Comments:

Comments is arbitrary string to associate with authentication record

Enabled:

TRUE of FALSE value to enable or disable the authentication record. Record will only be used if this value is True

1.43 Configuration Store

The Platform Configuration Store is a mechanism provided by the platform to facilitate the dynamic configuration of agents. The Platform Configuration Store works by informing agents of changes to their configuration store and the agent responding to those changes by updating any settings, subscriptions, or processes that are affected by the configuration of the Agent.

1.43.1 Configurations and Agents

Each agent has it's own configuration store (or just store). Agents are not given access to any other agent's store.

The existence of a store is not dependent on the existence of an agent installed on the platform.

Each store has a unique identity. Stores are matched to agents at agent runtime via the agent's VIP IDENTITY. Therefore the store for an agent is the store with the same identity as the agent's VIP IDENTITY.

When a user updates a configuration in the store the platform immediately informs the agent of the change. The platform will not send another update until the Agent finishes processing the first. The platform will send updates to the agent, one file at a time, in the order the changes were received.

1.43.2 Configuration Names

Every configuration in an agent's store has a unique name. When a configuration is added to an agent's store with the same name as an existing configuration it will replace the existing configuration. The store will remove any leading or trailing whitespace, "/", and "" from the name.

1.43.3 Configuration File Types

The configuration store will automatically parse configuration files before presenting them to an agent. Additionally, the configuration store does support storing raw data and giving to the agent unparsed. Most Agents will require the configuration to be parsed. Any Agent that requires raw data will specifically mention the requirement in its documentation.

This system removes the requirement that configuration files for an agent be in a specific format. For instance a registry configuration for a driver may be JSON instead of CSV if that is more convenient for the user. This will work as long as the JSON parses into an equivalent set of objects as an appropriate CSV file.

Currently the store supports parsing JSON and CSV files with support for more files types to come.

JSON

The store uses the same JSON parser that agents use to parse their configuration files. Therefore it supports Python style comments and must create an object or list when parsed.

```
{
    "result": "PREEMPTED", #This is a comment.
    "info": null,
    "data": {
                "agentID": "my_agent", #This is another comment.
                "taskID": "my_task"
            }
}
```

CSV

A CSV file is represented as a list of objects. Each object represents a row in the CSV file.

For instance this simple CSV file:

Table 25: Example CSV

Volttron Point Name	Modbus Register	Writable	Point Address
ReturnAirCO2	>f	FALSE	1001
ReturnAirCO2Stpt	>f	TRUE	1011

Is the equivalent to this JSON file:

```
[
    {
        "Volttron Point Name": "ReturnAirCO2",
        "Modbus Register": ">f",
        "Writable": "FALSE",
        "Point Address": "1001"
    },
    {
        "Volttron Point Name": "ReturnAirCO2Stpt",
        "Modbus Register": ">f",
        "Writable": "TRUE",
        "Point Address": "1011"
    }
]
```

1.43.4 File references

The Platform Configuration Store supports referencing one configuration file from another. If a referenced file exists the contents of that file will replace the file reference when the file is processed by the agent. Otherwise the reference will be replaced with null (or in Python, `None`).

Only configurations that are parsed by the platform (currently JSON or CSV) will be examined for references. If the file referenced is another parsed file type (JSON or CSV, currently) then the replacement will be the parsed contents of the file, otherwise it will be the raw contents of the file.

In a JSON object the name of a value will never be considered a reference.

A file reference is any value string that starts with `config://`. The rest of the string is the name of another configuration. The configuration name is converted to lower case for comparison purposes.

Consider the following configuration files named *devices/vav1.config* and *registries/vav.csv*, respectively:

```
{
    "driver_config": {"device_address": "10.1.1.5",
                      "device_id": 500},

    "driver_type": "bacnet",
    "registry_config":"config://registries/vav.csv",
    "campus": "pnnl",
    "building": "isb1",
    "unit": "vav1"
}
```

Table 26: vav.csv

Volttron Point Name	Modbus Register	Writable	Point Address
ReturnAirCO2	>f	FALSE	1001
ReturnAirCO2Stpt	>f	TRUE	1011

The resulting configuration returns when an agent asks for *devices/vav1.config*.

```
{
    "driver_config": {"device_address": "10.1.1.5",
                      "device_id": 500},

    "driver_type": "bacnet",
    "registry_config":[
                          {
                              "Volttron Point Name": "ReturnAirCO2",
                              "Modbus Register": ">f",
                              "Writable": "FALSE",
                              "Point Address": "1001"
                          },
                          {
                              "Volttron Point Name": "ReturnAirCO2Stpt",
                              "Modbus Register": ">f",
                              "Writable": "TRUE",
                              "Point Address": "1011"
                          }
                      ],
    "campus": "pnnl",
    "building": "isb1",
    "unit": "vav1"
}
```

Circular references are not allowed. Adding a file that creates a circular reference will cause that file to be rejected by the platform.

If a configuration is changed in any way and that configuration is referred to by another configuration then the agent considers the referring configuration as changed. Thus a set of configurations with references can be considered one large configuration broken into pieces for the users convenience.

Multiple configurations may all reference a single configuration. For instance, when configuring drivers in the Master Driver you may have multiple drivers reference the same registry if appropriate.

1.43.5 Modifying the Configuration Store

Currently the configuration store must be modified through the command line. See *Commandline Interface*.

Config Store Command Line Tools

Command line management of the Configuration Store is done with the *vctl config* sub-commands.

Store Configuration

To store a configuration in the Configuration Store use the *store* sub-command:

```
vctl config store <agent vip identity> <configuration name> <infile>
```

- **agent vip identity** - The agent store to add the configuration to.
- **configuration name** - The name to give the configuration in the store.
- **infile** - The file to ingest into the store.

Optionally you may specify the file type of the file. Defaults to `--json`.

- `--json` - Interpret the file as JSON.
- `--csv` - Interpret the file as CSV.
- `--raw` - Interpret the file as raw data.

Delete Configuration

To delete a configuration in the Configuration Store use the *delete* sub-command:

```
vctl config delete <agent vip identity> <configuration name>
```

- **agent vip identity** - The agent store to delete the configuration from.
- **configuration name** - The name of the configuration to delete.

To delete all configurations for an agent in the Configuration Store use `--all` switch in place of the configuration name:

```
vctl config delete <agent vip identity> --all
```

Get Configuration

To get the current contents of a configuration in the Configuration Store use the *get* sub-command:

```
vctl config get <agent vip identity> <configuration name>
```

- **agent vip identity** - The agent store to retrieve the configuration from.
- **configuration name** - The name of the configuration to get.

By default this command will return the json representation of what is stored.

- `--raw` - Return the raw version of the file.

List Configurations

To get the current list of agents with configurations in the Configuration Store use the *list* sub-command:

```
vctl config list
```

To get the current list of configurations for an agent include the Agent's VIP IDENTITY:

```
vctl config list <agent vip identity>
```

- **agent vip identity** - The agent store to retrieve the configuration from.

Edit Configuration

To edit a configuration in the Configuration Store use the *edit* sub-command:

```
vctl config edit <agent vip identity> <configuration name>
```

- **agent vip identity** - The agent store containing the configuration.
- **configuration name** - The name of the configuration to edit.

The configuration must exist in the store to be edited.

By default *edit* will try to open the file with the *nano* editor. The *edit* command will respect the *EDITOR* environment variable. You may override this with the *–editor* option.

Agent Configuration Store

This document describes the configuration store feature and explains how an agent uses it.

The configuration store enables users to store agent configurations on the platform and allows the agent to automatically retrieve them during runtime. Users may update the configurations and the agent will automatically be informed of the changes.

Compatibility

Supporting the configuration store will *not* be required by Agents, however the usage will be strongly encouraged as it should substantially improve user experience.

The previous method for configuring an agent will still be available to agents (and in some cases required), however agents can be created to only work with the configuration store and not support the old method at all.

It will be possible to create an agent to use the traditional method for configuration to establish defaults if no configuration exist in the platform configuration store.

Configuration Names and Paths

Any valid OS file path name is a valid configuration name. Any leading or trailing "/", "" and whitespace is removed by the store.

The canonical name for the main agent configuration is *config*.

The configuration subsystem remembers the case of configuration names. Name matching is case insensitive both on the Agent and platform side. Configuration names are reported to agent callbacks in the original case used when adding them to the configuration. If a new configuration is store with a different case of an existing name the new name case is used.

Configuration Ownership

Each configuration belongs to one agent and one agent only. When an agent refers to a configuration file via it's path it does not need to supply any information about its identity to the platform in the file path. The only configurations an agent has direct access to are it's own. The platform will only inform the owning agent configuration changes.

Configuration File Types

Configurations files come in three types: *json*, *csv*, and *raw*. The type of a configuration file is declared when it is added to or changed in the store.

The parser assumes the first row of every CSV file is a header.

Invalid JSON or CSV files are rejected at the time they are added to the store.

Raw files are unparsed and accepted as is.

Other parsed types may be added in the future.

Configuration File Representation to Agents

JSON

A JSON file is parsed and represented as appropriate data types to the requester.

Consider a file with the following contents:

```
{
    "result": "PREEMPTED",
    "info": null,
    "data": {
                "agentID": "my_agent",
                "taskID": "my_task"
            }
}
```

The file will be parsed and presented as a dictionary with 3 values to the requester.

CSV

A CSV file is represented as a list of objects. Each object represents a row in the CSV file.

For instance this (simplified) CSV file:

Table 27: Example CSV

Volttron Point Name	Modbus Register	Writable	Point Address
ReturnAirCO2	>f	FALSE	1001
ReturnAirCO2Stpt	>f	TRUE	1011

will be represented like this:

```
[
    {
        "Volttron Point Name": "ReturnAirCO2",
        "Modbus Register": ">f",
        "Writable": "FALSE",
        "Point Address": "1001"
    },
    {
        "Volttron Point Name": "ReturnAirCO2Stpt",
        "Modbus Register": ">f",
```

(continues on next page)

(continued from previous page)

```
        "Writable": "TRUE",
        "Point Address": "1011"
    }
]
```

Raw

Raw files are represented as a string containing the contents of the file.

File references

The *Platform Configuration Store* supports referencing one configuration file from another. If a referenced file exists the contents of that file will replace the file reference when the file is sent to the owning agent. Otherwise the reference will be replaced with None.

Only configurations that are parsed by the platform (currently "json" or "csv") will be examined for references. If the file referenced is another parsed file type (JSON or CSV, currently) then the replacement will be the parsed contents of the file.

In a JSON object the name of a value will never be considered a reference.

A file reference is any value string that starts with `config://`. The rest of the string is the path in the config store to that configuration. The config store path is converted to lower case for comparison purposes.

Consider the following configuration files named *devices/vav1.config* and *registries/vav.csv*, respectively:

```
{
    "driver_config": {"device_address": "10.1.1.5",
                      "device_id": 500},

    "driver_type": "bacnet",
    "registry_config":"config://registries/vav.csv",
    "campus": "pnnl",
    "building": "isb1",
    "unit": "vav1"
}
```

Table 28: vav.csv

Volttron Point Name	Modbus Register	Writable	Point Address
ReturnAirCO2	>f	FALSE	1001
ReturnAirCO2Stpt	>f	TRUE	1011

The resulting configuration returns when an agent asks for *devices/vav1.config*. The Python object will have the following configuration:

```
{
    "driver_config": {"device_address": "10.1.1.5",
                      "device_id": 500},

    "driver_type": "bacnet",
    "registry_config":[
                            {
```

(continues on next page)

(continued from previous page)

```
                                    "Volttron Point Name": "ReturnAirCO2",
                                    "Modbus Register": ">f",
                                    "Writable": "FALSE",
                                    "Point Address": "1001"
                                },
                                {
                                    "Volttron Point Name": "ReturnAirCO2Stpt",
                                    "Modbus Register": ">f",
                                    "Writable": "TRUE",
                                    "Point Address": "1011"
                                }
                           ],
    "campus": "pnnl",
    "building": "isb1",
    "unit": "vav1"
}
```

Circular references are not allowed. Adding a file that creates a circular reference will cause that file to be rejected by the platform.

If a file is changed in anyway (*NEW*, *UPDATE*, or *DELETE*) and that file is referred to by another file then the platform considers the referring configuration as changed. The configuration subsystem on the Agent will call every callback listening to a file or any file referring to that file either directly or indirectly.

Agent Configuration Sub System

The configuration store shall be implemented on the Agent(client) side in the form of a new subsystem called config.

The subsystem caches configurations as the platform updates the state to the agent. Changes to the cache triggered by an RPC call from the platform will trigger callbacks in the agent.

No callback methods are called until the *onconfig* phase of agent startup. A new phase to agent startup called *onconfig* will be added to the *Core ‘class. Originally it was planned to have this run after the ‘onstart* phase has completed but that is currently not possible. Ideally if an agent is using the config store feature it will not need any *onstart* methods.

When the *onconfig* phase is triggered the subsystem will retrieve the current configuration state from the platform and call all callbacks registered to a configuration in the store to the *NEW* action. No callbacks are called before this point in agent startup.

The first time callbacks are called at agent startup any callbacks subscribed to a configuration called *config* are called first.

Configuration Subsystem Agent Methods

These methods are part of the interface available to the Agent.

> **config.get(config_name=”config”)** - Get the contents of a configuration. If no name is provided the contents of the main agent configuration “config” is returned. This may not be called before *onstart* methods are called. If called during the *onstart* phase it will trigger the subsystem to initialize early but will not trigger any callbacks.
>
> **config.subscribe(callback, action=(“NEW”, “UPDATE”, “DELETE”), pattern=”*”)** - Sets up a callback for handling a configuration change. The platform will automatically update the agent when a configuration changes ultimately triggering all callbacks that match the pattern specified. The action argument describes the types of configuration change action that will trigger the callback. Possible actions are *NEW*, *UPDATE*, and *DELETE* or a tuple of any combination of actions. If no action is supplied

the callback happens for all changes. A list of actions can be supplied if desired. If no file name pattern is supplied then the callback is called for all configurations. The pattern is an regex used match the configuration name.

The callback will also be called if any file referenced by a configuration file is changed.

The signature of the callback method is `callback(config_name, action, contents)` where *file_name* is the file that triggered the callback, action is the action that triggered the callback, and contents are the new contents of the configuration. Contents will be `None` on a *DELETE* action. All callbacks registered for *NEW* events will be called at agent startup after all *osntart* methods have been called. Unlike pubsub subscriptions, this may be called at any point in an agent's lifetime.

config.unsubscribe(callback=None, config_name_pattern=None) - Unsubscribe from configuration changes. Specifying a callback only will unsubscribe that callback from all config name patterns they have been bound to. If a pattern only is specified then all callbacks bound to that pattern will be removed. Specifying both will remove that callback from that pattern. Calling with no arguments will remove all subscriptions.

config.unsubscribe_all() - Unsubscribe from all configuration changes.

config.set(config_name, contents, trigger_callback=False) - Set the contents of a configuration. This may not be called before *onstart* methods are called. This can be used by an agent to store agent state across agent installations. This will *NOT* trigger any callbacks unless *trigger_callback* is set to *True*. To prevent deadlock with the platform this method may not be called from a configuration callback function. Doing so will raise a *RuntimeError* exception.

This will not modify the local configuration cache the Agent maintains. It will send the configuration change to the platform and rely on the subsequent *update_config* call.

config.delete(config_name, trigger_callback=False) - Remove the configuration from the store. This will *NOT* trigger any callbacks unless trigger_callback is *True*. To prevent deadlock with the platform this method may not be called from a configuration callback function. Doing so will raise a *RuntimeError* exception.

config.list( ) - Returns a list of configuration names.

config.set_default(config_name, contents, trigger_callback=False) - Set a default value for a configuration. *DOES NOT* modify the platform's configuration store but creates a default configuration that is used for agent configuration callbacks if the configuration does not exist in the store or the configuration is deleted from the store. The callback will only be triggered if *trigger_callback* is true and the configuration store subsystem on the agent is not aware of a configuration with that name from the platform store.

Typically this will be called in the *__init__* method of an agent with the parsed contents of the packaged configuration file. This may not be called from a configuration callback. Doing so will raise a *RuntimeError*.

config.delete_default(config_name, trigger_callback=False) - Delete a default value for a configuration. This method is included for for completeness and is unlikely to be used in agent code. This may not be called from a configuration callback. Doing so will raise a *RuntimeError*.

Configuration Sub System RPC Methods

These methods are made available on each agent to allow the platform to communicate changes to a configuration to the affected agent. As these methods are not part of the exposed interface they are subject to change.

config.update(config_name, action, contents=None, trigger_callback=True) - called by the platform when a configuration was changed by some method other than the Agent changing the configuration itself. Trigger callback tells the agent whether or not to call any callbacks associate with the configuration.

Notes on trigger_callback

As the configuration subsystem calls all callbacks in the *onconfig* phase and none are called beforehand the *trigger_callback* setting is effectively ignored if an agent sets a configuration or default configuration before the end of the *onstart* phase.

Platform Configuration Store

The platform configuration store handles the storage and maintenance of configuration states on the platform.

As these methods are not part of the exposed interface they are subject to change.

Platform RPC Methods

Methods for Agents

Agent methods that change configurations do not trigger any callbacks unless trigger_callback is True.

set_config(config_name, contents, trigger_callback=False) - Change/create a configuration file on the platform.

get_configs() - Get all of the configurations for an Agent.

delete_config(config_name, trigger_callback=False) - Delete a configuration.

Methods for Management

manage_store_config(identity, config_name, contents, config_type=”raw”) - Change/create a configuration on the platform for an agent with the specified identity

manage_delete_config(identity, config_name) - Delete a configuration for an agent with the specified identity. Calls the agent’s update_config with the action *DELETE_ALL* and no configuration name.

manage_delete_store(identity) - Delete all configurations for a VIP IDENTITY.

manage_list_config(identity) - Get a list of configurations for an agent with the specified identity.

manage_get_config(identity, config_name, raw=True) - Get the contents of a configuration file. If raw is set to *True* this function will return the original file, otherwise it will return the parsed representation of the file.

manage_list_stores() - Get a list of all the agents with configurations.

Direct Call Methods

Services local to the platform who wish to use the configuration store may use two helper methods on the agent class created for this purpose. This allows the auth service to use the config store before the router is started.

delete(self, identity, config_name, trigger_callback=False) - Same as functionality as *delete_config*, but the caller must specify the identity of the config store.

store(self, identity, config_name, contents, trigger_callback=False) - Same functionality as set_config, but the caller must specify the identity of the config store.

Command Line Interface

The command line interface will consist of a new commands for the *volttron-ctl* program called *config* with four sub-commands called *store*, *delete*, *list*, *get*. These commands will map directly to the management RPC functions in the previous section.

Disabling the Configuration Store

Agents may optionally disable support for the configuration store by passing `enable_store=False` to the *__init__* method of the Agent class. This allows temporary agents to not spin up the subsystem when it is not needed. Platform service agents that do not yet support the configuration store and the temporary agents used by *volttron-ctl* will set this value.

1.44 Platform Security

There are various security-related topics throughout VOLTTRON's documentation. This is a quick roadmap for finding security documentation.

A core component of VOLTTRON is its *message bus*. The security of this message bus is crucial to the entire system. The *VOLTTRON Interconnect Protocol* provides communication over the message bus.

VIP was built with security in mind from the ground up. VIP uses encrypted channels and enforces agent *authentication* by default for all network communication. VIP's *authorization* mechanism allows system administrators to limit agent capabilities with fine granularity.

Even with these security mechanisms built into VOLTTRON, it is important for system administrators to *harden VOLTTRON's underlying OS*.

The VOLTTRON team has engaged with PNNL's Secure Software Central team to create a threat profile document. You can read about the threat assessment findings and how the VOLTTRON team is addressing them here: SSC Threat Profile

Additional documentation related to VIP authentication and authorization is available here:

1.44.1 Running Agents as Unix Users

This VOLTTRON feature will cause the platform to create a new, unique Unix user(agent users) on the host machine for each agent installed on the platform. This user will have restricted permissions for the file system, and will be used to run the agent process.

Warning: The Unix user starting the VOLTTRON platform will be given limited sudo access to create and delete agent users.

Since this feature requires system level changes (e.g. sudo access, user creation, file permission changes), the initial step needs to be run as root or user with *sudo* access. This can be a user other than Unix user used to run the VOLTTRON platform.

All files and folder created by the VOLTTRON process in this mode would not have any access to others by default. Permission for Unix group others would be provided to specific files and folder based on VOLTTRON process requirement.

It is recommended that you use a new *VOLTTRON_HOME* to run VOLTTRON in secure mode. Converting a existing VOLTTRON instance to secure mode is also possible but would involve some manual changes. Please see the section *Porting existing volttron home to secure mode*.

Note: VOLTTRON has to be bootstrapped as prerequisite to running agents as unique users.

Setup agents to run using unique users

1. **This feature requires acl to be installed.**

   Make sure the *acl* library is installed. If you are running on a Docker image *acl* might not be installed by default.

   ```
   apt-get install acl
   ```

2. Agents now run as a user different from VOLTTRON platform user. Agent users should have *read* and *execute* permissions to all directories in the path to the Python executable used by VOLTTRON. For example, if VOLTTRON is using a virtual environment, then agent users should have *read* permissions to *<ENV_DIR>/bin/python* and *read and execute* permission to all the directories in the path *<ENV_DIR>/bin*. This can be achieved by running:

   ```
   chmod -R o+rx <ENV_DIR>/bin
   ```

3. **Run scripts/secure_user_permissions.sh as root or using sudo**

   This script *MUST* be run as root or using *sudo*. This script gives the VOLTTRON platform user limited *sudo* access to create a new Unix user for each agent. All users created will be of the format *volttron_<timestamp>*.

   This script prompts for:

   a. **volttron platform user** - Unix user who would be running the VOLTTRON platform. This should be an existing Unix user. On a development machine this could be the Unix user you logged in as to check out VOLTTRON source

   b. **VOLTTRON_HOME directory** - The absolute path of the volttron home directory.

   c. **Volttron instance name if VOLTTRON_HOME/config does not exist** -

      If the *VOLTTRON_HOME/config* file exists then instance name is obtained from that config file. If not, the user will be prompted for an instance name. *volttron_<instance_name> MUST* be a 23 characters or shorter containing only characters valid as Unix user names.

   This script will create necessary entries in */etc/sudoers.d/volttron* to allow the VOLTTRON platform user to create and delete agent users, the VOLTTRON agent group, and run any non-sudo command as the agent users.

   This script will also create *VOLTTRON_HOME* and the config file if given a new VOLTTRON home directory when prompted.

4. **Continue with VOLTTRON bootstrap and setup as normal** - point to the *VOLTTRON_HOME* that you provided in step 2.

5. **On agent install (or agent start for existing agents)** - a unique agent user(Unix user) is created and the agent is started as this user. The agent user name is recorded in *USER_ID* file under the agent install directory (*VOLTTRON_HOME/agents/<agent-uuid>/USER_ID*). Subsequent agent restarts will read the content of the *USER_ID* file and start the agent process as that user.

6. **On agent uninstall** - The agent user is deleted and the agent install directory is deleted.

Creating new Agents

In this secure mode, agents will only have read write access to the agent-data directory under the agent install directory - *VOLTTRON_HOME/agents/<agent_uuid>/<agent_name>/<agent_name>.agent-data*. Attempting to write in any other folder under *VOLTTRON_HOME* **will result in permission errors**.

Changes to existing agents in secure mode

Due to the above change, **SQL historian has been modified to create its database by default under its agent-data directory** if no path is given in the config file. If providing a path to the database in the config file, please provide a directory where agent will have write access. This can be an external directory for which agent user (*recorded in VOLTTRON_HOME/agents/<agent-uuid>/USER_ID*) has *read, write, and execute* access.

Porting existing VOLTTRON home to secure mode

When running *scripts/secure_users_permissions.sh* you will be prompted for a *VOLTTRON_HOME* directory. If this directory exists and contains a volttron config file, the script will update the file locations and permissions of existing VOLTTRON files including installed directories. However this step has the following limitations:

1. **You will NOT be able to revert to insecure mode once the changes are done.** - Once setup is complete, changing the config file manually to make parameter *secure-agent-users* to *False*, may result inconsistent VOLTTRON behavior
2. The VOLTTRON process and all agents have to be restarted to take effect
3. **Agents can only to write to its own agent-data dir.** - If your agents writes to any directory outside *$VOLTTRON_HOME/agents/<agent-uuid>/<agent-name>/agent-name.agent-data* move existing files and update the agent configuration such that the agent writes to the *agent-name.agent-data* dir. For example, if you have a *SQLHistorian* which writes a *.sqlite* file to a subdirectory under *VOLTTRON_HOME* that is not *$VOLTTRON_HOME/agents/<agent-uuid>/<agent-name>/agent-name.agent-data* this needs to be manually updated.

1.44.2 Key Stores

Warning: Most VOLTTRON users should not need to directly interact with agent key stores. These are notes for VOLTTRON platform developers. This is not a stable interface and the implementation details are subject to change.

Each agent has its own encryption key-pair that is used to *authenticate* itself with the VOLTTRON platform. A key-pair comprises a public key and a private (secret) key. These keys are saved in a "key store", which is implemented by the `KeyStore class`. Each agent has its own key store.

Key Store Locations

There are two main locations key stores will be saved. Installed agents' key stores are in the the agent's data directory:

```
$VOLTTRON_HOME/agents/<AGENT_UUID>/<AGENT_NAME>/keystore.json
```

Agents that are not installed, such as platform services and stand-alone agents, store their key stores here:

```
$VOLTTRON_HOME/keystores/<VIP_IDENTITY>/keystore.json
```

Generating a Key Store

Agents automatically retrieve keys from their key store unless both the `publickey` and `secretkey` parameters are specified when the agent is initialized. If an agent's key store does not exist it will automatically be generated upon access.

Users can generate a key pair by running the following command:

```
vctl auth keypair
```

1.44.3 Known Hosts File

Before an agent can connect to a VOLTTRON platform that agent must know the platform's VIP address and public key (known as the *server key*). It can be tedious to manually keep track of server keys and match them with their corresponding addresses.

The purpose of the known-hosts file is to save a mapping of platform addresses to server keys. This way the user only has to specify a server key one time.

Saving a Server Key

Suppose a user wants to connect to a platform at `192.168.0.42:22916`, and the platform's public key is `uhjbCUm3kT5QWj5Py9w0XZ7c1p6EP8pdo4Hq4dNEIiQ`. To save this address-to-server-key association, the user can run:

```
volttron-ctl auth add-known-host --host 192.168.0.42:22916 --serverkey␣
↪uhjbCUm3kT5QWj5Py9w0XZ7c1p6EP8pdo4Hq4dNEIiQ
```

Now agents on this system will automatically use the correct server key when connecting to the platform at `192.168.0.42:22916`.

Server Key for Local Platforms

When a platform starts it automatically adds its public key to the known-hosts file. Thus agents connecting to the local VOLTTRON platform (on the same system and using the same `$VOLTTRON_HOME`) will automatically be able to retrieve the platform's public key.

Know-Host-File Details

Note: The following details regarding the known-hosts file are subject to change. These notes are primarily for developers, but the may be helpful if troubleshooting an issue. **The known-hosts file should not be edited directly.**

File Location

The known-hosts-file is stored at `$VOLTTRON_HOME/known_hosts`.

File Contents

Here are the contents of an example known-hosts file:

```
{
    "@": "FSG7LHhy3v8tdNz3gK35G6-oxUcyln54pYRKu5fBJzU",
    "127.0.0.1:22916": "FSG7LHhy3v8tdNz3gK35G6-oxUcyln54pYRKu5fBJzU",
    "127.0.0.2:22916": "FSG7LHhy3v8tdNz3gK35G6-oxUcyln54pYRKu5fBJzU",
    "127.0.0.1:12345": "FSG7LHhy3v8tdNz3gK35G6-oxUcyln54pYRKu5fBJzU",
    "192.168.0.42:22916": "uhjbCUm3kT5QWj5Py9w0XZ7c1p6EP8pdo4Hq4dNEIiQ"
}
```

The first four entries are for the local platform. (They were automatically added when the platform started.) The first entry with the `@` key is for IPC connections, and the entries with the `127.0.0.*` keys are for local TCP connections. Note that a single VOLTTRON platform can bind to multiple TCP addresses, and each address will be automatically added to the known-hosts file. The last entry is for a remote VOLTTRON platform. (It was added in the *Saving a Server Key* section.)

1.45 Troubleshooting

This section contains individual documents intended to help the user troubleshoot various platform components. For troubleshooting of individual agents and drivers please refer to the corresponding document for each.

1.45.1 RabbitMQ Troubleshooting

Check the status of the federation connection

```
$RABBITMQ_HOME/sbin/rabbitmqctl eval 'rabbit_federation_status:status().'
```

If everything is properly configured, then the status is set to *running*. If not look for the error status. Some of the typical errors are:

a. **failed_to_connect_using_provided_uris** - Check if RabbitMQ user is created in downstream server node. Refer to step 3-b of federation setup

b. **unknown ca** - Check if the root CAs are copied to all the nodes correctly. Refer to step 2 of federation setup

c. **no_suitable_auth_mechanism** - Check if the AMPQ/S ports are correctly configured.

Check the status of the shovel connection

```
RABBITMQ_HOME/sbin/rabbitmqctl eval 'rabbit_shovel_status:status().'
```

If everything is properly configured, then the status is set to *running*. If not look for the error status. Some of the typical errors are:

a. **failed_to_connect_using_provided_uris** - Check if RabbitMQ user is created in subscriber node. Refer to step 3-b of shovel setup

b. **unknown ca** - Check if the root CAs are copied to remote servers correctly. Refer to step 2 of shovel setup

c. **no_suitable_auth_mechanism** - Check if the AMPQ/S ports are correctly configured.

Check the RabbitMQ logs for any errors

```
tail -f <volttron source dir>/rabbitmq.log
```

Rabbitmq startup hangs

a. Check for errors in the RabbitMQ log. There is a *rabbitmq.log* file in your VOLTTRON source directory that is a symbolic link to the RabbitMQ server logs.

b. Check for errors in syslog (*/var/log/syslog* or */var/log/messages*)

c. If there are no errors in either of the logs, restart the RabbitMQ server in foreground and see if there are any errors written on the console. Once you find the error you can kill the process by entering *Ctl+C*, fix the error and start rabbitmq again using `./start-rabbitmq` from VOLTTRON source directory.

   ```
   ./stop-volttron
   ./stop-rabbitmq
   @RABBITMQ_HOME/sbin/rabbitmq-server
   ```

SSL trouble shooting

There are few things that are essential for SSL certificates to work right.

a. Please use a unique common-name for CA certificate for each VOLTTRON instance. This is configured under *certificate-data* in the *rabbitmq_config.yml* or if no yml file is used while configuring a VOLTTRON single instance (using `vcfg --rabbitmq single`). Certificate generated for agent will automatically get agent's VIP identity as the certificate's common-name

b. The host name in the SSL certificate should match hostname used to access the server. For example, if the fully qualified domain name was configured in the *certificate-data*, you should use the fully qualified domain name to access RabbitMQ's management url.

c. Check if your system time is correct especially if you are running virtual machines. If the system clock is not right, it could lead to SSL certificate errors

DataMover troubleshooting

If output from *volttron.log* is not as expected check for `{'alert_key': 'historian_not_publishing'}` in the callee node's *volttron.log*. Most likely cause is the historian is not running properly or credentials between caller and callee nodes was not set properly.

1.46 Applications

These resources summarize the use of the sample applications that have been created by VOLTTRON users. For detailed information on these applications, refer to the report Transactional Network Platform.

Note, as of VOLTTRON 4.0, applications are now in their own repository at: https://github.com/VOLTTRON/volttron-applications

1.46.1 Acquiring Third Party Agent Code

Add the volttron-applications repository under the volttron/applications directory by using following command:

> git subtree add –prefix applications https://github.com/VOLTTRON/volttron-applications.git develop –squash

Passive Automated Fault Detection and Diagnostic Agent

The Passive Automated Fault Detection and Diagnostic (Passive AFDD) agent is used to identify problems in the operation and performance of air-handling units (AHUs) or packaged rooftop units (RTUs). Air-side economizers modulate controllable dampers to use outside air to cool instead of (or to supplement) mechanical cooling, when outdoor-air conditions are more favorable than the return-air conditions. Unfortunately, economizers often do not work properly, leading to increased energy use rather than saving energy. Common problems include incorrect control strategies, diverse types of damper linkage and actuator failures, and out-of-calibration sensors. These problems can be detected using sensor data that is normally used to control the system.

The Passive AFDD requires the following data fields to perform the fault detection and diagnostics:

- Outside-air temperature
- Return-air temperature
- Mixed-air temperature
- Outside-air damper position/signal
- Supply fan status
- Mechanical cooling status
- Heating status.

The AFDD supports both real-time data via a Modbus or BACnet device, or input of data from a csv style text document.

The following section describes how to configure the Passive AFDD agent, methods for data input (real-time data from a device or historical data in a comma separated value formatted text file), and launching the Passive AFDD agent.

Note: A proactive version of the Passive AFDD exists as a PNNL application (AFDDAgent). This application requires active control of the RTU for fault detection and diagnostics to occur. The Passive AFDD was created to allow more users a chance to run diagnostics on their HVAC equipment without the need to actively modify the controls of the system.

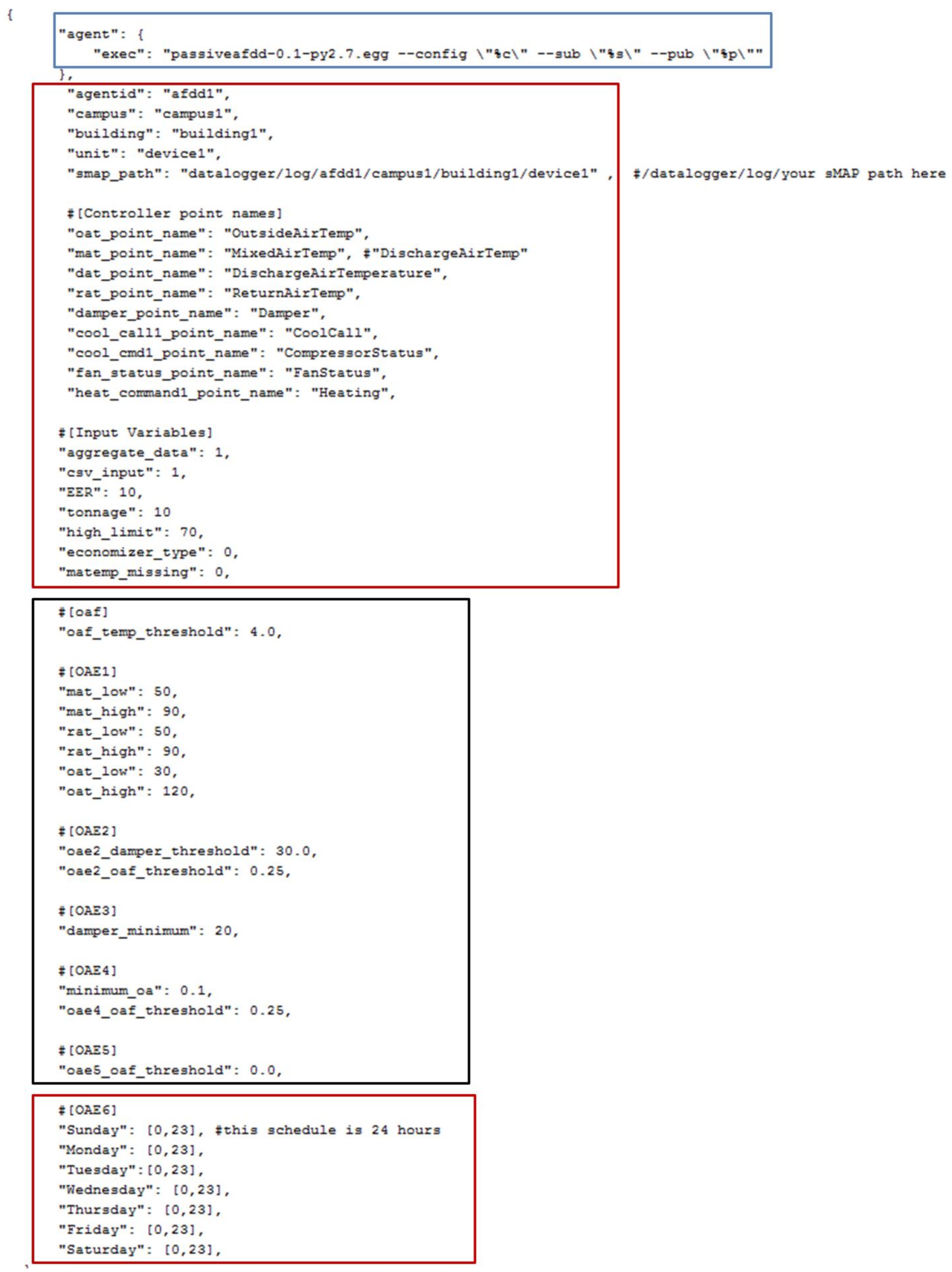
Configuring the Passive AFDD Agent

Before launching the Passive AFDD agent, several parameters require configuration. The AFDD utilizes the same JSON style configuration file used by the Actuator, Listener, and Weather agents. The threshold parameters used for the fault detection algorithms are pre-configured and will work well for most RTUs or AHUs. Figure 1 shows an example configuration file for the AFDD agent.

The parameters boxed in black (in Figure 1) are the pre-configured fault detection thresholds; these do not require any modification to run the Passive AFDD agent. The parameters in the example configuration that are boxed in red will require user input. The following list describes each user configurable parameter and their possible values:

- **agentid** – This is the ID used when making schedule, set, or get requests to the Actuator agent; usually a string data type.
- **campus** – Campus name as configured in the sMAP driver. This parameter builds the device path that allows the Actuator agent to set and get values on the device; usually a string data type.

- **building** – Building name as configured in the sMAP driver. This parameter builds the device path that allows the Actuator agent to set and get values on the device; usually a string data type.
- **unit** – Device name as configured in the sMAP driver. This parameter builds the device path that allows the Actuator agent to set and get values on the device; usually a string data type. Note: The campus, building, and unit parameters are used to build the device path (campus/building/unit). The device path is used for communication on the message bus.
- **controller point names** – When using real-time communication, the Actuator agent identifies what registers or values to set or get by the point name you specify. This name must match the "Point Name" given in the Modbus registry file, as specified in VOLTTRON Core Services.
- **aggregate_data** – When using real-time data sampled at an interval of less than 1 hour or when inputting data via a csv file sampled at less than 1 hour intervals, set this flag to "1." Value should be an integer or floating-point number (i.e., 1 or 1.0)
- **csv_input** – Flag to indicate if inputting data from a csv text file. Set to "0" for use with real-time data from a device or "1" if data is input from a csv text file. It should be an integer or floating point number (i.e., 1 or 1.0)
- **EER** – Energy efficiency ratio for the AHU or RTU. It should be an integer or floating-point number (i.e., 10 or 10.0)
- **tonnage** – Cooling capacity of the AHU or RTU in tons of cooling. It should be an integer or floating-point number (i.e., 10 or 10.0)
- **economizer_type** – This field indicates what type of economizer control is used. Set to "0" for differential dry-bulb control or to "1" for high limit dry-bulb control. It should be an integer or floating-point number.
- **high_limit** – If the economizer is using high-limit dry-bulb control, this value indicates what the outside-air temperature high limit should be. The input should be floating-point number (i.e., 60.0)
- **matemp_missing** – Flag used to indicate if the mixed-air temperature is missing for this system. If utilizing csv data input, simply set this flag to "1" and replace the mixed-air temperature column with discharge-air temperature data. If using real-time data input, change the field "mat_point_name" under Point Names section to the point name indicating the discharge-air temperature. It should be an integer or floating-point number (i.e., 1 or 1.0)
- **OAE6** – This section contains the schedule information for the AHU or RTU. The default is to indicate a 24-hour schedule for each day of the week. To modify this, change the numbers in the bracketed list next to the corresponding day with which you are making operation schedule modifications. For example: "Saturday": [0,0] (This indicates the system is off on Saturdays).



```
{
    "agent": {
        "exec": "passiveafdd-0.1-py2.7.egg --config \"%c\" --sub \"%s\" --pub \"%p\""
    },
     "agentid": "afdd1",
     "campus": "campus1",
     "building": "building1",
     "unit": "device1",
     "smap_path": "datalogger/log/afdd1/campus1/building1/device1" ,  #/datalogger/log/your sMAP path here

     #[Controller point names]
     "oat_point_name": "OutsideAirTemp",
     "mat_point_name": "MixedAirTemp", #"DischargeAirTemp"
     "dat_point_name": "DischargeAirTemperature",
     "rat_point_name": "ReturnAirTemp",
     "damper_point_name": "Damper",
     "cool_call1_point_name": "CoolCall",
     "cool_cmd1_point_name": "CompressorStatus",
     "fan_status_point_name": "FanStatus",
     "heat_command1_point_name": "Heating",

    #[Input Variables]
    "aggregate_data": 1,
    "csv_input": 1,
    "EER": 10,
    "tonnage": 10
    "high_limit": 70,
    "economizer_type": 0,
    "matemp_missing": 0,

    #[oaf]
    "oaf_temp_threshold": 4.0,

    #[OAE1]
    "mat_low": 50,
    "mat_high": 90,
    "rat_low": 50,
    "rat_high": 90,
    "oat_low": 30,
    "oat_high": 120,

    #[OAE2]
    "oae2_damper_threshold": 30.0,
    "oae2_oaf_threshold": 0.25,

    #[OAE3]
    "damper_minimum": 20,

    #[OAE4]
    "minimum_oa": 0.1,
    "oae4_oaf_threshold": 0.25,

    #[OAE5]
    "oae5_oaf_threshold": 0.0,

    #[OAE6]
    "Sunday": [0,23], #this schedule is 24 hours
    "Monday": [0,23],
    "Tuesday":[0,23],
    "Wednesday": [0,23],
    "Thursday": [0,23],
    "Friday": [0,23],
    "Saturday": [0,23],
```

Figure 1. Example Passive AFDD Agent Configuration File

Launching the Passive AFDD Agent

The Passive AFDD agent performs passive diagnostics on AHUs or RTUs, monitors and utilizes sensor data, but does not actively control the devices. Therefore, the agent does not require interaction with the Actuator agent. Steps for launching the agent are provided below.

In a terminal window, enter the following commands:

1. Run *pack_install* script on Passive AFDD agent:

```
$ . scripts/core/pack_install.sh applications/PassiveAFDD applications/PassiveAFDD/
↪passiveafdd.launch.json passive-afdd
```

Upon successful completion of this command, the terminal output will show the install directory, the agent UUID (unique identifier for an agent; the UUID shown in red is only an example and each instance of an agent will have a different UUID), and the agent name (blue text):

```
Installed /home/volttron-user/.volttron/packaged/passiveafdd-0.1-py2-none-any.whl as␣
↪5df00517-6a4e-4283-8c70-5f0759713c64 passiveafdd-0.1
```

2. Start the agent:

```
$ vctl start --tag passive-afdd
```

3. Verify that the agent is running:

```
$ vctl status
$ tail -f volttron.log
```

If changes are made to the Passive AFDD agent's configuration file after the agent is launched, it is necessary to stop and reload the agent. In a terminal, enter the following commands:

```
$ vctl stop --tag passive-afdd
$ vctl remove --tag passive-afdd
```

Then re-build and start the updated agent.

When the AFDD agent is monitoring a device via the message bus, the agent relies on the periodic data published from the sMAP driver. The AFDD agent then aggregates this data each hour and performs the diagnostics on the average hourly data. The result is written to a csv text file, which is appended if the file already exists. This file is in a folder titled "Results" under the (`<project directory>/applications/pnnl/PassiveAFDD/passiveafdd`) directory. Below is a key that describes how to interpret the diagnostic results:

Diagnostic Code	Code Message
AFDD-1 (Temperature Sensor Fault)	
20	No faults detected
21	Temperature sensor fault
22	Conditions not favorable for diagnostic
23	Mixed-air temperature outside of expected range
24	Return-air temperature outside of expected range
25	Outside-air temperature outside of expected range
27	Missing data necessary for fault detection
29	Unit is off (No Fault)
AFDD-2 (RTU Economizing When it Should)	
30	No faults detected

Continue

Table 29 – continued from previous page

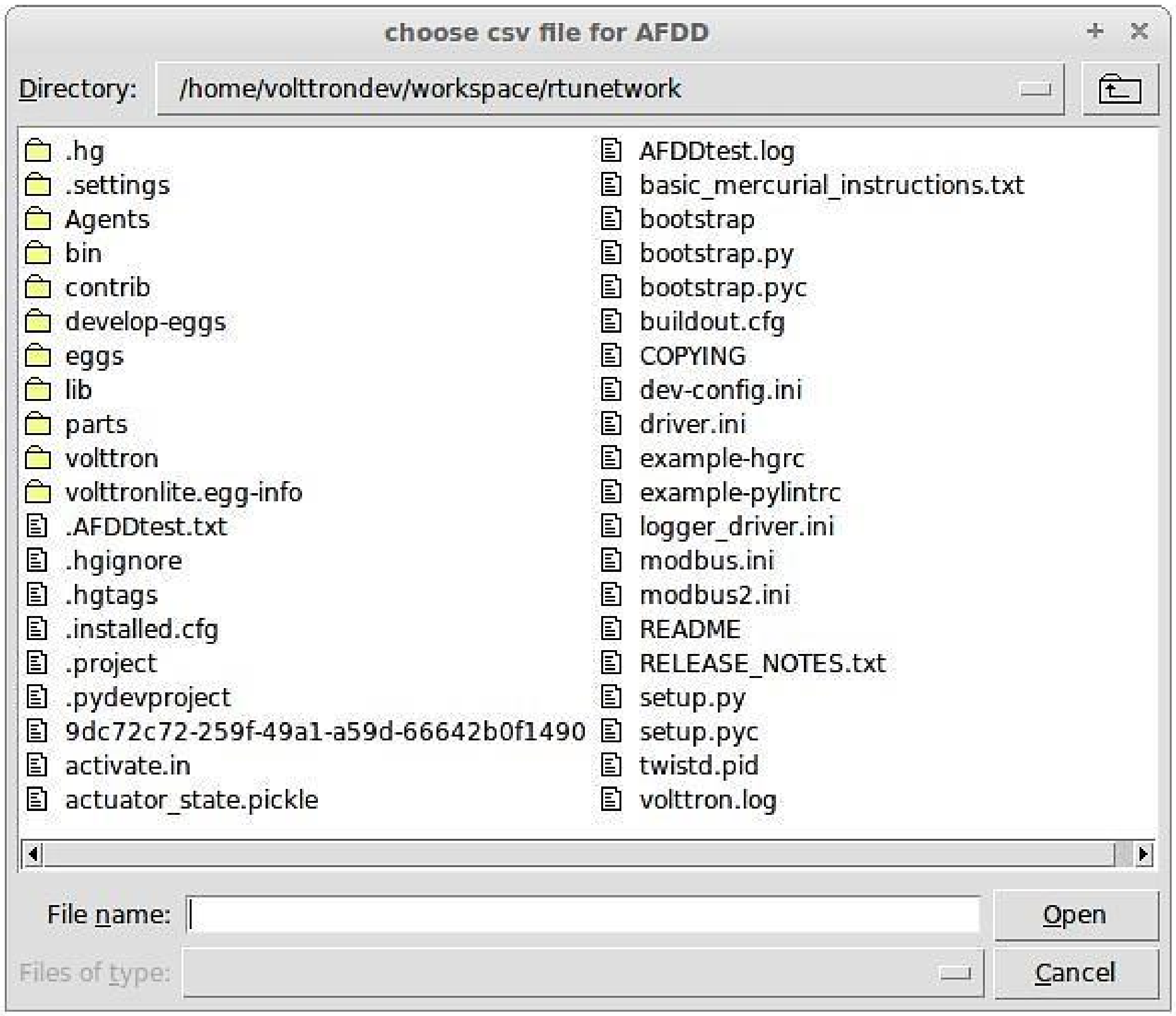
Diagnostic Code	Code Message
31	Unit is not currently cooling or conditions are not favorable for economizin
32	Insufficient outdoor air when economizing (Fault)
33	Outdoor-air damper is not fully open when the unit should be economizing
36	OAD is open but conditions were not favorable for OAF calculation (No Fa
37	Missing data necessary for fault detection (No Fault)
38	OAD is open when economizing but OAF calculation led to an unexpected
39	Unit is off (No Fault)
AFDD-3 (Unit Economizing When it Should)	
40	No faults detected
41	Damper should be at minimum position but is not (Fault)
42	Damper is at minimum for ventilation (No Fault)
43	Conditions favorable for economizing (No Fault)
47	Missing data necessary for fault detection (No Fault)
49	Unit is off (No Fault)
AFDD-4 (Excess Outdoor-air Intake)	
50	No faults detected
51	Excessive outdoor-air intake
52	Damper is at minimum but conditions are not favorable for OAF calculatio
53	Damper is not at minimum (Fault)
56	Unit should be economizing (No Fault)
57	Missing data necessary for fault detection (No Fault)
58	Damper is at minimum but OAF calculation led to an unexpected value (No
59	Unit is off (No Fault)
AFDD-5 (Insufficient Outdoor-air Ventilation)	
60	No faults detected
61	Insufficient outdoor-air intake (Fault)
62	Damper is at minimum but conditions are not favorable for OAF calculatio
63	Damper is not at minimum when is should not be (Fault)
66	Unit should be economizing (No Fault)
67	Missing data necessary for fault detection (No Fault)
68	Damper is at minimum but conditions are not favorable for OAF calculatio
69	Unit is off (No Fault)
AFDD-6 (Schedule)	
70	Unit is operating correctly based on input on/off time (No Fault)
71	Unit is operating at a time designated in schedule as “off” time
77	Missing data

Launching the AFDD for CSV Data Input

When utilizing the AFDD agent and inputting data via a csv text file, set the **csv_input** parameter, contained in the AFDD configuration file, to “1.”

- Launch the agent normally.
- A small file input box will appear. Navigate to the csv data file and select the csv file to input for the diagnostic.
- The result will be created for this RTU or AHU in the results folder described.

Figure 2 shows the dialog box that is used to input the csv data file.



```
.hg                                      AFDDtest.log
.settings                                basic_mercurial_instructions.txt
Agents                                   bootstrap
bin                                      bootstrap.py
contrib                                  bootstrap.pyc
develop-eggs                             buildout.cfg
eggs                                     COPYING
lib                                      dev-config.ini
parts                                    driver.ini
volttron                                 example-hgrc
volttronlite.egg-info                    example-pylintrc
.AFDDtest.txt                            logger_driver.ini
.hgignore                                modbus.ini
.hgtags                                  modbus2.ini
.installed.cfg                           README
.project                                 RELEASE_NOTES.txt
.pydevproject                            setup.py
9dc72c72-259f-49a1-a59d-66642b0f1490     setup.pyc
activate.in                              twistd.pid
actuator_state.pickle                    volttron.log
```

Figure 2 File Selection Dialog Box when Inputting Data in a csv File

If "Cancel" is pushed on the file input dialog box, the AFDD will acknowledge that no file was selected. The Passive AFDD must be restarted to run the diagnostics. If a non-csv file is selected, the AFDD will acknowledge the file selected was not a csv file. The AFDD must be restarted to run the diagnostics.

Figure 3 shows a sample input data in a csv format. The header, or name for each column from the data input csv file used for analysis, should match the name given in the configuration file, as shown in Figure 1, boxed in red.

```
Timestamp,OutsideAirTemp,ReturnAirTemp,MixedAirTemp,CompressorStatus,HeatingStatus,FanStatus,Damper
5/19/2012 6:00,48.902,56.43727273,58.68472222,0,0,0,0
5/19/2012 7:00,51.12316667,59.47933333,59.58916667,0,0,0,0
5/19/2012 8:00,54.70866667,61.1625,64.34266667,0,0,0,0
```

Figure 3 Sample of CSV Data for Passive AFDD Agent

The Demand Response (DR) Agent

Many utilities around the country have or are considering implementing dynamic electrical pricing programs that use time-of-use (TOU) electrical rates. TOU electrical rates vary based on the demand for electricity. Critical peak pricing (CPP), also referred to as critical peak days or event days, is an electrical rate where utilities charge an increased price above normal pricing for peak hours on the CPP day. CPP times coincide with peak demand on the utility; these CPP events are generally called between 5 to 15 times per year and occur when the electrical demand is high and the supply is low. Customers on a flat standard rate who enroll in a peak time rebate program receive rebates for using less electricity when a utility calls for a peak time event. Most CPP events occur during the summer season on very hot days. The initial implementation of the DR agent addresses CPP events where the RTU would normally be cooling. This implementation can be extended to handle CPP events for heating during the winter season as well. This implementation of the DR agent is specific to the CPP, but it can easily be modified to work with other incentive signals (real-time pricing, day ahead, etc.).

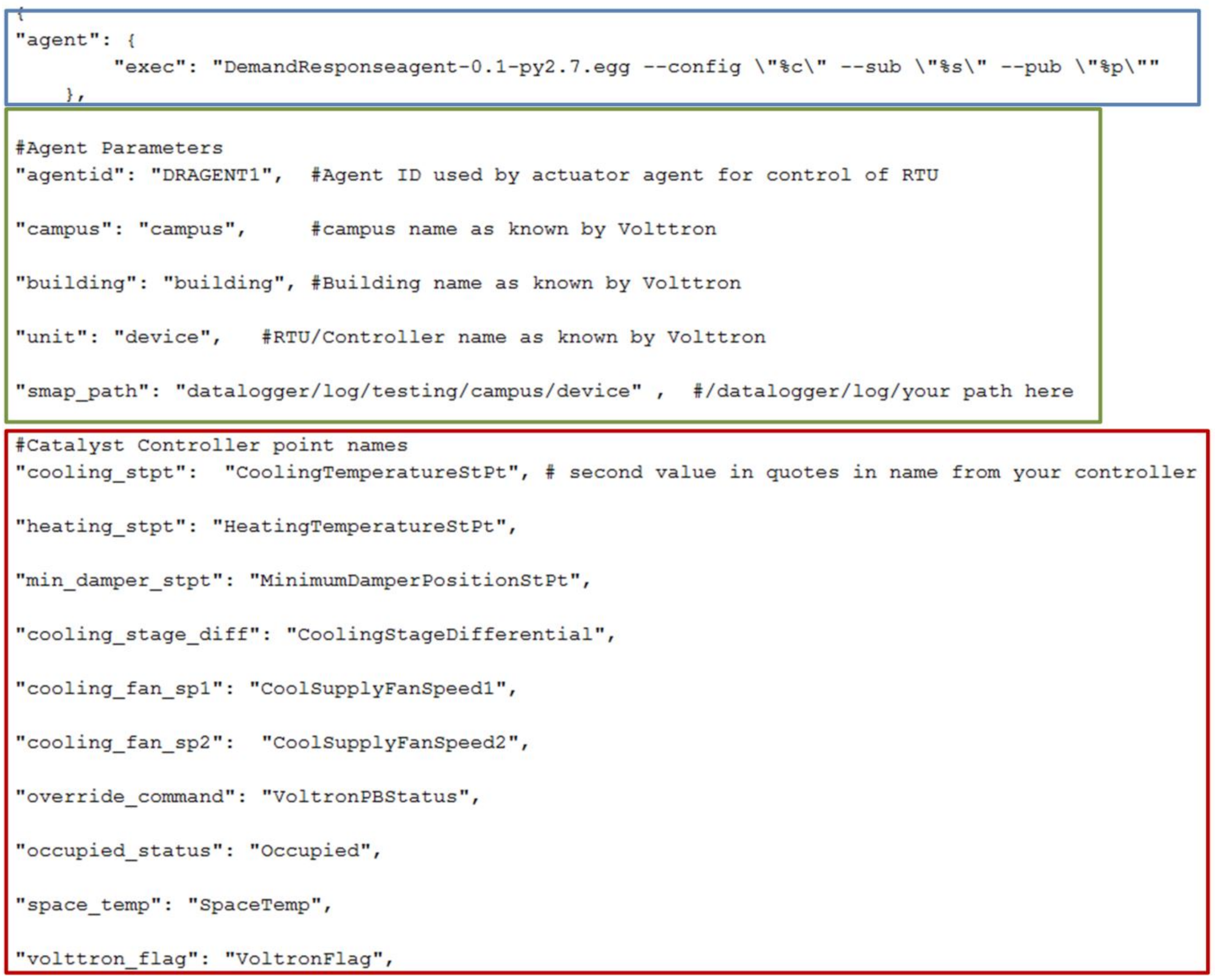
The main goal of the building owner/operator is to minimize the electricity consumption during peak summer periods on a CPP day. To accomplish that goal, the DR agent performs three distinct functions:

- **Step 1 – Pre-Cooling:** Prior to the CPP event period, the cooling and heating (to ensure the RTU is not driven into a heating mode) set points are reset lower to allow for pre-cooling. This step allows the RTU to cool the building below its normal cooling set point while the electrical rates are still low (compared to CPP events). The cooling set point is typically lowered between 3 and 5oF below the normal. Rather than change the set point to a value that is 3 to 5oF below the normal all at once, the set point is gradually lowered over a period of time.
- **Step 2 – Event:** During the CPP event, the cooling set point is raised to a value that is 4 to 5oF above the normal, the damper is commanded to a position that is slightly below the normal minimum (half of the normal minimum), the fan speed is slightly reduced (by 10% to 20% of the normal speed, if the unit has a variable-frequency drive (VFD)), and the second stage cooling differential (time delay between stage one and stage two cooling) is increased (by few degrees, if the unit has multiple stages). The modifications to the normal set points during the CPP event for the fan speed, minimum damper position, cooling set point, and second stage cooling differential are user adjustable. These steps will reduce the electrical consumption during the CPP event. The pre-cooling actions taken in step 1 will allow the temperature to slowly float up to the CPP cooling temperature set point and reduce occupant discomfort during the attempt to shed load.
- **Step 3 – Post-Event:** The DR agent will begin to return the RTU to normal operations by changing the cooling and heating set points to their normal values. Again, rather than changing the set point in one step, the set point is changed gradually over a period of time to avoid the “rebound” effect (a spike in energy consumption after the CPP event when RTU operations are returning to normal).

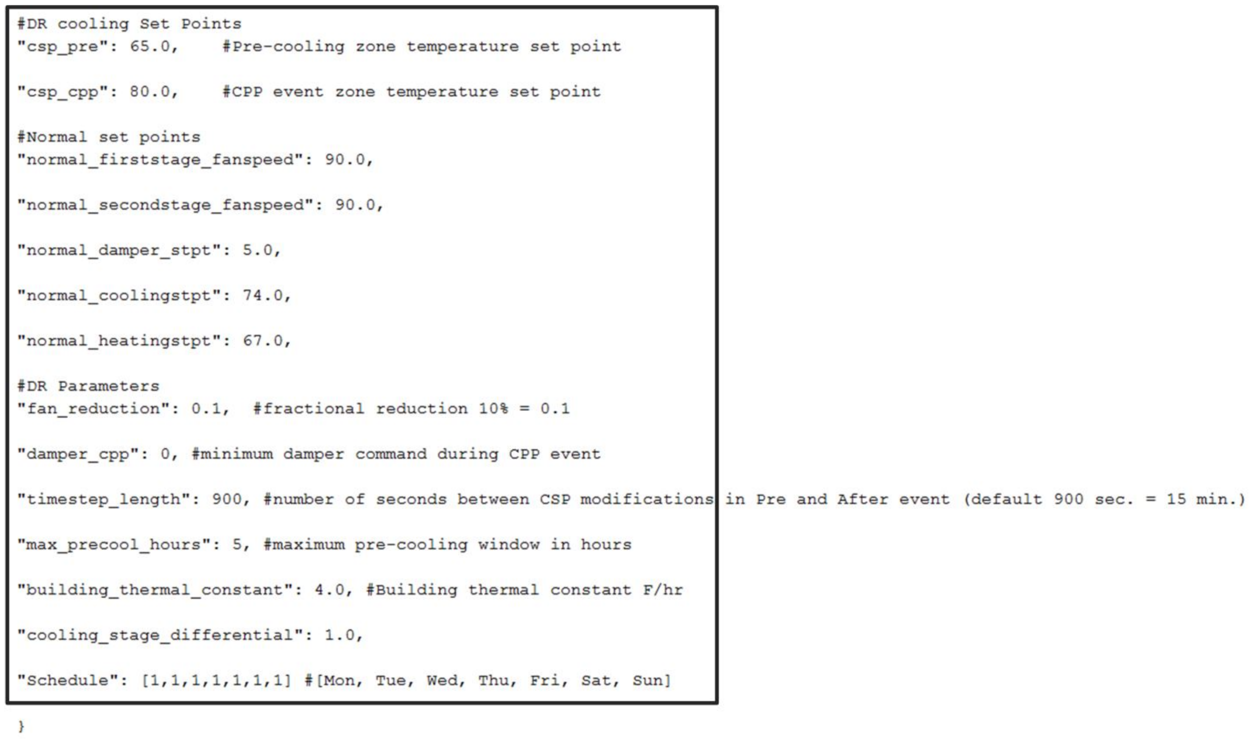
The following section will detail how to configure and launch the DR agent.

Configuring DR Agent

Before launching the DR agent, several parameters require configuration. The DR utilizes the same JSON style configuration file that the Actuator, Listener, and Weather agent use. A notable limitation of the DR agent is that the DR agent requires active control of an RTU/AHU. The DR agent modifies set points on the controller or thermostat to reduce electrical consumption during a CPP event. The DR agent must be able to **set** certain values on the RTU/AHU controller or thermostat via the Actuator agent. Figure 4 shows a sample configuration file for the DR agent:



```
{
"agent": {
        "exec": "DemandResponseagent-0.1-py2.7.egg --config \"%c\" --sub \"%s\" --pub \"%p\""
    },

#Agent Parameters
"agentid": "DRAGENT1",  #Agent ID used by actuator agent for control of RTU

"campus": "campus",     #campus name as known by Volttron

"building": "building", #Building name as known by Volttron

"unit": "device",   #RTU/Controller name as known by Volttron

"smap_path": "datalogger/log/testing/campus/device" ,  #/datalogger/log/your path here

#Catalyst Controller point names
"cooling_stpt":  "CoolingTemperatureStPt", # second value in quotes in name from your controller

"heating_stpt": "HeatingTemperatureStPt",

"min_damper_stpt": "MinimumDamperPositionStPt",

"cooling_stage_diff": "CoolingStageDifferential",

"cooling_fan_sp1": "CoolSupplyFanSpeed1",

"cooling_fan_sp2":  "CoolSupplyFanSpeed2",

"override_command": "VoltronPBStatus",

"occupied_status": "Occupied",

"space_temp": "SpaceTemp",

"volttron_flag": "VoltronFlag",
```



```
#DR cooling Set Points
"csp_pre": 65.0,    #Pre-cooling zone temperature set point

"csp_cpp": 80.0,    #CPP event zone temperature set point

#Normal set points
"normal_firststage_fanspeed": 90.0,

"normal_secondstage_fanspeed": 90.0,

"normal_damper_stpt": 5.0,

"normal_coolingstpt": 74.0,

"normal_heatingstpt": 67.0,

#DR Parameters
"fan_reduction": 0.1,  #fractional reduction 10% = 0.1

"damper_cpp": 0, #minimum damper command during CPP event

"timestep_length": 900, #number of seconds between CSP modifications in Pre and After event (default 900 sec. = 15 min.)

"max_precool_hours": 5, #maximum pre-cooling window in hours

"building_thermal_constant": 4.0, #Building thermal constant F/hr

"cooling_stage_differential": 1.0,

"Schedule": [1,1,1,1,1,1,1] #[Mon, Tue, Wed, Thu, Fri, Sat, Sun]

}
```

Figure 4 Example Configuration File for the DR Agent

The parameters boxed in black (Figure 4) are the demand response parameters; these may require modification to ensure the DR agent and corresponding CPP event are executed as one desires. The parameters in the example configuration that are boxed in red are the controller or thermostat points, as specified in the Modbus or BACnet (depending on what communication protocol your device uses) registry file, that the DR agent will set via the Actuator agent. These device points must be writeable, and configured as such, in the registry (Modbus or BACnet) file. The following list describes each user configurable parameter:

- **agentid** - This is the ID used when making schedule, set, or get requests to the Actuator agent; usually a string data type.
- **campus** - Campus name as configured in the sMAP driver. This parameter builds the device path that allows the Actuator agent to set and get values on the device; usually a string data type.
- **building** - Building name as configured in the sMAP driver. This parameter builds the device path that allows the Actuator agent to set and get values on the device; usually a string data type.
- **unit** - Device name as configured in the sMAP driver. This parameter builds the device path that allows the Actuator agent to set and get values on the device; usually a string data type. Note: The campus, building, and unit parameters are used to build the device path (campus/building/unit). The device path is used for communication on the message bus.
- **csp_pre** - Pre-cooling space cooling temperature set point.
- **csp_cpp** - CPP event space cooling temperature set point.
- **normal_firststage_fanspeed** - Normal operations, first stage fan speed set point.
- **normal_secondstage_fanspeed** - Normal operations, second stage fan speed set point.
- **normal_damper_stpt** - Normal operations, minimum outdoor-air damper set point.
- **normal_coolingstpt** - Normal operations, space cooling temperature set point.
- **normal_heatingstpt** - Normal operations, space heating temperature set point.

- **fan_reduction** - Fractional reduction in fan speeds during CPP event (default: 0.1-10%).
- **damper_cpp** - CPP event, minimum outdoor-air damper set point.
- **max_precool_hours** - Maximum allotted time for pre-cooling, in hours.
- **cooling_stage_differential** - Difference in actual space temperature and set-point temperature before second stage cooling is activated.
- **schedule** - Day of week occupancy schedule "0" indicate unoccupied day and "1" indicate occupied day (e.g., [1,1,1,1,1,1,1] = [Mon, Tue, Wed, Thu, Fri, Sat, Sun]).

OpenADR (Open Automated Demand Response)

Open Automated Demand Response (OpenADR) is an open and standardized way for electricity providers and system operators to communicate DR signals with each other and with their customers using a common language over any existing IP-based communications network, such as the Internet. Lawrence Berkeley National Laboratory created an agent to receive DR signals from an external source (e.g., OpenADR server) and publish this information on the message bus. The DR agent subscribes to the OpenADR topic and utilizes the contents of this message to coordinate the CPP event.

The OpenADR signal is formatted as follows:

```
'openadr/event',{'Content-Type': ['application/json'], 'requesterID': 'openadragent'},
↪ {'status': 'near',
'start_at': '2013-6-15 14:00:00', 'end_at': '2013-10-15 18:00:00', 'mod_num': 0, 'id':
'18455630-a5c4-4e4a-9d53-b3cf989ccf1b','signals': 'null'}
```

The red text in the signal is the topic associated with CPP events that are published on the message bus. The text in dark blue is the message; this contains the relevant information on the CPP event for use by the DR agent.

If one desires to test the behavior of a device when responding to a DR event, such an event can be simulated by manually publishing a DR signal on the message bus. From the base VOLTTRON directory, in a terminal window, enter the following commands:

1. Activate project:

```
$ source env/bin/activate
```

2. Start Python interpreter:

```
$ python
```

3. Import VOLTTRON modules:

```
$ from volttron.platform.vip.agent import Core, Agent
```

4. Import needed Python library:

```
$ import gevent
```

5. Instantiate agent (agent will publish OpenADR message):

```
$ agent = Agent(address='ipc://@/home/volttron-user/.volttron/run/vip.socket')
```

6. Ensure the setup portion of the agent run loop is executed:

```
$ gevent.spawn(agent.core.run).join(0)
```

7. Publish simulated OpenADR message:

```
$ agent.vip.pubsub.publish(peer='pubsub', topic='openadr/event',headers={},
message={'id': 'event_id','status': 'active', 'start_at': 10-30-15 15:00', 'end_at':
↪'10-30-15
18:00'})
```

To cancel this event, enter the following command:

```
$ agent.vip.pubsub.publish(peer='pubsub', topic='openadr/event',headers={}, message={
↪'id':
'event_id','status': 'cancelled', 'start_at': 10-30-15 15:00', 'end_at': '10-30-15␣
↪18:00'})
```

The DR agent will use the most current signal for a given day. This allows utilities/OpenADR to modify the signal up to the time prescribed for pre-cooling.

DR Agent Output to sMAP

After the DR agent has been configured, the agent can be launched. To launch the DR agent from the base VOLTTRON directory, enter the following commands in a terminal window:

1. Run *pack_install* script on DR agent:

```
$ . scripts/core/pack_install.sh applications/DemandResponseAgent
applications/DemandResponseAgent/demandresponse.launch.json dr-agent
```

Upon successful completion of this command, the terminal output will show the install directory, the agent UUID (unique identifier for an agent; the UUID shown in red is only an example and each instance of an agent will have a different UUID) and the agent name (blue text):

```
Installed
/home/volttron-user/.volttron/packaged/DemandResponseagent-0.1-py2-none-
any.whlas 5b1706d6-b71d-4045-86a3-8be5c85ce801
DemandResponseagent-0.1
```

2. Start the agent:

```
$ vctl start --tag dr-agent
```

3. Verify that agent is running:

```
$ vctl status
$ tail -f volttron.log
```

If changes are made to the DR agent's configuration file after the agent is launched, it is necessary to stop and reload the agent. In a terminal, enter the following commands:

```
$ vctl stop --tag dr-agent
$ vctl remove --tag dr-agent
```

Then re-build and start the updated agent.

1.46.2 Simulation Subsystem

The simulation subsystem includes a set of device simulators and a clock that can run faster (or slower) than real time. It can be used to test VOLTTRON agents or drivers. It could be particularly useful when simulating multi-agent and/or multi-driver scenarios.

The source code for the agents and drivers comprising this subsystem resides in the https://github.com/VOLTTRON/volttron-applications github repository.

This subsystem is designed to be extended easily. Its initial delivery includes a set of simulated energy devices that report status primarily in terms of power (kilowatts) produced and consumed. It could easily be adapted, though, to simulate and report data for devices that produce, consume and manage resources other than energy.

Three agents work together to run a simulation:

1. **SimulationClockAgent.** This agent manages the simulation's clock. After it has been supplied with a start time, a stop time, and a clock-speed multiplier, and it has been asked to start a simulation, it provides the current simulated time in response to requests. If no stop time has been provided, the SimulationClockAgent continues to manage the simulation clock until the agent is stopped. If no clock-speed multiplier has been provided, the simulation clock runs at normal wall-clock speed.
2. **SimulationDriverAgent.** Like MasterDriverAgent, this agent is a front-end manager for device drivers. It handles get_point/set_point requests from other agents, and it periodically "scrapes" and publishes each driver's points. If a device driver has been built to run under MasterDriverAgent, with a few minor modifications (detailed below) it can be adapted to run under SimulationDriverAgent.
3. **SimulationAgent.** This agent configures, starts, and reports on a simulation. It furnishes a variety of configuration parameters to the other simulation agents, starts the clock, subscribes to scraped driver points, and generates a CSV output file.

Four device drivers have been provided:

1. **storage (simstorage).** The storage driver simulates an energy storage device (i.e., a battery). When it receives a power dispatch value (positive to charge the battery, negative to discharge it), it adjusts its charging behavior accordingly. Its reported power doesn't necessarily match the dispatch value, since (like an actual battery) it stays within configured max-charge/max-discharge limits, and its power dwindles as its state of charge approaches a full or empty state.
2. **pv (simpv).** The PV driver simulates a photovoltaic array (solar panels), reporting the quantity of solar power produced. Solar power is calculated as a function of (simulated) time, using a data file of incident-sunlight metrics. A year's worth of solar data has been provided as a sample resource.
3. **load (simload).** The load driver simulates the behavior of a power consumer such as a building, reporting the quantity of power consumed. It gets its power metrics as a function of (simulated) time from a data file of power readings. A year's worth of building-load data has been provided as a sample resource.
4. **meter (simmeter).** The meter driver simulates the behavior of a circuit's power meter. This driver, as delivered, is actually just a shell of a simulated device. It's able to report power as a function of (simulated) time, but it has no built-in default logic for deciding what particular power metrics to report.

Linux Installation

The following steps describe how to set up and run a simulation. They assume that `VOLTTRON / volttron` and `VOLTTRON / volttron-applications` repositories have been downloaded from github, and that Linux shell variables `$VOLTTRON_ROOT` and `$VOLTTRON_APPLICATIONS_ROOT` point at the root directories of these repositories.

First, create a soft link to the applications directory from the volttron directory, if that hasn't been done already:

```
$ cd $VOLTTRON_ROOT
$ ln -s $VOLTTRON_APPLICATIONS_ROOT applications
```

With VOLTTRON running, load each simulation driver's configuration into a "simulation.driver" config store:

```
$ export SIMULATION_DRIVER_ROOT=$VOLTTRON_ROOT/applications/kisensum/Simulation/
↪SimulationDriverAgent

$ vctl config store simulation.driver simload.csv $SIMULATION_DRIVER_ROOT/simload.csv␣
↪--csv
$ vctl config store simulation.driver devices/simload $SIMULATION_DRIVER_ROOT/simload.
↪config

$ vctl config store simulation.driver simmeter.csv $SIMULATION_DRIVER_ROOT/simmeter.
↪csv --csv
$ vctl config store simulation.driver devices/simmeter $SIMULATION_DRIVER_ROOT/
↪simmeter.config

$ vctl config store simulation.driver simpv.csv $SIMULATION_DRIVER_ROOT/simpv.csv --
↪csv
$ vctl config store simulation.driver devices/simpv $SIMULATION_DRIVER_ROOT/simpv.
↪config

$ vctl config store simulation.driver simstorage.csv $SIMULATION_DRIVER_ROOT/
↪simstorage.csv --csv
$ vctl config store simulation.driver devices/simstorage $SIMULATION_DRIVER_ROOT/
↪simstorage.config
```

Install and start each simulation agent:

```
$ export SIMULATION_ROOT=$VOLTTRON_ROOT/applications/kisensum/Simulation
$ export VIP_SOCKET="ipc://$VOLTTRON_HOME/run/vip.socket"

$ python scripts/install-agent.py \
    --vip-identity simulation.driver \
    --tag          simulation.driver \
    --agent-source $SIMULATION_ROOT/SimulationDriverAgent \
    --config       $SIMULATION_ROOT/SimulationDriverAgent/simulationdriver.config \
    --force \
    --start

$ python scripts/install-agent.py \
    --vip-identity simulationclock \
    --tag          simulationclock \
    --agent-source $SIMULATION_ROOT/SimulationClockAgent \
    --config       $SIMULATION_ROOT/SimulationClockAgent/simulationclock.config \
    --force \
    --start

$ python scripts/install-agent.py \
    --vip-identity simulationagent \
    --tag          simulationagent \
    --agent-source $SIMULATION_ROOT/SimulationAgent \
    --config       $SIMULATION_ROOT/SimulationAgent/simulationagent.config \
    --force \
    --start
```

SimulationAgent Configuration Parameters

This section describes SimulationAgent's configurable parameters. Each of these has a default value and behavior, allowing the simulation to be run "out of the box" without configuring any parameters.

Type	Param Name	Data Type	Default	Comments
General	agent_id	str	simulation	
General	heartbeat_period	int sec	5	
General	sim_driver_list	list of str	[simload, simmeter, simpv, simstorage]	Allowed keywords are simload, simmeter, simpv, simstorage.
Clock	sim_start	datetime str	2017-02-02 13:00:00	
Clock	sim_end	datetime str	None	If None, sim doesn't stop.
Clock	sim_speed	float sec	180.0	This is a multiplier, e.g. 1 sec actual time = 180 sec sim time.
Load	load_timestamp_column_header	str	local_date	
Load	load_power_column_header	str	load_kw	
Load	load_data_frequency_min	int min	15	
Load	load_data_year	str	2015	
Load	load_csv_file_path	str	~/repos/volttron-applications/kisensum/ Simulation/SimulationAgent/data/load_and_pv.csv	~ and shell variables in the pathname will be expanded. The file must exist.
PV	pv_panel_area	float m2	50.0	
PV	pv_efficiency	float 0.0-1.0	0.75	
PV	pv_data_frequency_min	int min	30	
PV	pv_data_year	str	2015	
PV	pv_csv_file_path	str	~/repos/volttron-applications/kisensum/ Simulation/SimulationAgent/data/nrel_pv_readings.csv	~ and shell variables in the pathname will be expanded. The file must exist.
Storage	storage_soc_kwh	float kWh	30.0	
Storage	storage_max_soc_kwh	float kWh	50.0	
Storage	storage_max_charge_kw	float kW	15.0	
Storage	storage_max_discharge_kw	float kW	12.0	
Storage	storage_reduced_charge_soc_threshold	float 0.0-1.0	0.80	Charging will be reduced when SOC % > this value.
Storage	storage_reduced_discharge_soc_threshold	float 0.0-1.0	0.20	Discharging will be reduced when SOC % < this value.
Dispatch	storage_setpoint_rule	str keyword	oscillation	See below.
Dispatch	positive_dispatch_kw	float kW >= 0.0	15.0	
Dis-	nega-	float	-15.0	

The **oscillation** setpoint rule slowly oscillates between charging and discharging based on the storage device's state of charge (SOC):

```
If SOC < (``go_positive_if_below`` * ``storage_max_soc_kwh``):
    dispatch power = ``positive_dispatch_kw``

If SOC > (``go_negative_if_above`` * ``storage_max_soc_kwh``)
    dispatch power = ``negative_dispatch_kw``

Otherwise:
    dispatch power is unchanged from its previous value.
```

The **alternate** setpoint rule is used when `storage_setpoint_rule` has been configured with any value other than **oscillation**. It simply charges at the dispatched charging value (subject to the constraints of the other parameters, e.g. `storage_max_discharge_kw`):

```
dispatch power = ``positive_dispatch_kw``
```

Driver Parameters and Points

Load Driver

The load driver's parameters specify how to look up power metrics in its data file.

Type	Name	Data Type	Default	Comments
Param/Point	csv_file_path	string		This parameter must be supplied by the agent.
Param/Point	timestamp_column_header	string	local_date	
Param/Point	power_column_header	string	load_kw	
Param/Point	data_frequency_min	int	15	
Param/Point	data_year	string	2015	
Point	power_kw	float	0.0	
Point	last_timestamp	datetime		

Meter Driver

Type	Name	Data Type	Default	Comments
Point	power_kw	float	0.0	
Point	last_timestamp	datetime		

PV Driver

The PV driver's parameters specify how to look up sunlight metrics in its data file, and how to calculate the power generated from that sunlight.

Type	Name	Data Type	Default	Comments
Param/Point	csv_file_path	string		This parameter must be supplied by the agent.
Param/Point	max_power_kw	float	10.0	
Param/Point	panel_area	float	50.0	
Param/Point	efficiency	float	0.75	
Param/Point	data_frequency_min	int	30	
Param/Point	data_year	string	2015	
Point	power_kw	float	0.0	
Point	last_timestamp	datetime		

Storage Driver

The storage driver's parameters describe the device's power and SOC limits, its initial SOC, and the SOC thresholds at which charging and discharging start to be reduced as its SOC approaches a full or empty state. This reduced power is calculated as a straight-line reduction: charging power is reduced in a straight line from `reduced_charge_soc_threshold` to 100% SOC, and discharging power is reduced in a straight line from `reduced_discharge_soc_threshold` to 0% SOC.

Type	Name	Data Type	Default	Comments
Param/Point	max_charge_kw	float	15.0	
Param/Point	max_discharge_kw	float	15.0	
Param/Point	max_soc_kwh	float	50.0	
Param/Point	soc_kwh	float	25.0	
Param/Point	reduced_charge_soc_threshold	float	0.8	
Param/Point	reduced_discharge_soc_threshold	float	0.2	
Point	dispatch_kw	float	0.0	
Point	power_kw	float	0.0	
Point	last_timestamp	datetime		

Working with the Sample Data Files

The Load and PV simulation drivers report power readings that are based on metrics from sample data files. The software distribution includes sample Load and PV files containing at least a year's worth of building-load and sunlight data.

CSV files containing different data sets of load and PV data can be substituted by specifying their paths in SimulationAgent's configuration, altering its other parameters if the file structures and/or contents are different.

Load Data File

`load_and_pv.csv` contains building-load and PV power readings at 15-minute increments from 01/01/2014 - 12/31/2015. The data comes from a location in central Texas. The file's data columns are: `utc_date`, `local_date`, `time_offset`, `load_kw`, `pv_kw`. The load driver looks up the row with a matching local_date and returns its load_kw value.

Adjust the following SimulationAgent configuration parameters to change how load power is derived from the data file:

- Use `load_csv_file_path` to set the path of the sample data file
- Use `load_data_frequency_min` to set the frequency of the sample data

- Use `load_data_year` to set the year of the sample data
- Use `load_timestamp_column_header` to indicate the header name of the timestamp column
- Use `load_power_column_header` to indicate the header name of the power column

PV Data File

`nrel_pv_readings.csv` contains irradiance data at 30-minute increments from 01/01/2015 - 12/31/2015, downloaded from NREL's National Solar Radiation Database, https://nsrdb.nrel.gov. The file's data columns are: `Year, Month, Day, Hour, Minute, DHI, DNI, Temperature`. The PV driver looks up the row with a matching date/time and uses its DHI (diffuse horizontal irradiance) to calculate the resulting solar power produced:

```
power_kw = irradiance * panel_area * efficiency / elapsed_time_hrs
```

Adjust the following SimulationAgent configuration parameters to change how solar power is derived from the data file:

- Use `pv_csv_file_path` to set the path of the sample data file
- Use `pv_data_frequency_min` to set the frequency of the sample data
- Use `pv_data_year` to set the year of the sample data
- Use `pv_panel_area` and `pv_efficiency` to indicate how to transform an irradiance measurement in wh/m2 into a power reading in kw.

If a PV data file will be used that has a column structure which differs from the one in the supplied sample, an adjustment may need to be made to the simpv driver software.

Running the Simulation

One way to monitor the simulation's progress is to look at debug trace in VOLTTRON's log output, for example:

```
2017-05-01 15:05:42,815 (simulationagent-1.0 9635) simulation.agent DEBUG: 2017-05-01␣
↪15:05:42.815484 Initializing drivers
2017-05-01 15:05:42,815 (simulationagent-1.0 9635) simulation.agent DEBUG: ␣
↪Initializing Load: timestamp_column_header=local_date, power_column_header=load_kw,␣
↪data_frequency_min=15, data_year=2015, csv_file_path=/Users/robcalvert/repos/
↪volttron-applications/kisensum/Simulation/SimulationAgent/data/load_and_pv.csv
2017-05-01 15:05:42,823 (simulationagent-1.0 9635) simulation.agent DEBUG: ␣
↪Initializing PV: panel_area=50, efficiency=0.75, data_frequency_min=30, data_
↪year=2015, csv_file_path=/Users/robcalvert/repos/volttron-applications/kisensum/
↪Simulation/SimulationAgent/data/nrel_pv_readings.csv
2017-05-01 15:05:42,832 (simulationagent-1.0 9635) simulation.agent DEBUG: ␣
↪Initializing Storage: soc_kwh=30.0, max_soc_kwh=50.0, max_charge_kw=15.0, max_
↪discharge_kw=12.0, reduced_charge_soc_threshold = 0.8, reduced_discharge_soc_
↪threshold = 0.2
2017-05-01 15:05:42,844 (simulationagent-1.0 9635) simulation.agent DEBUG: 2017-05-01␣
↪15:05:42.842162 Started clock at sim time 2017-02-02 13:00:00, end at 2017-02-02␣
↪16:00:00, speed multiplier = 180.0
2017-05-01 15:05:57,861 (simulationagent-1.0 9635) simulation.agent DEBUG: 2017-05-01␣
↪15:05:57.842164 Reporting at sim time 2017-02-02 13:42:00
2017-05-01 15:05:57,862 (simulationagent-1.0 9635) simulation.agent DEBUG:  devices/
↪simload/power_kw = 486.1
2017-05-01 15:05:57,862 (simulationagent-1.0 9635) simulation.agent DEBUG:  devices/
↪simpv/power_kw = -0.975
```

(continues on next page)

(continued from previous page)

```
2017-05-01 15:05:57,862 (simulationagent-1.0 9635) simulation.agent DEBUG:  devices/
↪simstorage/dispatch_kw = 0.0
2017-05-01 15:05:57,862 (simulationagent-1.0 9635) simulation.agent DEBUG:  devices/
↪simstorage/last_timestamp = 2017-02-02 13:33:00
2017-05-01 15:05:57,862 (simulationagent-1.0 9635) simulation.agent DEBUG:  devices/
↪simstorage/power_kw = 0.0
2017-05-01 15:05:57,862 (simulationagent-1.0 9635) simulation.agent DEBUG:  devices/
↪simstorage/soc_kwh = 30.0
2017-05-01 15:05:57,862 (simulationagent-1.0 9635) simulation.agent DEBUG:  net_power_
↪kw = 485.125
2017-05-01 15:05:57,862 (simulationagent-1.0 9635) simulation.agent DEBUG:  report_
↪time = 2017-02-02 13:42:00
2017-05-01 15:05:57,862 (simulationagent-1.0 9635) simulation.agent DEBUG:          ␣
↪Setting storage dispatch to 15.0 kW
2017-05-01 15:06:12,901 (simulationagent-1.0 9635) simulation.agent DEBUG: 2017-05-01␣
↪15:06:12.869471 Reporting at sim time 2017-02-02 14:30:00
2017-05-01 15:06:12,901 (simulationagent-1.0 9635) simulation.agent DEBUG:  devices/
↪simload/power_kw = 467.5
2017-05-01 15:06:12,901 (simulationagent-1.0 9635) simulation.agent DEBUG:  devices/
↪simpv/power_kw = -5.925
2017-05-01 15:06:12,901 (simulationagent-1.0 9635) simulation.agent DEBUG:  devices/
↪simstorage/dispatch_kw = 15.0
2017-05-01 15:06:12,901 (simulationagent-1.0 9635) simulation.agent DEBUG:  devices/
↪simstorage/last_timestamp = 2017-02-02 14:27:00
2017-05-01 15:06:12,901 (simulationagent-1.0 9635) simulation.agent DEBUG:  devices/
↪simstorage/power_kw = 15.0
2017-05-01 15:06:12,901 (simulationagent-1.0 9635) simulation.agent DEBUG:  devices/
↪simstorage/soc_kwh = 43.5
2017-05-01 15:06:12,901 (simulationagent-1.0 9635) simulation.agent DEBUG:  net_power_
↪kw = 476.575
2017-05-01 15:06:12,901 (simulationagent-1.0 9635) simulation.agent DEBUG:  report_
↪time = 2017-02-02 14:30:00
2017-05-01 15:06:12,901 (simulationagent-1.0 9635) simulation.agent DEBUG:          ␣
↪Setting storage dispatch to 15.0 kW
2017-05-01 15:06:27,931 (simulationagent-1.0 9635) simulation.agent DEBUG: 2017-05-01␣
↪15:06:27.907951 Reporting at sim time 2017-02-02 15:15:00
2017-05-01 15:06:27,931 (simulationagent-1.0 9635) simulation.agent DEBUG:  devices/
↪simload/power_kw = 474.2
2017-05-01 15:06:27,931 (simulationagent-1.0 9635) simulation.agent DEBUG:  devices/
↪simpv/power_kw = -11.7
2017-05-01 15:06:27,932 (simulationagent-1.0 9635) simulation.agent DEBUG:  devices/
↪simstorage/dispatch_kw = 15.0
2017-05-01 15:06:27,932 (simulationagent-1.0 9635) simulation.agent DEBUG:  devices/
↪simstorage/last_timestamp = 2017-02-02 15:03:00
2017-05-01 15:06:27,932 (simulationagent-1.0 9635) simulation.agent DEBUG:  devices/
↪simstorage/power_kw = 5.362
2017-05-01 15:06:27,932 (simulationagent-1.0 9635) simulation.agent DEBUG:  devices/
↪simstorage/soc_kwh = 48.033
2017-05-01 15:06:27,932 (simulationagent-1.0 9635) simulation.agent DEBUG:  net_power_
↪kw = 467.862
2017-05-01 15:06:27,932 (simulationagent-1.0 9635) simulation.agent DEBUG:  report_
↪time = 2017-02-02 15:15:00
2017-05-01 15:06:27,932 (simulationagent-1.0 9635) simulation.agent DEBUG:          ␣
↪Setting storage dispatch to -15.0 kW
2017-05-01 15:06:42,971 (simulationagent-1.0 9635) simulation.agent DEBUG: 2017-05-01␣
↪15:06:42.939181 Reporting at sim time 2017-02-02 16:00:00
2017-05-01 15:06:42,971 (simulationagent-1.0 9635) simulation.agent DEBUG:  devices/
↪simload/power_kw = 469.5
```

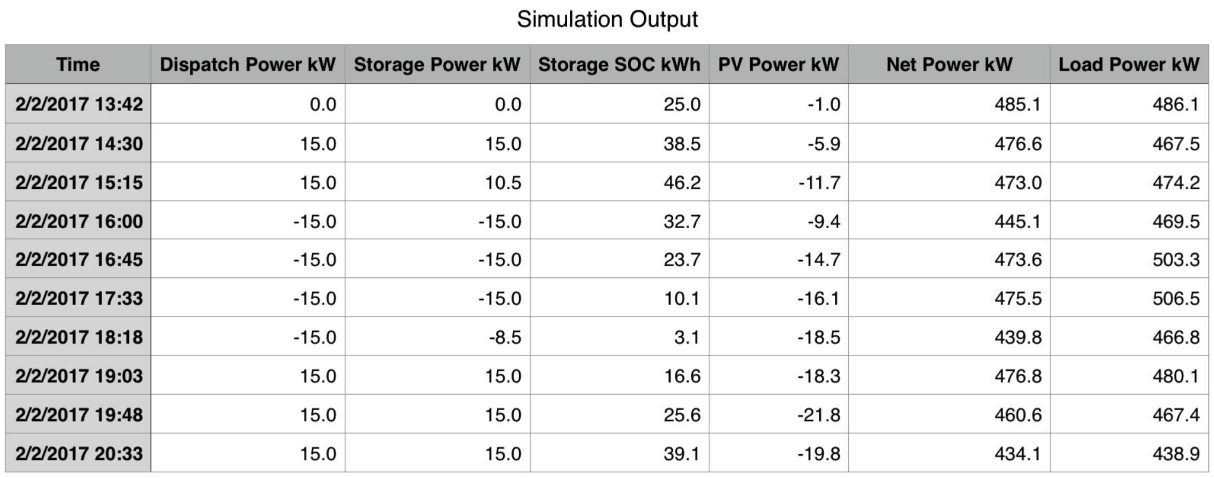
(continues on next page)

(continued from previous page)

```
2017-05-01 15:06:42,971 (simulationagent-1.0 9635) simulation.agent DEBUG:  devices/
↪simpv/power_kw = -9.375
2017-05-01 15:06:42,971 (simulationagent-1.0 9635) simulation.agent DEBUG:  devices/
↪simstorage/dispatch_kw = -15.0
2017-05-01 15:06:42,971 (simulationagent-1.0 9635) simulation.agent DEBUG:  devices/
↪simstorage/last_timestamp = 2017-02-02 15:57:00
2017-05-01 15:06:42,971 (simulationagent-1.0 9635) simulation.agent DEBUG:  devices/
↪simstorage/power_kw = -12.0
2017-05-01 15:06:42,971 (simulationagent-1.0 9635) simulation.agent DEBUG:  devices/
↪simstorage/soc_kwh = 37.233
2017-05-01 15:06:42,971 (simulationagent-1.0 9635) simulation.agent DEBUG:  net_power_
↪kw = 448.125
2017-05-01 15:06:42,971 (simulationagent-1.0 9635) simulation.agent DEBUG:  report_
↪time = 2017-02-02 16:00:00
2017-05-01 15:06:42,971 (simulationagent-1.0 9635) simulation.agent DEBUG:          ␣
↪Setting storage dispatch to -15.0 kW
2017-05-01 15:06:58,001 (simulationagent-1.0 9635) simulation.agent DEBUG: The␣
↪simulation has ended.
```

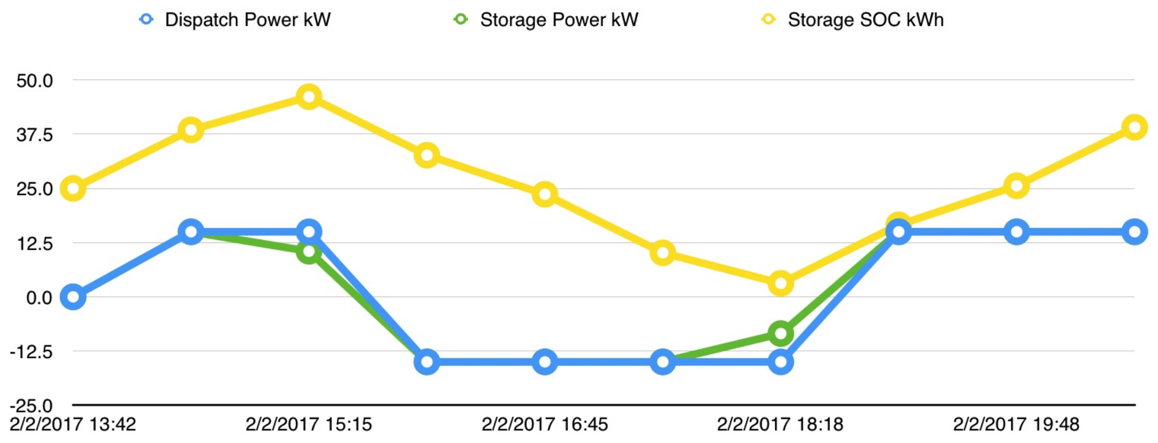
Report Output

The SimulationAgent also writes a CSV output file so that simulation results can be reported by spreadsheets, for example this graph of the simulated storage device following an oscillating dispatch:



Simulation Output

Time	Dispatch Power kW	Storage Power kW	Storage SOC kWh	PV Power kW	Net Power kW	Load Power kW
2/2/2017 13:42	0.0	0.0	25.0	-1.0	485.1	486.1
2/2/2017 14:30	15.0	15.0	38.5	-5.9	476.6	467.5
2/2/2017 15:15	15.0	10.5	46.2	-11.7	473.0	474.2
2/2/2017 16:00	-15.0	-15.0	32.7	-9.4	445.1	469.5
2/2/2017 16:45	-15.0	-15.0	23.7	-14.7	473.6	503.3
2/2/2017 17:33	-15.0	-15.0	10.1	-16.1	475.5	506.5
2/2/2017 18:18	-15.0	-8.5	3.1	-18.5	439.8	466.8
2/2/2017 19:03	15.0	15.0	16.6	-18.3	476.8	480.1
2/2/2017 19:48	15.0	15.0	25.6	-21.8	460.6	467.4
2/2/2017 20:33	15.0	15.0	39.1	-19.8	434.1	438.9



Using the Simulation Framework to Test a Driver

If you're developing a VOLTTRON driver, and you intend to add it to the drivers managed by MasterDriverAgent, then with a few tweaks, you can adapt it so that it's testable from this simulation framework.

As with drivers under MasterDriverAgent, your driver should be go in a .py module that implements a Register class and an Interface class. In order to work within the simulation framework, simulation drivers need to be adjusted as follows:

- Place the module in the interfaces directory under SimulationDriverAgent.
- The module's Register class should inherit from SimulationRegister.
- The module's Interface class should inherit from SimulationInterface.
- If the driver has logic that depends on time, get the simulated time by calling self.sim_time().

Add files with your driver's config and point definitions, and load them into the config store:

```
$ vctl config store simulation.driver \
    yourdriver.csv \
    $VOLTTRON_ROOT/applications/kisensum/Simulation/SimulationDriverAgent/yourdriver.
↪csv --csv
$ vctl config store simulation.driver \
```

(continues on next page)

(continued from previous page)

```
    devices/yourdriver \
    $VOLTTRON_ROOT/applications/kisensum/Simulation/SimulationDriverAgent/yourdriver.
↪config
```

To manage your driver from the SimulationAgent, first add the driver to the sim_driver_list in that agent's config:

```
"sim_driver_list": ["simload", "simpv", "simstorage", "youdriver"]
```

Then, if you choose, you can also revise SimulationAgent's config and logic to scrape and report your driver's points, and/or send RPC requests to your driver.

For Further Information

If you have comments or questions about this simulation support, please contact Rob Calvert at Kisensum, Inc.:

- (github) @rob-calvert
- (email) rob@kisensum.com

1.46.3 Open ADR

OpenADR (Automated Demand Response) is a standard for alerting and responding to the need to adjust electric power consumption in response to fluctuations in grid demand. OpenADR communications are conducted between Virtual Top Nodes (VTNs) and Virtual End Nodes (VENs).

In this implementation, a VOLTTRON agent, OpenADRVenAgent, is made available as a VOLTTRON service. It acts as a VEN, communicating with its VTN via EiEvent and EiReport services in conformance with a subset of the OpenADR 2.0b specification.

A VTN server has also been implemented, with source code in the kisensum/openadr folder of the volttron-applications git repository. As has been described below, it communicates with the VEN and provides a web user interface for defining and reporting on Open ADR events.

The OpenADR 2.0b specification (http://www.openadr.org/specification) is available from the OpenADR Alliance. This implementation also generally follows the DR program characteristics of the Capacity Program described in Section 9.2 of the OpenADR Program Guide (http://www.openadr.org/assets/openadr_drprogramguide_v1.0.pdf).

The OpenADR Capacity Bidding program relies on a pre-committed agreement about the VEN's load shed capacity. This agreement is reached in a bidding process transacted outside of the OpenADR interaction, typically with a long-term scope, perhaps a month or longer. The VTN can "call an event," indicating that a load-shed event should occur in conformance with this agreement. The VTN indicates the level of load shedding desired, when the event should occur, and for how long. The VEN responds with an "optIn" acknowledgment. (It can also "optOut," but since it has been pre-committed, an "optOut" may incur penalties.)

Reference Application

This reference application for VOLTTRON's OpenADR Virtual End Node (VEN) and its Simulation Subsystem demonstrates interactions between the VOLTTRON VEN agent and simulated devices. It employs a Virtual Top Node (VTN) server, demonstrating the full range of interaction and communication in a VOLTTRON implementation of the OpenADR (Automated Demand Response) standard.

The simulation subsystem, described in more detail in *Simulated Subsystem*, includes a set of device simulators and a clock that can run faster (or slower) than real time (using ReferenceApp's default configuration, the clock runs at normal speed).

Eight VOLTTRON agents work together to run this simulation:

1. **ReferenceAppAgent.** This agent configures, starts, and reports on a simulation. It furnishes a variety of configuration parameters to the other simulation agents, starts the clock, subscribes to scraped driver points, and generates a CSV output file. The ReferenceApp also serves as the mediator between the simulated device drivers and the VEN, adjusting driver behavior (particularly the behavior of the "simstorage" battery) while an OpenADR event is in progress, and aggregating and relaying relevant driver metrics to the VEN for reporting to the VTN.
2. **SimulationClockAgent.** This agent manages the simulation's clock. After it has been supplied with a start time, a stop time, and a clock-speed multiplier, and it has been asked to start a simulation, it provides the current simulated time in response to requests. If no stop time has been provided (this is the default behavior while the ReferenceApp is managing the clock), the SimulationClockAgent runs the simulation until the agent is stopped. If no clock-speed multiplier has been provided, the simulation clock runs at normal wallclock speed.
3. **SimulationDriverAgent.** Like MasterDriverAgent, this agent is a front-end manager for device drivers. It handles get_point/set_point requests from other agents, and it periodically "scrapes" and publishes each driver's points. If a device driver has been built to run under MasterDriverAgent, with a few minor modifications (detailed below) it can be adapted to run under SimulationDriverAgent.
4. **ActuatorAgent.** This agent manages write access to device drivers. Another agent may request a scheduled time period, called a Task, during which it controls a device.
5. **OpenADRVenAgent.** This agent implements an OpenADR Virtual End Node (VEN). It receives demand-response event notifications from a Virtual Top Node (VTN), making the event information available to the ReferenceAppAgent and other interested VOLTTRON agents. It also reports metrics to the VTN based on information furnished by the ReferenceAppAgent.
6. **SQLHistorian.** This agent, a "platform historian," captures metrics reported by the simulated devices, storing them in a SQLite database.
7. **VolttronCentralPlatform.** This agent makes the platform historian's device metrics available for reporting by the VolttronCentralAgent.
8. **VolttronCentralAgent.** This agent manages a web user interface that can produce graphical displays of the simulated device metrics captured by the SQLHistorian.

Three simulated device drivers are used:

1. **storage (simstorage).** The storage driver simulates an energy storage device (i.e., a battery). When it receives a power dispatch value (positive to charge the battery, negative to discharge it), it adjusts the storage unit's charging behavior accordingly. Its reported power doesn't necessarily match the dispatch value, since (like an actual battery) it stays within configured max-charge/max-discharge limits, and power dwindles as its state of charge approaches a full or empty state.
2. **pv (simpv).** The PV driver simulates a photovoltaic array (solar panels), reporting the quantity of solar power produced. Solar power is calculated as a function of (simulated) time, using a data file of incident-sunlight metrics. A year's worth of solar data has been provided as a sample resource.
3. **load (simload).** The load driver simulates the behavior of a power consumer such as a building, reporting the quantity of power consumed. It gets its power metrics as a function of (simulated) time from a data file of power readings. A year's worth of building-load data has been provided as a sample resource.

Linux Installation

The following steps describe how to set up and run a simulation. They assume that the `VOLTTRON / volttron` and `VOLTTRON / volttron-applications` repositories have been downloaded from github.

Installing and running a simulation is walked through in the Jupyter notebook in `$VOLTTRON_ROOT/examples/JupyterNotebooks/ReferenceAppAgent.ipynb`. In order to run this notebook, install Jupyter and start the Jupyter server:

```
$ cd $VOLTTRON_ROOT
$ source env/bin/activate
$ pip install jupyter
$ jupyter notebook
```

By default, a browser will open with the Jupyter Notebook dashboard at **http://localhost:8888**. Run the notebook by navigating in the Jupyter Notebook dashboard to **http://localhost:8888/tree/examples/JupyterNotebooks/ReferenceAppAgent.ipynb**.

ReferenceAppAgent Configuration Parameters

This section describes ReferenceAppAgents's configurable parameters. Each of these has a default value and behavior, allowing the simulation to be run "out of the box" without configuring any parameters.

Type	Param Name	Data Type	Default
General	agent_id	str	reference_app
General	heartbeat_period	int sec	5
General	sim_driver_list	list of str	[simload, simpv, simstorage]
General	opt_type	str	optIn
General	report_interval_secs	int sec	30
General	baseline_power_kw	int kw	500
Clock	sim_start	datetime str	2017-04-30 13:00:00
Clock	sim_end	datetime str	None
Clock	sim_speed	float sec	1.0
Load	load_timestamp_column_header	str	local_date
Load	load_power_column_header	str	load_kw
Load	load_data_frequency_min	int min	15
Load	load_data_year	str	2015
Load	load_csv_file_path	str	~/repos/volttron-applications/kisensum/ ReferenceAppA
PV	pv_panel_area	float m2	1000.0
PV	pv_efficiency	float 0.0-1.0	0.75
PV	pv_data_frequency_min	int min	30
PV	pv_data_year	str	2015
PV	pv_csv_file_path	str	~/repos/volttron-applications/kisensum/ ReferenceAppA
Storage	storage_soc_kwh	float kWh	450.0
Storage	storage_max_soc_kwh	float kWh	500.0
Storage	storage_max_charge_kw	float kW	150.0
Storage	storage_max_discharge_kw	float kW	150.0
Storage	storage_reduced_charge_soc _threshold	float 0.0-1.0	0.80
Storage	storage_reduced_discharge_s oc_threshold	float 0.0-1.0	0.20
Dispatch	positive_dispatch_kw	float kW >= 0.0	150.0
Dispatch	negative_dispatch_kw	float kW <= 0.0	-150.0
Dispatch	go_positive_if_below	float 0.0-1.0	0.1
Dispatch	go_negative_if_above	float 0.0-1.0	0.9
Report	report_interval	int seconds	15
Report	report_file_path	str	$VOLTTRON_HOME/run/simulation_out.csv
Actuator	actuator_id	str	simulation.actuator

Type	Param Name	Data Type	Default
VEN	venagent_id	str	venagent

Driver Parameters and Points

Load Driver

The load driver's parameters specify how to look up power metrics in its data file.

Type	Name	Data Type	Default	Comments
Param/Point	csv_file_path	string		This parameter must be supplied by the agent.
Param/Point	timestamp_column_header	string	local_date	
Param/Point	power_column_header	string	load_kw	
Param/Point	data_frequency_min	int	15	
Param/Point	data_year	string	2015	
Point	power_kw	float	0.0	
Point	last_timestamp	datetime		

PV Driver

The PV driver's parameters specify how to look up sunlight metrics in its data file, and how to calculate the power generated from that sunlight.

Type	Name	Data Type	Default	Comments
Param/Point	csv_file_path	string		This parameter must be supplied by the agent.
Param/Point	max_power_kw	float	10.0	
Param/Point	panel_area	float	50.0	
Param/Point	efficiency	float	0.75	
Param/Point	data_frequency_min	int	30	
Param/Point	data_year	string	2015	
Point	power_kw	float	0.0	
Point	last_timestamp	datetime		

Storage Driver

The storage driver's parameters describe the device's power and SOC limits, its initial SOC, and the SOC thresholds at which charging and discharging start to be reduced as its SOC approaches a full or empty state. This reduced power is calculated as a straight-line reduction: charging power is reduced in a straight line from `reduced_charge_soc_threshold` to 100% SOC, and discharging power is reduced in a straight line from `reduced_discharge_soc_threshold` to 0% SOC.

Type	Name	Data Type	Default	Comments
Param/Point	max_charge_kw	float	15.0	
Param/Point	max_discharge_kw	float	15.0	
Param/Point	max_soc_kwh	float	50.0	
Param/Point	soc_kwh	float	25.0	
Param/Point	reduced_charge_soc_threshold	float	0.8	
Param/Point	reduced_discharge_soc_threshold	float	0.2	
Point	dispatch_kw	float	0.0	
Point	power_kw	float	0.0	
Point	last_timestamp	datetime		

VEN Configuration

The VEN may be configured according to its documentation *here*.

Running the Simulation

There are three main ways to monitor the ReferenceApp simulation's progress.

One way is to look at debug trace in VOLTTRON's log output, for example:

```
2018-01-08 17:41:30,333 (referenceappagent-1.0 23842) referenceapp.agent DEBUG: 2018-
↪01-08 17:41:30.333260 Initializing drivers
2018-01-08 17:41:30,333 (referenceappagent-1.0 23842) referenceapp.agent DEBUG:     ␣
↪  Initializing Load: timestamp_column_header=local_date, power_column_header=load_
↪kw, data_frequency_min=15, data_year=2015, csv_file_path=/home/ubuntu/repos/
↪volttron-applications/kisensum/ReferenceAppAgent/data/load_and_pv.csv
2018-01-08 17:41:30,379 (referenceappagent-1.0 23842) referenceapp.agent DEBUG:     ␣
↪  Initializing PV: panel_area=50.0, efficiency=0.75, data_frequency_min=30, data_
↪year=2015, csv_file_path=/home/ubuntu/repos/volttron-applications/kisensum/
↪ReferenceAppAgent/data/nrel_pv_readings.csv
2018-01-08 17:41:30,423 (referenceappagent-1.0 23842) referenceapp.agent DEBUG:     ␣
↪  Initializing Storage: soc_kwh=25.0, max_soc_kwh=50.0, max_charge_kw=15.0, max_
↪discharge_kw=15.0, reduced_charge_soc_threshold = 0.8, reduced_discharge_soc_
↪threshold = 0.2
2018-01-08 17:41:32,331 (referenceappagent-1.0 23842) referenceapp.agent DEBUG: 2018-
↪01-08 17:41:32.328390 Reporting at sim time 2018-01-08 17:41:31.328388
2018-01-08 17:41:32,331 (referenceappagent-1.0 23842) referenceapp.agent DEBUG:     ␣
↪  net_power_kw = 0
2018-01-08 17:41:32,331 (referenceappagent-1.0 23842) referenceapp.agent DEBUG:     ␣
↪  report_time = 2018-01-08 17:41:31.328388
2018-01-08 17:41:32,338 (referenceappagent-1.0 23842) referenceapp.agent DEBUG:     ␣
↪          Setting storage dispatch to 15.0 kW
2018-01-08 17:41:46,577 (referenceappagent-1.0 23842) referenceapp.agent DEBUG:␣
↪Received event: ID=4, status=far, start=2017-12-01 18:40:55+00:00, end=2017-12-02␣
↪18:37:56+00:00, opt_type=none, all params={"status": "far", "signals": "{\"null\":
↪{\"intervals\": {\"0\": {\"duration\": \"PT23H57M1S\", \"uid\": \"0\", \"payloads\
↪": {}}}, \"currentLevel\": null, \"signalID\": null}}", "event_id": "4", "start_time
↪": "2017-12-01 18:40:55+00:00", "creation_time": "2018-01-08 17:41:45.774548", "opt_
↪type": "none", "priority": 1, "end_time": "2017-12-02 18:37:56+00:00"}
2018-01-08 17:41:46,577 (referenceappagent-1.0 23842) referenceapp.agent DEBUG:␣
↪Sending an optIn response for event ID 4
2018-01-08 17:41:46,583 (referenceappagent-1.0 23842) referenceapp.agent DEBUG: 2018-
↪01-08 17:41:46.576130 Reporting at sim time 2018-01-08 17:41:46.328388
```

(continues on next page)

(continued from previous page)

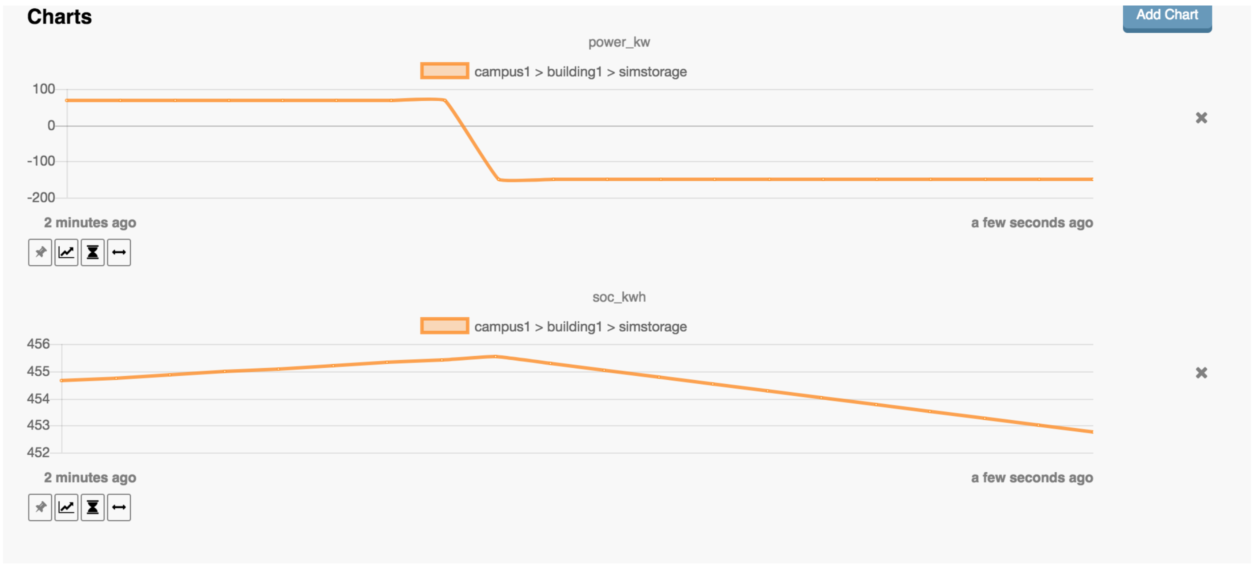
```
2018-01-08 17:41:46,583 (referenceappagent-1.0 23842) referenceapp.agent DEBUG:     ␣
↪  devices/simload/power_kw = 519.3
2018-01-08 17:41:46,583 (referenceappagent-1.0 23842) referenceapp.agent DEBUG:     ␣
↪  devices/simpv/power_kw = -17.175
2018-01-08 17:41:46,583 (referenceappagent-1.0 23842) referenceapp.agent DEBUG:     ␣
↪  devices/simstorage/dispatch_kw = 15.0
2018-01-08 17:41:46,584 (referenceappagent-1.0 23842) referenceapp.agent DEBUG:     ␣
↪  devices/simstorage/power_kw = 15.0
2018-01-08 17:41:46,584 (referenceappagent-1.0 23842) referenceapp.agent DEBUG:     ␣
↪  devices/simstorage/soc_kwh = 25.025
2018-01-08 17:41:46,584 (referenceappagent-1.0 23842) referenceapp.agent DEBUG:     ␣
↪  net_power_kw = 49.755
2018-01-08 17:41:46,584 (referenceappagent-1.0 23842) referenceapp.agent DEBUG:     ␣
↪  report_time = 2018-01-08 17:41:46.328388
2018-01-08 17:41:46,596 (referenceappagent-1.0 23842) referenceapp.agent DEBUG:     ␣
↪          Setting storage dispatch to 15.0 kW
2018-01-08 17:41:48,617 (referenceappagent-1.0 23842) referenceapp.agent DEBUG:␣
↪Received event: ID=4, status=completed, start=2017-12-01 18:40:55+00:00, end=2017-
↪12-02 18:37:56+00:00, opt_type=optIn, all params={"status": "completed", "signals":
↪"{\"null\": {\"intervals\": {\"0\": {\"duration\": \"PT23H57M1S\", \"uid\": \"0\", \
↪"payloads\": {}}}, \"currentLevel\": null, \"signalID\": null}}", "event_id": "4",
↪"start_time": "2017-12-01 18:40:55+00:00", "creation_time": "2018-01-08 17:41:45.
↪774548", "opt_type": "optIn", "priority": 1, "end_time": "2017-12-02 18:37:56+00:00
↪"}
2018-01-08 17:42:59,563 (referenceappagent-1.0 23842) referenceapp.agent DEBUG: 2018-
↪01-08 17:42:59.559264 Reporting at sim time 2018-01-08 17:42:59.328388
2018-01-08 17:42:59,563 (referenceappagent-1.0 23842) referenceapp.agent DEBUG:     ␣
↪  devices/simload/power_kw = 519.3
2018-01-08 17:42:59,563 (referenceappagent-1.0 23842) referenceapp.agent DEBUG:     ␣
↪  devices/simpv/power_kw = -17.175
2018-01-08 17:42:59,563 (referenceappagent-1.0 23842) referenceapp.agent DEBUG:     ␣
↪  devices/simstorage/dispatch_kw = 15.0
2018-01-08 17:42:59,563 (referenceappagent-1.0 23842) referenceapp.agent DEBUG:     ␣
↪  devices/simstorage/power_kw = 15.0
2018-01-08 17:42:59,563 (referenceappagent-1.0 23842) referenceapp.agent DEBUG:     ␣
↪  devices/simstorage/soc_kwh = 25.238
2018-01-08 17:42:59,563 (referenceappagent-1.0 23842) referenceapp.agent DEBUG:     ␣
↪  net_power_kw = 49.755
2018-01-08 17:42:59,563 (referenceappagent-1.0 23842) referenceapp.agent DEBUG:     ␣
↪  report_time = 2018-01-08 17:42:59.328388
2018-01-08 17:42:59,578 (referenceappagent-1.0 23842) referenceapp.agent DEBUG:     ␣
↪          Setting storage dispatch to -1.05158333333 kW
2018-01-08 17:43:01,596 (referenceappagent-1.0 23842) referenceapp.agent DEBUG: 2018-
↪01-08 17:43:01.589877 Reporting at sim time 2018-01-08 17:43:01.328388
2018-01-08 17:43:01,596 (referenceappagent-1.0 23842) referenceapp.agent DEBUG:     ␣
↪  devices/simload/power_kw = 519.3
2018-01-08 17:43:01,596 (referenceappagent-1.0 23842) referenceapp.agent DEBUG:     ␣
↪  devices/simpv/power_kw = -17.175
2018-01-08 17:43:01,597 (referenceappagent-1.0 23842) referenceapp.agent DEBUG:     ␣
↪  devices/simstorage/dispatch_kw = -1.05158333333
2018-01-08 17:43:01,597 (referenceappagent-1.0 23842) referenceapp.agent DEBUG:     ␣
↪  devices/simstorage/power_kw = -1.051
2018-01-08 17:43:01,597 (referenceappagent-1.0 23842) referenceapp.agent DEBUG:     ␣
↪  devices/simstorage/soc_kwh = 25.236
2018-01-08 17:43:01,597 (referenceappagent-1.0 23842) referenceapp.agent DEBUG:     ␣
↪  net_power_kw = 33.704
2018-01-08 17:43:01,597 (referenceappagent-1.0 23842) referenceapp.agent DEBUG:     ␣
↪  report_time = 2018-01-08 17:43:01.328388
```

(continues on next page)

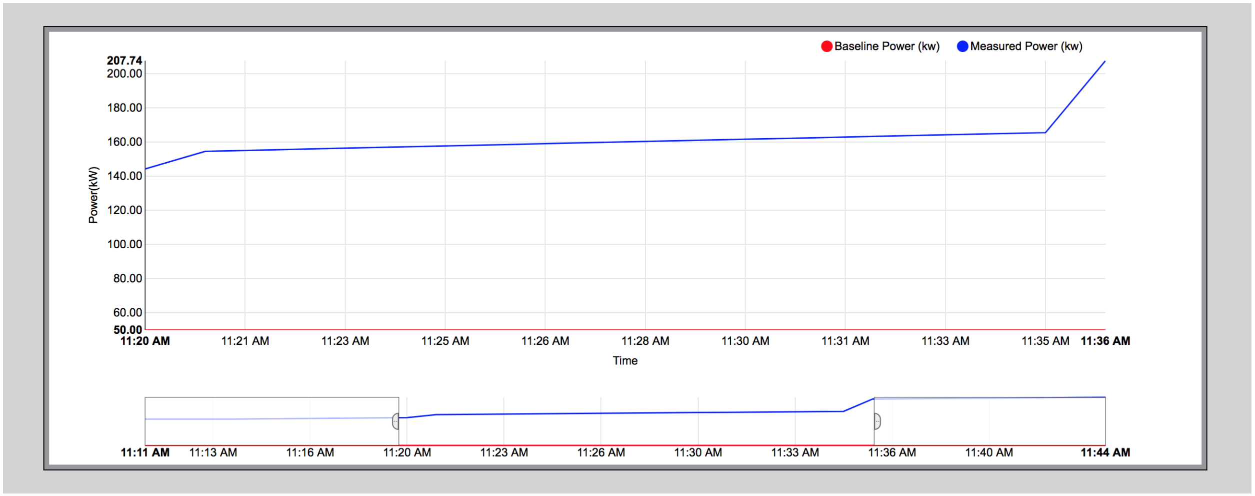
(continued from previous page)

```
2018-01-08 17:43:01,598 (referenceappagent-1.0 23842) referenceapp.agent DEBUG:␣
↪Reporting telemetry: {'start_time': '2018-01-08 17:42:31.598889+00:00', 'baseline_
↪power_kw': '50', 'current_power_kw': '33.704', 'end_time': '2018-01-08 17:43:01.
↪598889+00:00'}
2018-01-08 17:43:01,611 (referenceappagent-1.0 23842) referenceapp.agent DEBUG:      ␣
↪          Setting storage dispatch to -1.0515 kW
```

Another way to monitor progress is to launch the VolttronCentral web UI, which can be found at **http://127.0.0.1:8080/vc/index.html**. Here, in addition to checking agent status, one can track metrics reported by the simulated device drivers. For example, these graphs track the simstorage battery's power consumption and state of charge over time. The abrupt shift from charging to discharging happens because an OpenADR event has just started:

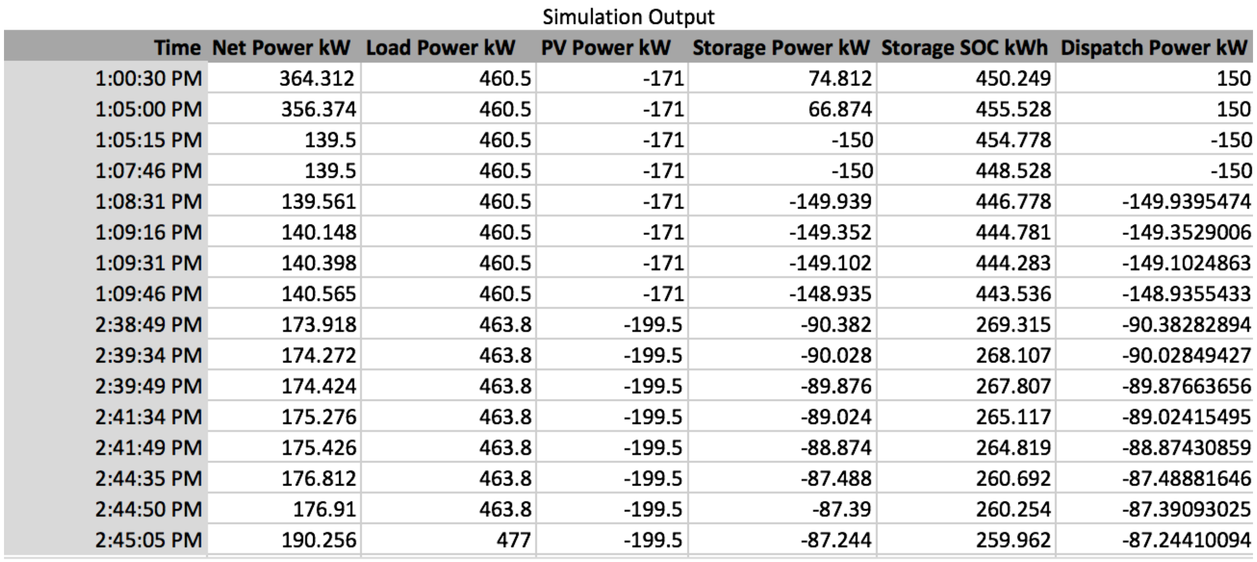


A third way to monitor progress, while there is an active DR event, is to examine the event's graph in the VTN web UI. This displays the VEN's power consumption, which is an aggregate of the consumption reported by each simulated device driver:



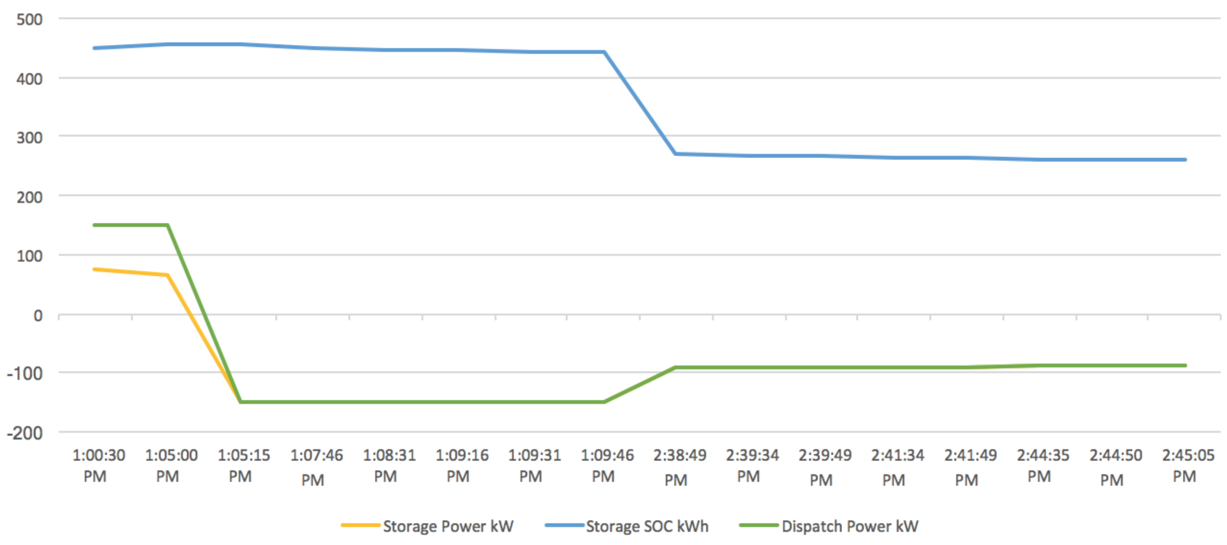
Report Output

The ReferenceAppAgent also writes a CSV output file so that simulation results can be reported in a spreadsheet, for example this graph of the simulated storage device:



Simulation Output

Time	Net Power kW	Load Power kW	PV Power kW	Storage Power kW	Storage SOC kWh	Dispatch Power kW
1:00:30 PM	364.312	460.5	-171	74.812	450.249	150
1:05:00 PM	356.374	460.5	-171	66.874	455.528	150
1:05:15 PM	139.5	460.5	-171	-150	454.778	-150
1:07:46 PM	139.5	460.5	-171	-150	448.528	-150
1:08:31 PM	139.561	460.5	-171	-149.939	446.778	-149.9395474
1:09:16 PM	140.148	460.5	-171	-149.352	444.781	-149.3529006
1:09:31 PM	140.398	460.5	-171	-149.102	444.283	-149.1024863
1:09:46 PM	140.565	460.5	-171	-148.935	443.536	-148.9355433
2:38:49 PM	173.918	463.8	-199.5	-90.382	269.315	-90.38282894
2:39:34 PM	174.272	463.8	-199.5	-90.028	268.107	-90.02849427
2:39:49 PM	174.424	463.8	-199.5	-89.876	267.807	-89.87663656
2:41:34 PM	175.276	463.8	-199.5	-89.024	265.117	-89.02415495
2:41:49 PM	175.426	463.8	-199.5	-88.874	264.819	-88.87430859
2:44:35 PM	176.812	463.8	-199.5	-87.488	260.692	-87.48881646
2:44:50 PM	176.91	463.8	-199.5	-87.39	260.254	-87.39093025
2:45:05 PM	190.256	477	-199.5	-87.244	259.962	-87.24410094



For Further Information

If you have comments or questions about this simulation support, please contact Rob Calvert or Nate Hill at Kisensum, Inc.:

- (github) @rob-calvert
- (email) rob@kisensum.com
- (github) @hillrnate
- (github) nate@kisensum.com

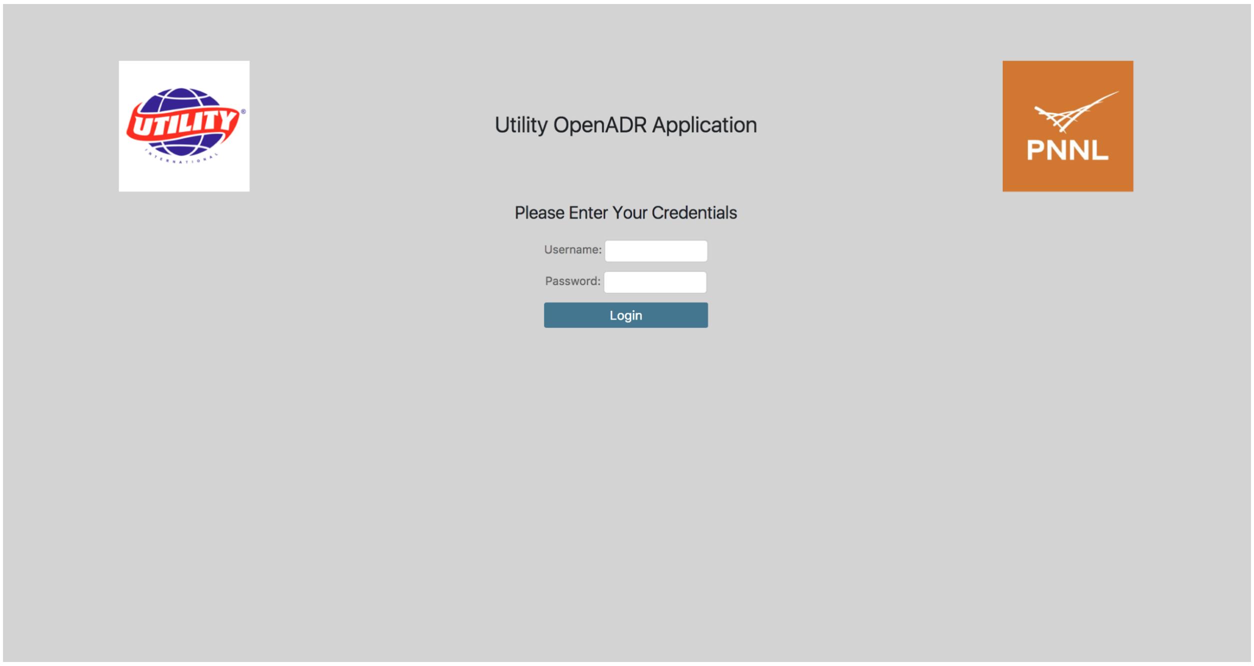
OpenADR VTN Server: User Guide

Warning: This VTN server implementation is incomplete, and is not supported by the VOLTTRON core team. For information about its status including known issues, refer to the *VTN Server Configuration docs*.

This guide assumes that you have a valid user account to access and log in to the VTN application website.

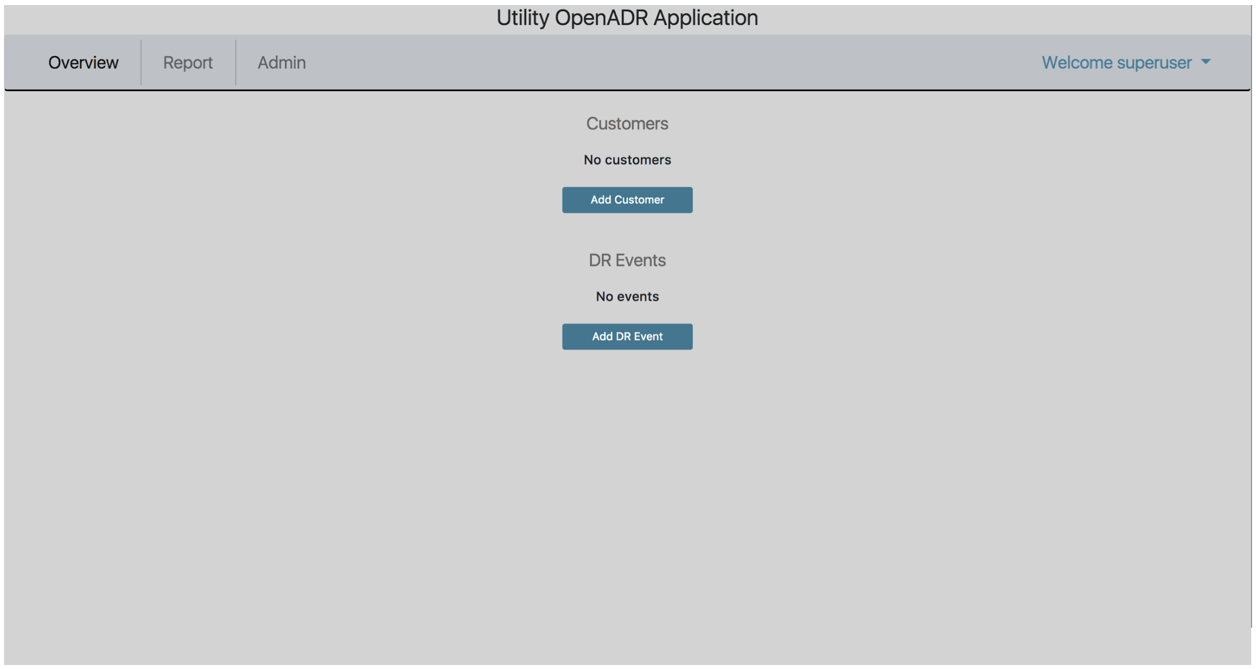
Login Screen

In order to begin using the VTN application, navigate to `\http://yourhostname*<or>*ip:8000/vtn`.



Overview Screen

Once logged in for the first time, this is the 'Overview' screen.



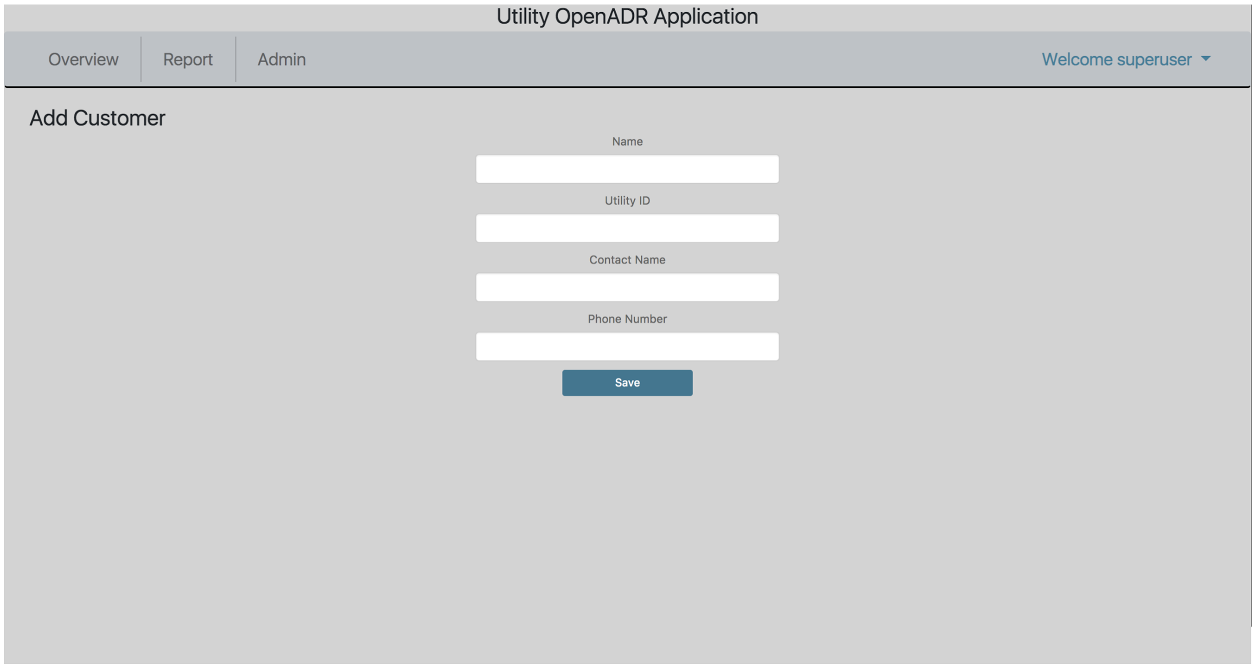
In order to begin scheduling DR events, one must first create at least one customer, with at least one associated site/VEN, and at least one sort of demand response (DR) program. A VTN will not be able to tell a VEN about DR Events if the VTN doesn't know about the VEN. A VTN knows about a VEN after a Site for the VEN has been created in the VTN application, and the VEN has contacted the VTN.

The rest of this document describes how to set up Customers, Sites, DR Programs, and DR Events, as well as how to export event data.

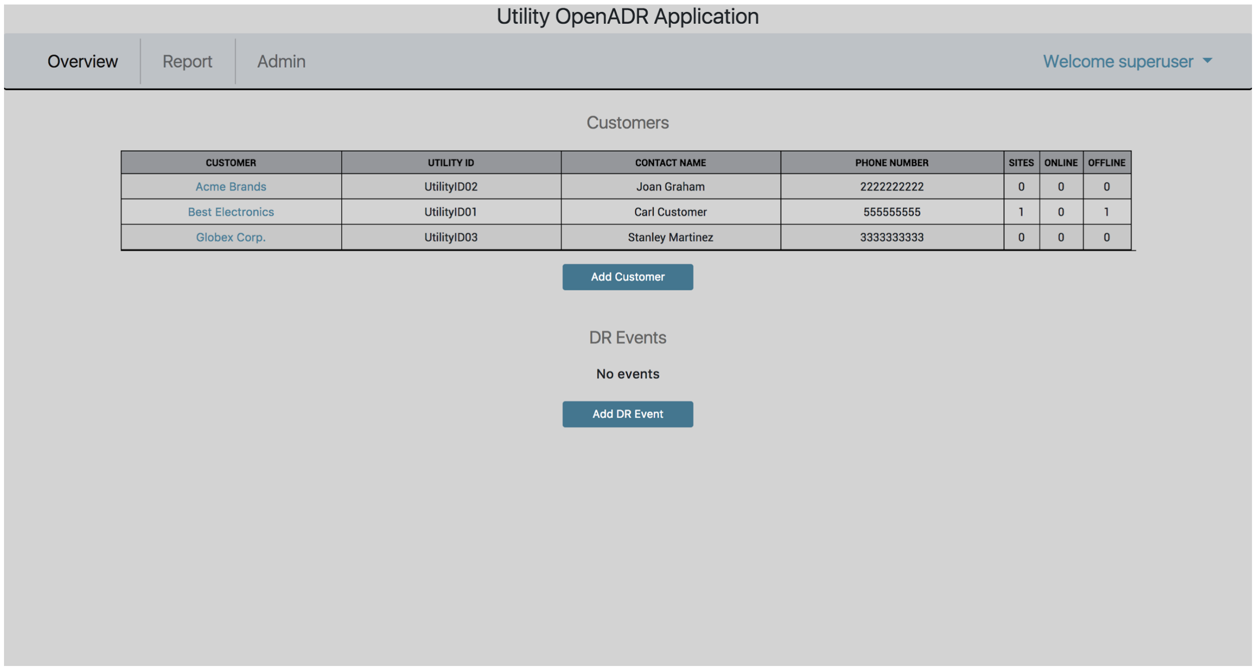
Create a Customer

Creating a Customer can be done by clicking on 'Add Customer' on the **Overview** screen.

The standard interface for adding a Customer:



Customers will appear on the **Overview** screen after they have been added.

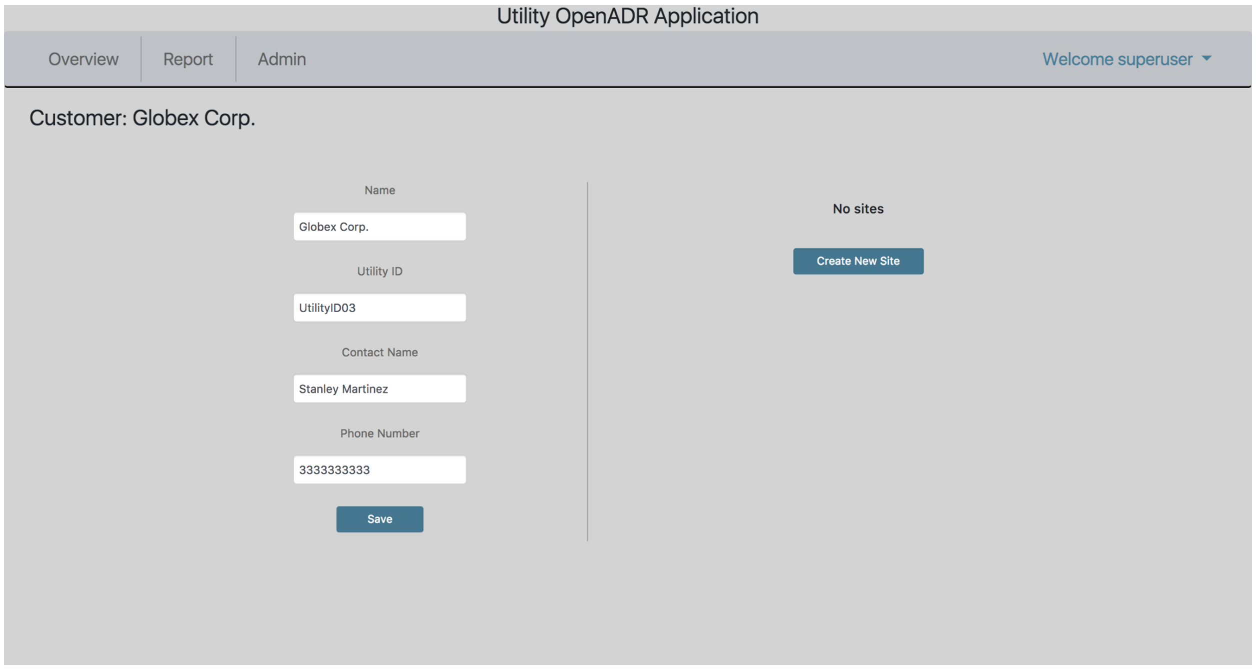


CUSTOMER	UTILITY ID	CONTACT NAME	PHONE NUMBER	SITES	ONLINE	OFFLINE
Acme Brands	UtilityID02	Joan Graham	2222222222	0	0	0
Best Electronics	UtilityID01	Carl Customer	555555555	1	0	1
Globex Corp.	UtilityID03	Stanley Martinez	3333333333	0	0	0

No events

Create a Site

At first, Customers will not have any Sites. To add a Site for a Customer, click on the Customer's name from the **Overview** screen, and then click 'Create New Site'.



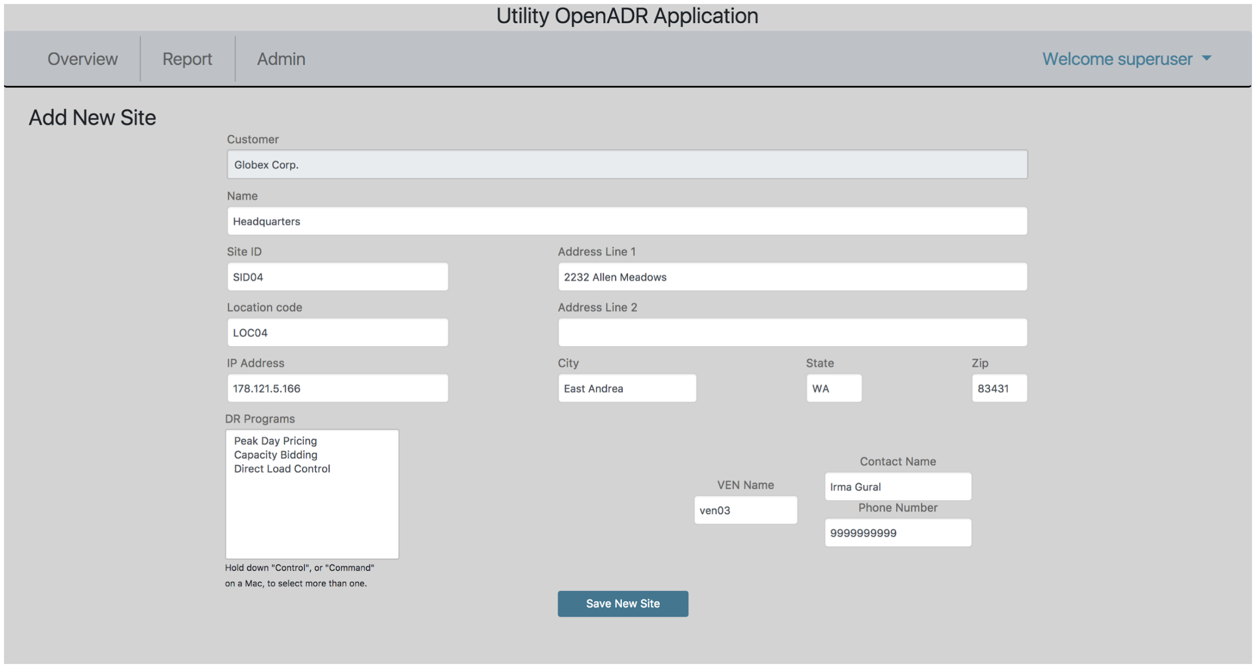
Customer: Globex Corp.

Name	Globex Corp.
Utility ID	UtilityID03
Contact Name	Stanley Martinez
Phone Number	3333333333

No sites

On the **Create Site** screen, DR Programs will appear in the 'DR Programs' multiple-select box if they have been added. This will be discussed soon. Selecting one or more DR Programs here means, when creating a DR Event with a specified DR Program, the site will be an available option for the given DR Event.

A site's 'VEN Name' is permanent. In order to change a Site's VEN Name, the Site must be deleted and re-added.

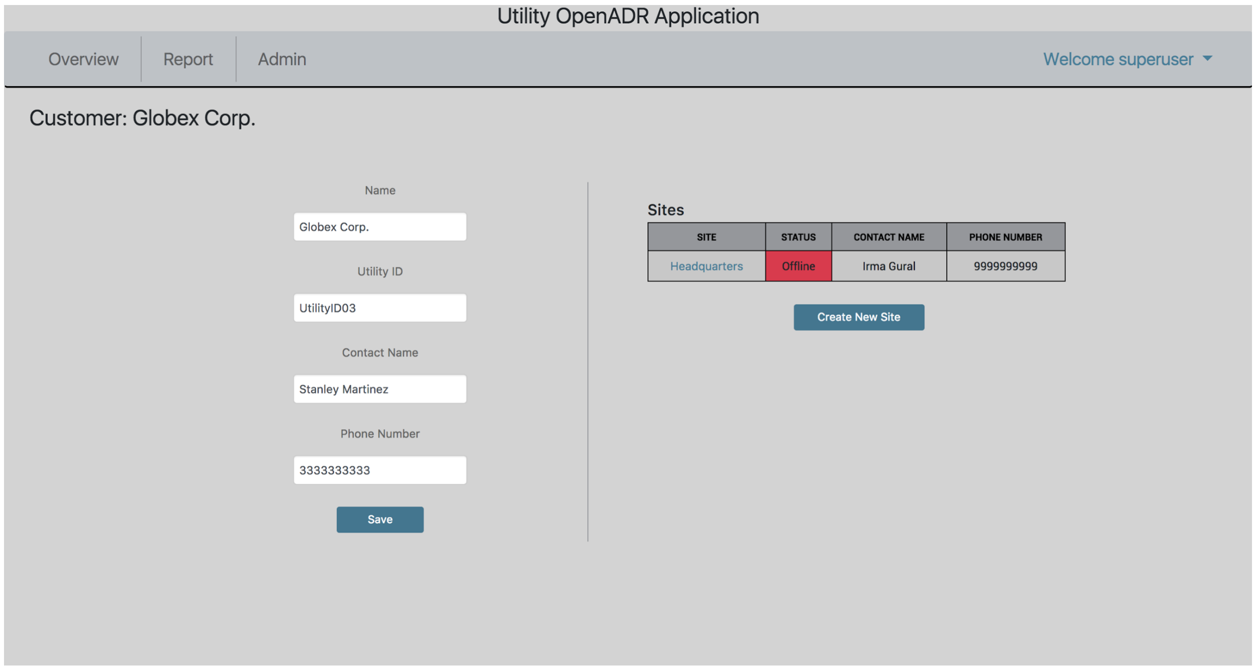


Add New Site

Customer	Globex Corp.
Name	Headquarters
Site ID	SID04
Address Line 1	2232 Allen Meadows
Location code	LOC04
Address Line 2	
IP Address	178.121.5.166
City	East Andrea
State	WA
Zip	83431
DR Programs	Peak Day Pricing Capacity Bidding Direct Load Control
VEN Name	ven03
Contact Name	Irma Gural
Phone Number	9999999999

Hold down "Control", or "Command" on a Mac, to select more than one.

After creating a Site for a given customer, the Site will appear offline until communication has been established with the Site's VEN within a configurable interval (default is 15 minutes).



Customer: Globex Corp.

Name

Globex Corp.

Utility ID

UtilityID03

Contact Name

Stanley Martinez

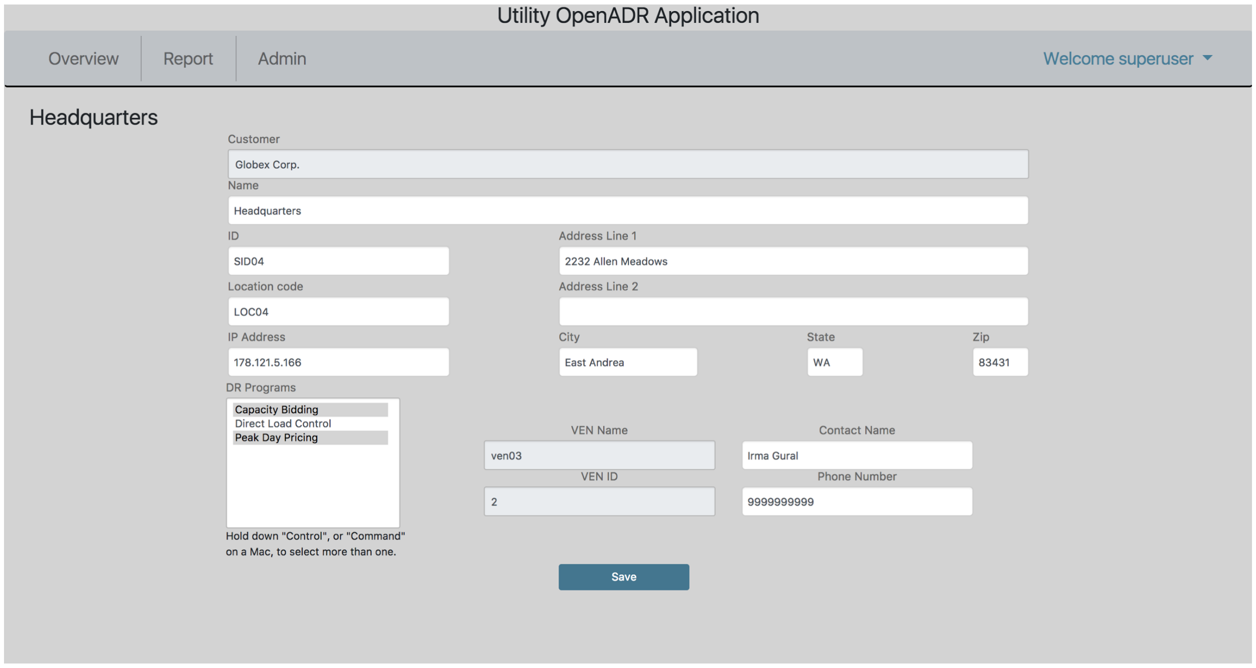
Phone Number

3333333333

Sites

SITE	STATUS	CONTACT NAME	PHONE NUMBER
Headquarters	Offline	Irma Gural	9999999999

Note: When editing a Site, you will notice an extra field on the Screen labeled 'VEN ID'. This field is assigned automatically upon creation of a Site and is used by the VTN to communicate with and identify the VEN.



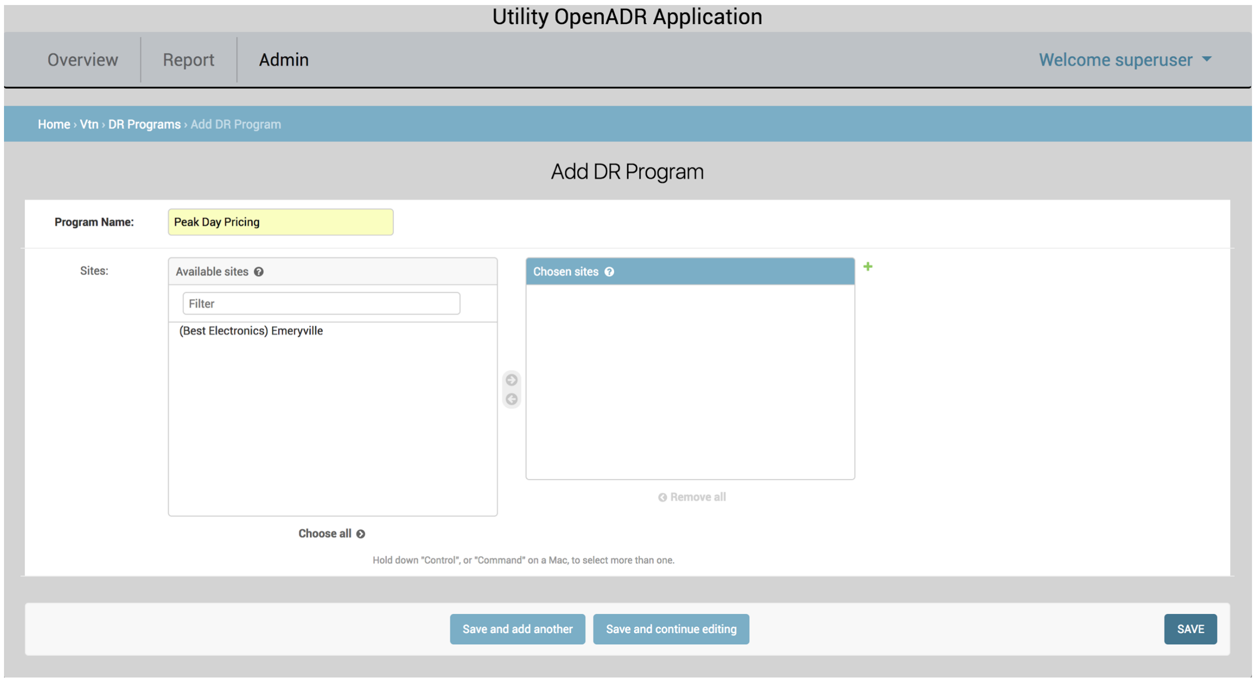
Headquarters

Customer	Globex Corp.
Name	Headquarters
ID	SID04
Address Line 1	2232 Allen Meadows
Location code	LOC04
Address Line 2	
IP Address	178.121.5.166
City	East Andrea
State	WA
Zip	83431
DR Programs	Capacity Bidding, Direct Load Control, Peak Day Pricing
VEN Name	ven03
VEN ID	2
Contact Name	Irma Gural
Phone Number	9999999999

Hold down "Control", or "Command" on a Mac, to select more than one.

Create a DR Program

DR Programs must be added via the Admin interface. DR Programs can be added with or without associated sites. In other words, a DR Program can be created with no sites, and sites can be added later, either by Creating/Editing a Site and selecting the DR Program, or by Creating/Editing the DR Program and adding the Site.


Utility OpenADR Application
Overview
Report
Admin
Welcome superuser
Home › Vtn › DR Programs › Add DR Program
Add DR Program

Program Name: Peak Day Pricing

Sites:

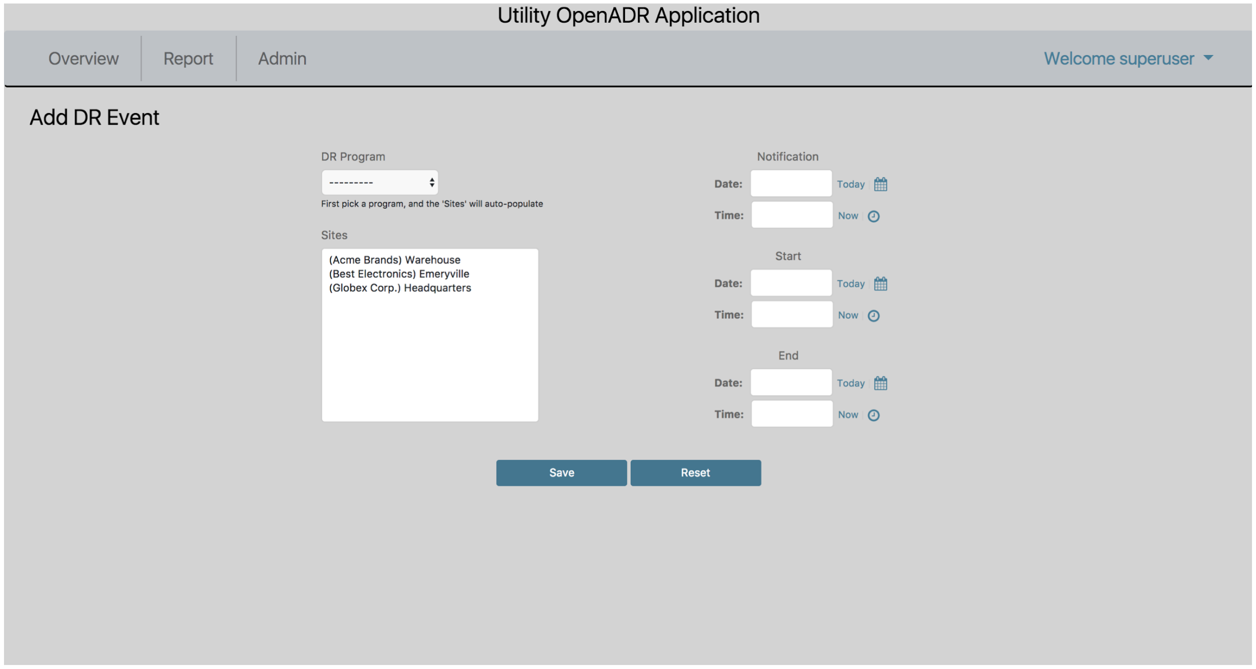
Available sites
Filter
(Best Electronics) Emeryville
Chosen sites
Remove all
Choose all
Hold down "Control", or "Command" on a Mac, to select more than one.
Save and add another
Save and continue editing
SAVE

Create a DR Event

Once a Customer (with at least one site) and a DR Program have been added, a DR Event can be created. This is done by navigating to the **Overview** screen and clicking 'Add DR Event'.

On the **Add DR Event** screen, the first step is to select a DR Program from the drop-down menu. Once a DR Program is selected, the 'Sites' multi-select box will auto-populate with the Sites that are associated with that DR Program.

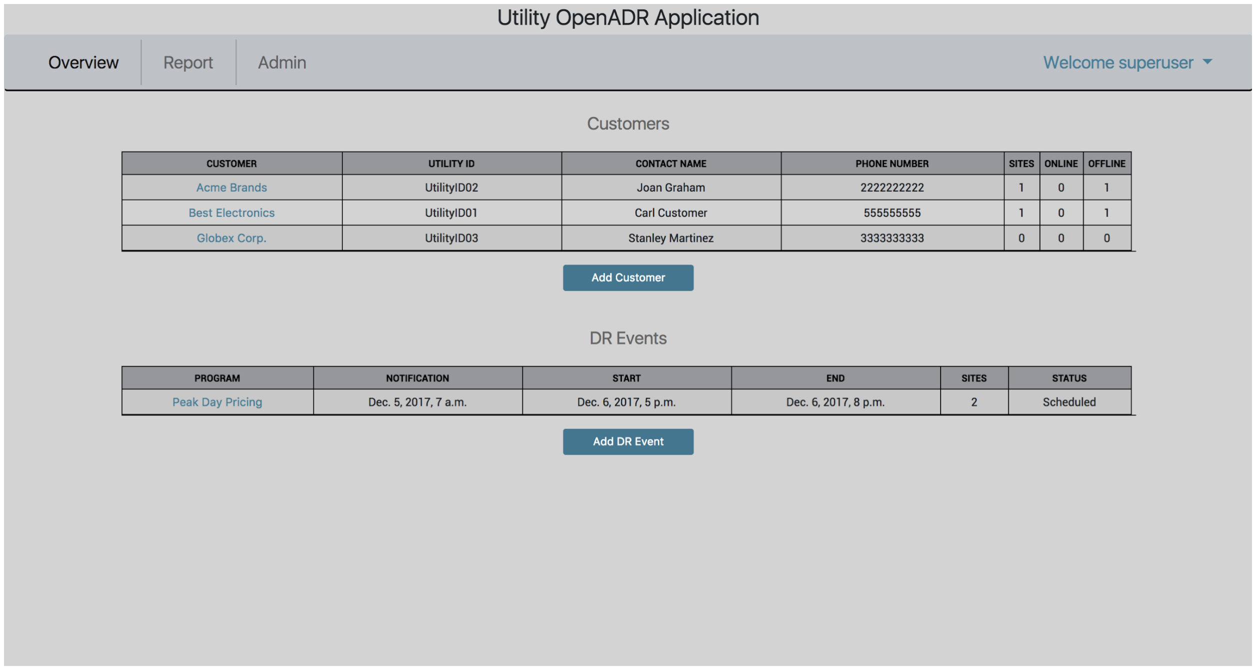
Note that the Notification Time is the absolute soonest time that a VEN will be notified of a DR Event. VENs will not 'know' about DR Events that apply to them until they have 'polled' the VTN after the Notification Time.


Utility OpenADR Application
Overview
Report
Admin
Welcome superuser
Add DR Event
DR Program

First pick a program, and the 'Sites' will auto-populate
Sites

(Acme Brands) Warehouse
(Best Electronics) Emeryville
(Globex Corp.) Headquarters

Active DR events are displayed on the **Overview** screen. DR Events are considered active if they have not been canceled and if they have not been completed.



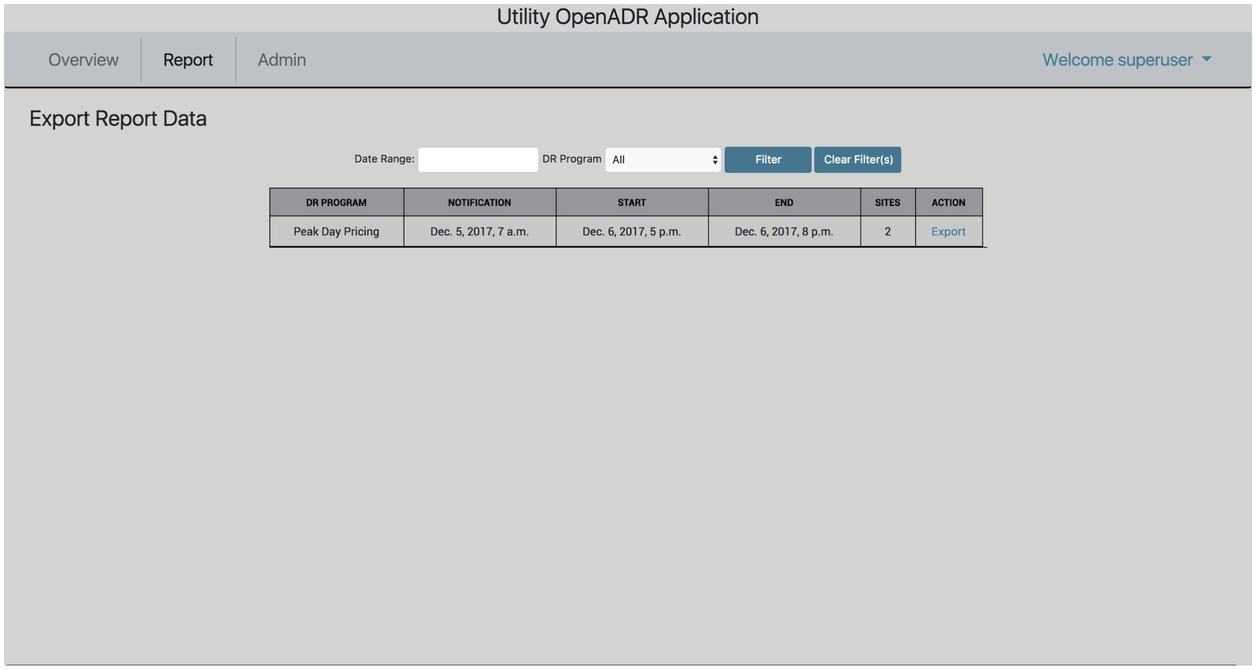
Customers

CUSTOMER	UTILITY ID	CONTACT NAME	PHONE NUMBER	SITES	ONLINE	OFFLINE
Acme Brands	UtilityID02	Joan Graham	2222222222	1	0	1
Best Electronics	UtilityID01	Carl Customer	555555555	1	0	1
Globex Corp.	UtilityID03	Stanley Martinez	3333333333	0	0	0

DR Events

PROGRAM	NOTIFICATION	START	END	SITES	STATUS
Peak Day Pricing	Dec. 5, 2017, 7 a.m.	Dec. 6, 2017, 5 p.m.	Dec. 6, 2017, 8 p.m.	2	Scheduled

Exporting event telemetry to a .csv is available on the **Report** tab. In the case of this VTN and its associated VENs, the telemetry that will be reported include **baseline power (kw)** and **measured power (kw)**.



Export Report Data

DR PROGRAM	NOTIFICATION	START	END	SITES	ACTION
Peak Day Pricing	Dec. 5, 2017, 7 a.m.	Dec. 6, 2017, 5 p.m.	Dec. 6, 2017, 8 p.m.	2	Export

OpenADR VTN Server: Installation and Configuration

The OpenADR VTN server is a partial implementation of the OpenADR VTN specification developed by Kisensum for interoperability with the VOLTTRON core VEN agent implementation. The VTN server resides in the VOLTTRON applications repository, and is not supported by the VOLTTRON core team.

Known issues: The Kisensum implementation of the VTN server does not currently include support for registration, including QueryRegistration requests, create and cancel party requests, etc. Additionally, it does not implement opt-in behavior as specified by OpenADR. Finally, it has been found that requests containing empty basic authentication will be served a 403 error, while requests with no authentication will proceed to the correct endpoint normally.

The Kisensum VTN server is a Django application written in Python 3 and utilizing a Postgres database.

Warning: If you are planning to install your VTN server on the same system that contains your VOLTTRON instance and you are using RabbitMQ with VOLTTRON, you will need to set up a new instance of RabbitMQ for VTN. In production, the VTN server should be on a different device than VOLTTRON, and as such it is recommended that your VTN server is in it's own VM or on it's own machine. If you still wish to set up two instances of RabbitMQ on the same system, please refer to https://www.rabbitmq.com for further details.

Get Source Code

To install the VTN server, first get the code by cloning volttron-applications from github and checking out the openadr software.

```
$ cd ~/repos
$ git clone https://github.com/volttron/volttron-applications
$ cd volttron-applications
$ git checkout master
```

Install Python 3

After installing Python3 on the server, configure an openadr virtual environment:

```
$ sudo pip install virtualenvwrapper
$ mkdir ~/.virtualenvs (if it doesn't exist already)
```

Edit **~/.bashrc** and add these lines:

```
export WORKON_HOME=$HOME/.virtualenvs
export PROJECT_HOME=$HOME/repos/volttron-applications/kisensum/openadr
source virtualenvwrapper.sh
```

Create the openadr project's virtual environment:

```
$ source ~/.bashrc
$ mkvirtualenv -p /usr/bin/python3 openadr
$ setvirtualenvproject openadr ~/repos/volttron-applications/kisensum/openadr
$ workon openadr
```

From this point on, use **workon openadr** to operate within the openadr virtual environment.

Create a local site override for Django's base settings file as follows. First, create **~/.virtualenvs/openadr/.settings** in a text editor, adding the following line to it:

```
openadr.settings.site
```

Then, edit **~/.virtualenvs/openadr/postactivate**, adding the following lines:

```
PROJECT_PATH=`cat "$VIRTUAL_ENV/$VIRTUALENVWRAPPER_PROJECT_FILENAME"`
PROJECT_ROOT=`dirname $PROJECT_PATH`
PROJECT_NAME=`basename $PROJECT_PATH`
SETTINGS_FILENAME=".settings"
ENV_FILENAME=".env_postactivate.sh"

# Load the default DJANGO_SETTINGS_MODULE from a .settings
# file in the django project root directory.
export OLD_DJANGO_SETTINGS_MODULE=$DJANGO_SETTINGS_MODULE
if [ -f $VIRTUAL_ENV/$SETTINGS_FILENAME ]; then
   export DJANGO_SETTINGS_MODULE=`cat "$VIRTUAL_ENV/$SETTINGS_FILENAME"`
fi
```

Finally, create **$PROJECT_HOME/openadr/openadr/openadr/settings/site.py**, which holds overrides to base.py, the Django base settings file. At a minimum, this file should contain the following:

```
from .base import *
ALLOWED_HOSTS = ['*']
```

A more restrictive ALLOWED_HOSTS setting (e.g. 'ki-evi.com') should be used in place of '*' if it is known.

Use Pip to Install Third-Party Software

```
$ workon openadr
$ pip install -r requirements.txt
```

Set up a Postgres Database

Install postgres.

Create a postgres user.

Create a postgres database named openadr.

(The user name, user password, and database name must match what is in **$PROJECT_HOME/openadr/openadr/settings/base.py** or the override settings in **$PROJECT_HOME/openadr/openadr/settings/local.py**.)

You may have to edit **/etc/postgresql/9.5/main/pg_hba.conf** to be 'md5' authorization for 'local'.

Migrate the Database and Create an Initial Superuser

```
$ workon openadr
$ cd openadr
$ python manage.py migrate
$ python manage.py createsuperuser
```

This is the user that will be used to login to the VTN application for the first time, and will be able to create other users and groups.

Configure Rabbitmq

rabbitmq is used by celery, which manages the openadr server's periodic tasks.

Install and run rabbitmq as follows (for further information, see http://www.rabbitmq.com/download.html):

```
$ sudo apt-get install rabbitmq-server
```

Start the rabbitmq server if it isn't already running:

```
$ sudo rabbitmq-server -detached (note the single dash)
```

Start the VTN Server

```
$ workon openadr
$ cd openadr
$ python manage.py runserver 0.0.0.0:8000
```

Start Celery

```
$ workon openadr
$ cd openadr
$ celery -A openadr worker -B
```

Configuration Parameters

The VTN supports the following configuration parameters, which can be found in **base.py** and overriden in **site.py**:

Parameter	Example	Description
VTN_ID	"vtn01"	OpenADR ID of this virtual top node. Virtual end nodes must know this VTN_ID to be able to communicate with the VTN.
ONLINE_INTERVAL_MINUTES	15	The amount of time, in minutes, that determines how long the VTN will wait until displaying a given VEN offline. In other words, if the VTN does not receive any communication from a given VEN within ONLINE_INTERVAL_MINUTES minutes, the VTN will display said VEN as offline.
GRAPH_TIMECHUNK_SECONDS	360	The VTN displays DR Event graph data by averaging individual VENs' telemetry by GRAPH_TIMECHUNK_SECONDS seconds. This value should be adjusted according to how often VENs are sending the VTN telemetry.

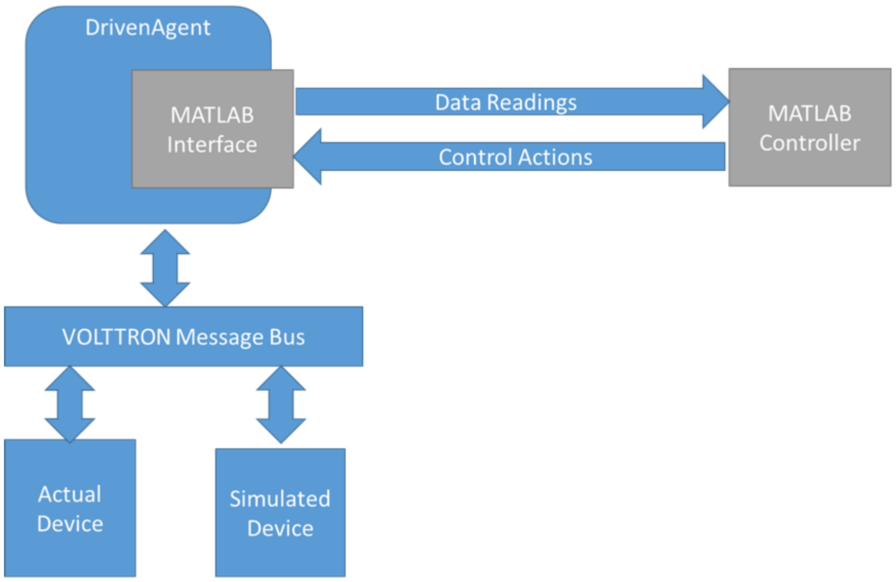
1.46.4 MatLab Integration

Overview:

Matlab-VOLTTRON integration allows Matlab applications to receive data from devices and send control commands to change points on those devices.

DrivenMatlabAgent in VOLTTRON allows this interaction by using ZeroMQ sockets to communicate with the Matlab application.

Data Flow Architecture:



Installation steps for system running Matlab:

1. Install python. Suggested 3.6.
2. Install pyzmq (tested with version 15.2.0) Follow steps at: https://github.com/zeromq/pyzmq
3. Install Matlab (tested with R2015b)
4. Start Matlab and set the python path. In the Matlab command window set the python path with *pyversion*:

```
>> pyversion python.exe
```

5. To test that the python path has been set correctly type following in the Matlab command window. Matlab shoud print the python path with version information.

```
>> pyversion
```

6. To test that the pyzmq library is installed correctly and is accessible from python inside Matlab, type the following in Matlab command window and it should show pyzmq version installed.

```
>> py.zmq.pyzmq_version()
```

7. Copy *example.m* from *volttron/examples/ExampleMatlabApplication/matlab* to your desired folder.

Run and test Matlab VOLTTRON Integration:

Assumptions

- Device driver agent is already developed

Installation:

1. Install VOLTTRON on a VM or different system than the one running Matlab.

   Follow link: http://volttron.readthedocs.io/en/develop/install.html

2. Add subtree volttron-applications under volttron/applications by using the following command:

```
git subtree add --prefix applications https://github.com/VOLTTRON/volttron-
↪applications.git develop --squash
```

Configuration

1. Copy example configuration file *applications/pnnl/DrivenMatlabAgent/config_waterheater* to *volttron/config*.
2. Change config_url and data_url in the new config file to the ipaddress of machine running Matlab. Keep the same port numbers.
3. Change campus, building and unit (device) name in the config file.
4. Open example.m and change following line:

```
matlab_result = '{"commands":{"Zone1":[["temperature",27]],"Zone2":[["temperature",
↪28]]}}';
```

Change it to include correct device name and point names in the format:

```
'{"commands":{"device1":[["point1",value1]],"device2":[["point2",value2]]}}';
```

Steps to test integration:

1. Start VOLTTRON
2. Run Actuator
3. Run device driver agent
4. Run DrivenMatlabAgent with the new config file
5. Run example.m in Matlab

Now whenever the device driver publishes the state of devices listed in the config file of DrivenMatlabAgent, DrivenMatlabAgent will send it to Matlab application and receive commands to send to devices.

Resources

http://www.mathworks.com/help/matlab/getting-started_buik_wp-3.html

1.47 Change Log

This section includes individual documents describing important changes to platform components, such as the RabbitMQ message bus implementation. For information on specific changes, please refer to the corresponding document.

1.47.1 Scalability Setup

Core Platform

- VIP router - how many messages per second can the router pass
- A single agent can connect and send messages to itself as quickly as possible
- Repeat but with multiple agents
- Maybe just increase the number of connected but inactive agents to test lookup times
- Inject faults to test impact of error handling
- Agents
- How many can be started on a single platform?
- How does it affect memory?
- How is CPU affected?

Socket types

- inproc - lockless, copy-free, fast
- ipc - local, reliable, fast
- tcp - remote, less reliable, possibly much slower
- test with different
  - latency
  - throughput
  - jitter (packet delay variation)
  - error rate

Subsystems

- ping - simple protocol which can provide baseline for other subsystems
- RPC - requests per second
- Pub/Sub - messages per second
- How does it scale as subscribers are added

Core Services

- historian
- How many records can be processed per second?
- drivers
- BACnet drivers use a virtual BACnet device as a proxy to do device communication. Currently there is no known upper limit to the number of devices that can be handled at once. The BACnet proxy opens a single UDP port to do all communication. In theory the upper limit is the point when UDP packets begin to be lost due to network congestion. In practice we have communicated with ~190 devices at once without issue.

- ModBUS opens up a TCP connection for each communication with a device and then closes it when finished. This has the potential to hit the limit for open file descriptors available to the master driver process. (Before, each driver would run in a separate process, but that quickly uses up sockets available to the platform.) To protect from this the master driver process raises the total allowed open sockets to the hard limit. The number of concurrently open sockets is throttled at 80% of the max sockets. On most Linux systems this is about 3200. Once that limit is hit additional device communications will have to wait in line for a socket to become available.

Tweaking tests

- Configure message size
- Perform with/without encryption
- Perform with/without authentication

Hardware profiling

- Perform tests on hardware of varying resources: Raspberry Pi, NUC, Desktop, etc.

Scenarios

- One platform controlling large numbers of devices
- One platform managing large numbers of platforms
- Peer communication (Hardware demo type setup)

Impact on Platform

What is the impact of a large number of devices being scraped on a platform (and how does it scale with the hardware)?

- Historians
- At what point are historians unable to keep up with the traffic being generated?
- Is the bottleneck the sqlite cache or the specific implementation (SQLite, MySQL)
- Do historian queues grow so large we have a memory problem?
- Large number of devices with small number of points vs small number of devices with large number of points
- How does a large message flow affect the router?
- Examine effects of the watermark (does increasing help)
- Response time for vctl commands (for instance: status)
- Affect on round trip times (Agent A sends message, Agent B replies, Agent A receives reply)
- Do messages get lost at some point (EAgain error)?
- What impact does security have? Are things significantly faster in developer-mode? (Option to turn off encryption, no longer available)
- Regulation Agent
  Every 10 minutes there is an action the master node determines. Duty cycle cannot be faster than that but is set to 2 seconds for simulation.
  Some clients miss duty cycle signal
  Mathematically each node solves ODE.

Model notes accept switch on/off from master.
Bad to lose connection to clients in the field

Chaos router to introduce delays and dropped packets.

MasterNode needs to have vip address of clients.

Experiment capture historian - not listening to devices, just capturing results

- Go straight to db to see how far behind other historians

Improvements Based on Results

Here is the list of scalability improvements so far:

Reduced the overhead of the base historian by removing unneeded writes to the backup db. Significantly improved performance on low end devices.

Added options to reduce the number of publishes per device per scrape. Common cases where per point publishes and breadth first topics are not needed the driver can be configured only publish the depth first "all" or any combination per device the operator needs. This dramatically decreases the platform hardware requirements while increasing the number of devices that can be scraped.

Added support for staggering device scrapes to reduce CPU load during a scrape.

Further ideas:

Determine if how we use ZMQ is reducing its efficiency.
Remove an unneeded index in historian backup db.
Increase backup db page count.

Scalability Planning

Goals

- Determine the limits of the number of devices that can be interacted with via a single Volttron platform.
- Determine how scaling out affects the rate at which devices are scraped. i.e. How long from the first device scrape to the last?
- Determine the effects of socket throttling in the master driver on the performance of Modbus device scraping.
- Measure total memory consumption of the Master Driver Agent at scale.
- Measure how well the base history agent and one or more of the concrete agents handle a large amount of data.
- Determine the volume of messages that can be achieved on the pubsub before the platform starts rejecting them.

Test framework

Test Devices

Simple, command line configured virtual devices to test against in both Modbus and BACnet flavors. Devices should create 10 points to read that generate either random or easily predictable (but not necessarily constant) data. Process should be completely self contained.

Test devices will be run on remote hosts from the Volttron test deployment.

Launcher Script

- The script will be configurable as to the number and type of devices to launch.
- The script will be configurable as to the hosts to launch virtual devices on.
- The script (probably a fabric script) will push out code for and launch one or more test devices on one or more machines for the platform to scrape.
- The script will generate all of the master driver configuration files to launch the master driver.
- The script may launch the master driver.
- The script may launch any other agents used to measure performance.

Shutdown Script

- The script (probably the same fabric script run with different options) will shutdown all virtual drivers on the network.
- The script may shutdown the master driver.
- The script may shutdown any related agents.

Performance Metrics Agent

This agent will track the publishes by the different drivers and generate data in some form to indicate:

- Total time for all devices to be scraped
- Any devices that were not successfully scraped.
- Performance of the message bus.

Additional Benefits

Most parts of a test bed run should be configurable. If a user wanted to verify that the Master Driver worked, for instance, they could run the test bed with only a few virtual device to confirm that the platform is working correctly.

Running a simple test

You will need 2 open terminals to run this test. (3 if you want to run the platform in it's own terminal)
Checkout the feature/scalability branch.

Start the platform.

Go to the volttron/scripts/scalability-testing directory in two different terminals. (Both with the environment activated)

In one terminal run:

```
python config_builder.py --count=1500 --scalability-test --scalability-test-
↪iterations=6 fake fake18.csv localhost
```

Change the path to fake.csv as needed.

(Optional) After it finishes run:

```
./launch_fake_historian.sh
```

to start the null historian.

In a separate terminal run:

```
./launch_scalability_drivers.sh
```

to start the scalability test.

This will emulate the scraping of 1500 devices with 18 points each 6 times, log the timing, and quit.

Redirecting the driver log output to a file can help improve performance. Testing should be done with and without the null historian.

Currently only the depth first all is published by drivers in this branch. Uncomment the other publishes in driver.py to test out full publishing. fake.csv has 18 points.

Optionally you can run two listener agents from the volttron/scripts directory in two more terminals with the command:

```
./launch_listener.sh
```

and rerun the test to see the how it changes the performance.

Real Driver Benchmarking

Scalability testing using actual MODBUS or BACnet drivers can be done using the virtual device applications in the scripts/scalability-testing/virtual-drivers/ directory. The configuration of the master driver and launching of these virtual devices on a target machine can be done automatically with fabric.

Setup

This requires two computers to run: One for the VOLTTRON platform to run the tests on ("the platform") and a target machine to host the virtual devices ("the target").

Target setup

The target machine must have the VOLTTRON source with the feature/scalability branch checked out and bootstrapped. Make a note of the directory of the VOLTTRON code.

Platform setup

With the VOLTTRON environment activated install fabric.

```
pip install fabric
```

Edit the file scripts/scalability-testing/test_settings.py as needed.

- virtual_device_host (string) - Login name and IP address of the target machine. This is used to remotely start and stop virtual devices via ssh. "volttron@10.0.0.1"

- device_types - map of driver types to tuple of the device count and registry config to use for the virtual devices. Valid device types are "bacnet" and "modbus".
- volttron_install - location of volttron code on the target.

To configure the driver on the platform and launch the virtual devices on the target run

```
fab deploy_virtual_devices
```

When prompted enter the password for the target machine. Upon completion virtual devices will be running on the target and configuration files written for the master driver.

Launch Test

If your test includes virtual BACnet devices be sure to configure and launch the BACnet Proxy before launching the scalability driver test.

(Optional)

```
./launch_fake_historian.sh
```

to start the null historian.

In a separate terminal run:

```
./launch_scalability_drivers.sh
```

to start the scalability test.

To stop the virtual devices run

```
fab stop_virtual_devices
```

and enter the user password when prompted.

1.47.2 Version History

VOLTTRON 1.0 – 1.2

- Agent execution platform
- Message bus
- Modbus and BACnet drivers
- Historian
- Data logger
- Device scheduling
- Device actuation
- Multi-node communication
- Weather service

VOLTTRON 2.0

- Advanced Security Features
- Guaranteed resource allocation to agents using execution contracts
- Signing and verification of agent packaging
- Agent mobility
- Admin can send agents to another platform
- Agent can request to move
- Enhanced command framework

VOLTTRON 3.0

- Modularize Data Historian
- Modularize Device Drivers
- Secure and accountable communication using the VIP
- Web Console for Monitoring and Administering VOLTTRON Deployments

VOLTTRON 4.0

- Documentation moved to ReadTheDocs
- VOLTTRON Configuration Wizard
- Configuration store to dynamically configure agents
- Aggregator agent for aggregating topics
- More reliable remote install mechanism
- UI for device configuration
- Automatic registration of VOLTTRON instances with management agent

VOLTTRON 5.0

- Tagging service for attaching metadata to topics for simpler retrieval
- Message bus performance improvement
- Multi-platform publish/subscribe for simpler coordination across platforms
- Drivers contributed back for SEP 2.0 and ChargePoint EV

VOLTTRON 6.0

- Maintained backward compatibility with communication between zmq and rmq deployments.
- Added DarkSky Weather Agent
- Web Based Additions
- Added CSR support for multiplatform communication

- Added SSL support to the platform for secure communication
- Backported SSL support to zmq based deployments.
- Upgraded VC to use the platform login.
- Added docker support to the test environment for easier Rabbitmq testing.
- Updated volttron-config (vcfg) to support both RabbitMQ and ZMQ including https based instances.
- Added test support for RabbitMQ installations of all core agents.
- Added multiplatform (zmq and rmq based platform) testing.
- Integrated RabbitMQ documentation into the core documentation.

VOLTTRON 7.0rc1

Python3 Upgrade

- Update libraries to appropriate and compatible versions
- String handling efficiency
- Encode/Decode of strings has been simplified and centralized
- Added additional test cases for frame serialization in ZMQ
- Syntax updates such difference in handling exceptions, dictionaries, sorting lists, pytest markers etc.
- Made bootstrap process simpler
- Resolved gevent monkey patch issues when using third party libraries

RabbitMQ Message Bus

- **Client code for integrating non-VOLTTRON applications with the message bus** available at: https://github.com/VOLTTRON/external-clients-for-rabbitmq
- Includes support for MQTT, non-VOLTTRON Python, and Java-based RabbitMQ clients

Config store secured

- Agents can prevent other agents from modifying their configuration store entry

Known Issues which will be dealt with for the final release:

- Python 3.7 has conflicts with some libraries such as gevent
- The VOLTTRON Central agent is not fully integrated into Python3
- CFFI library has conflicts on the Raspian OS which interferes with bootstrapping

VOLTTRON 7.0 Full Release

This is a full release of the 7.0 version of VOLTTRON which has been refactored to work with Python3. This release incorporates community feedback from the release candidate as well as new contributions and features. Major new features and highlights since the release candidate include:

- Added secure agent user feature which allows agents to be launched as a user separate from the platform. This protects the platform against malformed or malicious agents accessing platform level files
- Added a driver to interface with the Ecobee smart thermostat and make data available to agents on the platform
- Updated VOLTTRON Central UI to work with Python3
- Added web support to authenticate remote VOLTTRON ZMQ message bus-based connections
- Updated ZMQ-based multiplatform RPC with Python 3
- To reduce installation size and complexity, fewer services are installed by default
- MasterDriver dependencies are not installed by default during bootstrap. To use MasterDriver, please use the following command:

  ```
  python3 bootstrap.py --driver
  ```

- Web dependencies are not installed by default during bootstrap. To use the MasterWeb service, please use the following command:

  ```
  python3 bootstrap.py --web
  ```

- Added initial version of test cases for *volttron-cfg* (*vcfg*) utility
- On all arm-based systems, *libffi* is now a required dependency, this is reflected in the installation instructions
- On arm-based systems, Raspbian >= 10 or Ubuntu >= 18.04 is required
- Updated examples and several contributed features to Python 3
- Inclusion of docker in test handling for databases
- A new */gs* endpoint to access platform services without using Volttron Central through Json-RPC
- A new SCPAgent to transfer files between two remote systems

Known Issues

- Continued documentation updates to ensure correctness
- Rainforest Eagle driver is not yet upgraded to Python3
- A bug in the Modbus TK library prevents creating connections from 2 different masters to a single slave.
- BACnet Proxy Agent and BACnet auto configuration scripts require the version of BACPypes installed in the virtual environment of VOLTTRON to be version 0.16.7. We have pinned it to version 0.16.7 since it does not work properly in later versions of BACPypes.
- VOLTTRON 7.0 code base is not fully tested in Ubuntu 20.04 LTS so issues with this combination have not been addressed

1.47.3 Upgrading Existing Deployments

It is often recommended that users upgrade to the latest stable release of VOLTTRON for their deployments. Major releases include helpful new features, bug fixes, and other improvements. Please see the guides below for upgrading your existing deployment to the latest version.

VOLTTRON 7

VOLTTRON 7 includes a migration from Python 2.7 to Python 3.6, as well as security features, new agents, and more.

From 6.x

From version 6.x to 7.x important changes have been made to the virtual environment as well as *VOLTTRON_HOME*. Take the following steps to upgrade:

Note: The following instructions are for debian based Linux distributions (including Ubuntu and Linux Mint). For Red Hat, Arch or other distributions, please use the corresponding package manager and commands.

1. Install the VOLTTRON dependencies using the following command:

   ```
   sudo apt install python3-dev python3-venv libffi-dev
   ```

   Note: This assumes you have existing 6.x dependencies installed. If you're unsure, refer to the *platform installation* instructions.

2. Remove your existing virtual environment and run the bootstrap process.

   To remove the virtual environment, change directory to the VOLTTRON project root and run the *rm* command with the `-r` option.

   ```
   cd $VOLTTRON_ROOT/
   rm -r env
   ```

   Now you can use the included *bootstrap.py* script to set up the new virtual environment. For information on how to install dependencies for VOLTTRON integrations, run the script with the `--help` option.

   ```
   python3 bootstrap.py <options>
   ```

   Note: Because the new environment uses a different version of Python, using the `--force` option with bootstrap will throw errors. Please follow the above instructions when upgrading.

3. Make necessary VOLTTRON_HOME changes

   Warning: It is possible that some existing agents may continue to operate after the platform upgrade, however this is not true for most agents, and it is recommended to reinstall the agent to ensure the agent wheel is compatible and that there are no side-effects.

   A. Reinstall Agents

It is recommended to reinstall all agents that exist on the platform to ensure the agent wheel is compatible with Python3 VOLTTRON. In many cases, the configurations for version 7.x are backwards compatible with 6.x, requiring no additional changes from the user. For information on individual agent configs, please read through that agent's documentation.

B. Modify Agent Directories

Note: Modifying the agent directories is only necessary if not reinstalling agents.

To satisfy the security requirements of the secure agents feature included with VOLTTRON 7, changes have been made to the agent directory structure.

1. Keystore.json

The agent keystore file has been moved from the agent's *agent-data* directory to the agent's *dist-info* directory. To move the file, change directory to the agents install directory and use the *mv* command.

```
cd $VOLTTRON_HOME/agents/<agent uuid>/<agent dir>
mv <agent>agent.agent-data/keystore.json <agent>agent.dist-info/
```

2. Historian Database

Historians with a local database file have had their default location change do the *data* directory inside of the agent's install directory. It is recommended to relocate the file from $VOLTTRON_HOME/data to the agent's data directory. Alternatively, a path can be used if the user the agent is run as (the VOLTTRON user for deployments not using the secure agents feature) has read-write permissions for the file.

```
mv $VOLTTRON_HOME/data/historian.sqlite $VOLTTRON_HOME/agents/<agent uuid>
↪/<agent>/data
```

Warning: If not specifying a path to the database, the database will be created in the agent's data directory. This is important if removing or uninstalling the historian as the database file will be removed when the agent dir is cleaned up. Copy the database file to a temporary directory, reinstall the agent, and move the database file back to the agent's data directory

4. Forward Historian

For deployments which are passing data from 6.x VOLTTRON to the latest 7.x release, some users will experience timeout issues with the Forward Historian. By updating the 6.x deployment to the latest from the releases/6.x branch, and restarting the platform and forwarder, this issue can be resolved.

```
. env/bin/activate
./stop-volttron
git pull
git checkout releases/6.x
./start-volttron
vctl start <forward id or tag>
```

CHAPTER 2

Indices and tables

- genindex
- modindex