U.S. DEPARTMENT OF ENERGY

PNNL-24907

Prepared for the U.S. Department of Energy

under Contract DE-AC05-76RL01830

VOLTTRON 3.0: User Guide

RG Lutes
S Katipamula
BA Akyol
JN Haack
KE Monson
BJ Carpenter

November 2015

Pacific Northwest
NATIONAL LABORATORY

Proudly Operated by Battelle Since 1965

PNNL-24907

VOLTTRON 3.0: User Guide

RG Lutes
JN Haack
S Katipamula
KE Monson
BA Akyol
BJ Carpenter

November 2015

Prepared for
U.S. Department of Energy
under Contract DE-AC05-76RL01830

Pacific Northwest National Laboratory
Richland, Washington 99352

Summary

The Department of Energy's (DOE's) Building Technologies Office (BTO) is supporting the development of the concept of "transactional network" that supports energy, operational, and financial transactions between building systems (e.g., rooftop units -- RTUs), between building systems and the electric power grid using applications, or 'agents' that reside either on the equipment, on local building controllers or in the Cloud.

As part of this Transactional Network initiative, BTO has funded Pacific Northwest National Laboratory (PNNL) to develop an open-source, open-architecture platform that enables a variety of site/equipment specific applications to transact in a cost-effective and scalable way. The goal of this initiative is to lower the cost of entry for both existing and/or new service providers because the data transport or information exchange typically required for operational and energy-related products and services will be ubiquitous and interoperable.

The Transactional Network Platform (TNP) consists of VOLTTRON™ agent execution software, agents that perform critical services that enable and enhance the functionality of VOLTTRON, and a number of agents that perform a specific function (fault detection, demand response, weather service, logging service, etc.). The platform is intended to support energy, operational, and financial transactions between networked entities (equipment, organizations, buildings, grid, etc.).

This document is a user guide for the deployment of the transactional network platform and agent/application development within VOLTTRON. The intent of this user guide is to provide a description of the functionality of the transactional network platform. This document describes how to deploy the platform, including installation, use, guidance, and limitations. It also describes how additional features can be added to enhance its current functionality.

Table of Contents

Figures

1 Introduction

Pacific Northwest National Laboratory (PNNL), with funding from the Department of Energy's (DOE's) Building Technologies Office (BTO), designed, prototyped and tested a transactional network platform. The Transactional Network Platform (TNP) consists of VOLTTRON™ agent execution software, agents that perform critical services that enable and enhance the functionality of VOLTTRON, and a number of agents that perform a specific function (fault detection, demand response, etc.). The platform is intended to support energy, operational, and financial transactions between networked entities (equipment, organizations, buildings, grid, etc.) and enhance the control infrastructure of existing buildings through the use of open source device communication protocols and integrated analytics.

To encourage development and growth of the TNP, all the software related to VOLTTRON, platform services, and the agents within VOLTTRON are open source and employ a BSD (Berkeley Software Distribution) style license, allowing the free distribution and development of the TNP.

Enhancements to the platform such as agent mobility, signing and verification of agents, and resource management are available under a different license. Please, see Section 5 for a discussion of these features.

This guide is intended to give detailed instructions for the initial deployment of the TNP and VOLTTRON, how to launch of agents (applications) on the platform, and help developing new agents/applications for use on the TNP. This guide will also show how to communicate with devices (e.g., controllers, thermostats, etc.,) that utilize the Modbus or BACnet communication protocols.

1.1 Background

Today's building systems do not participate significantly in the energy market or provide services to power system operators. However, new smart grid technologies are creating a significant potential for building systems to participate in energy markets by providing ancillary services to power system operators. Communication networks and advanced control systems are a necessary enabler of this new potential. The transactional network platform will demonstrate the utilization of building systems (e.g., RTUs) for providing energy services to utilities using autonomous controllers. This platform will also allow for development of the next-generation control strategies and validation of the strategies by:

- Quantitative analysis of energy management opportunities within buildings
- Design, prototype, and analysis of the advanced controller strategies for building systems
- Design and analysis of communication network within building and external interfaces to utility communication networks
- Economics of control strategies.

The rate and granularity of the control for the building systems determines the types of utility services that can be provided.

1.2 Transactional Network Platform Overview

In the TNP, VOLTTRON connects devices (RTUs, AHUs, chillers, boilers, device controllers, meters, etc.) to applications implemented in the platform and in the Cloud, a data historian, and signals from the power grid. VOLTTRON is an agent execution platform providing services to its agents that allow them to easily communicate with physical devices and other resources. VOLTTRON also provides helper classes (software methods) to ease development and deployment of agents, and their deployment into the TNP.

Figure 1 shows the various components of the transactional network platform. The driver communicates to the building system controllers using Modbus or BACnet. It periodically reads data off the controller and publishes data to the message bus via a topic for each device; it also provides a means to send control commands from agents to controllers. The Actuator/Scheduler agent allows other applications on the platform to schedule times to interact with devices. This Scheduler agent ensures that multiple agents are not actively controlling a device and allows the user to set the relative priority of each application.

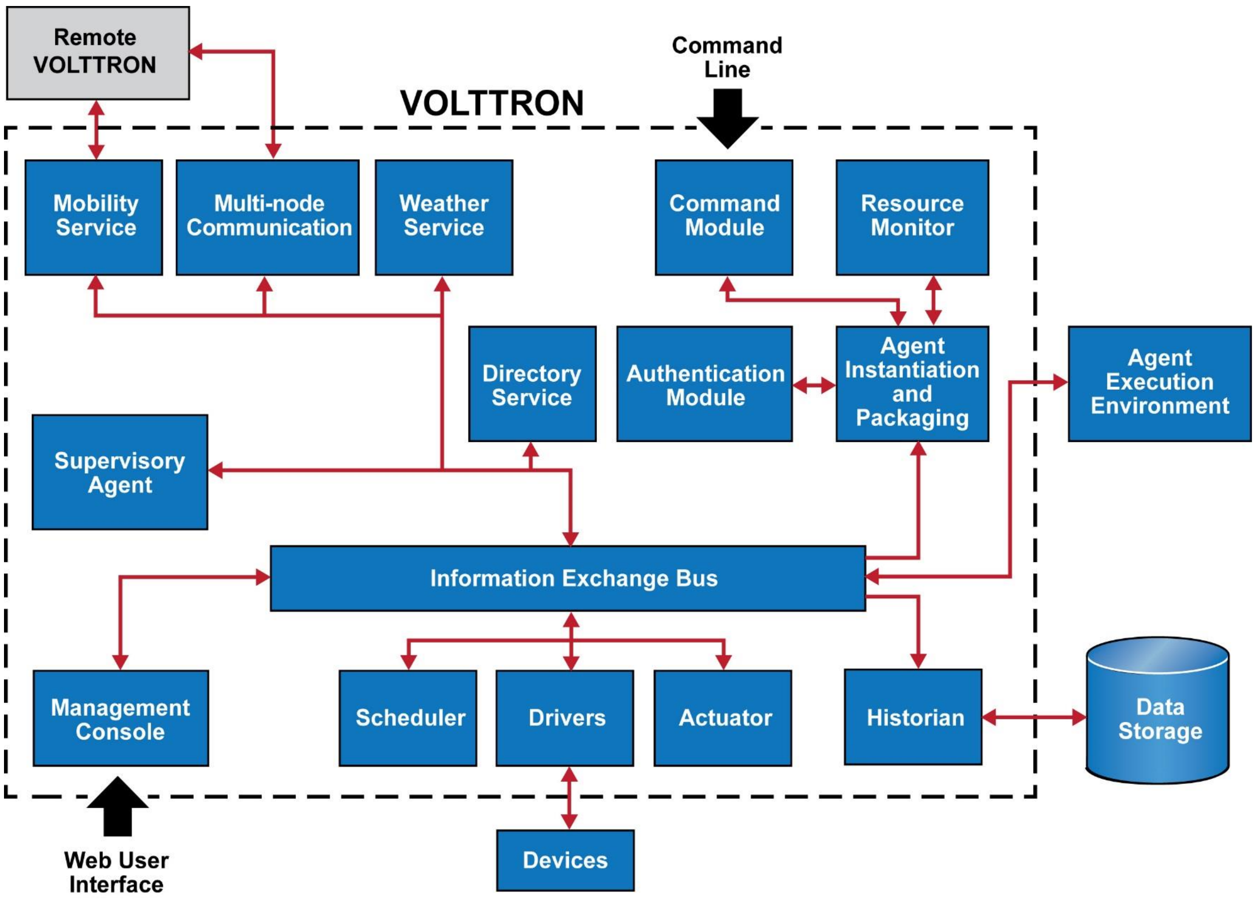


Figure 1: Illustration of the Various Components of the Transactional Network

Historian agents collect data from the message bus for storage and later retrieval, isolating agents from the specifics of the storage solution used. This allows multiple historians be used. For instance, to store data to a local database as well as to an external web service.

Agents and platform services shown in Figure 1 communicate with each other via the message bus using the publish/subscribe paradigm over a variety of topics. For example, the weather agent would publish weather information to a “weather” topic that interested agents would subscribe to. The platform itself publishes platform related messages to the “platform” topic (such as “shutdown”). Topics are hierarchical following the format “topic/subtopic/subtopic/…/…” allowing agents to be as general or as specific as desired with their subscriptions. For example, agents could subscribe to “weather/all” and get all weather data for a location or “weather/temperature” for only temperature data.

1.3 VOLTTRON Overview

VOLTTRON is an open-source, open-architecture platform that enables a variety of site/equipment specific applications to be applied in a cost-effective and scalable way. Such an open-source platform will lower the cost of entry for both existing and new service providers because the data transport or information exchange typically required for operational and energy-related products and services would be ubiquitous and interoperable.

1.3.1 VOLTTRON

VOLTTRON serves as an integrating platform for the components of the transactional network. It provides an environment for agent execution and serves as a single point of contact for interfacing with devices (RTUs, building systems, meters, etc.), external resources, and platform services such as data archival and retrieval. VOLTTRON provides a collection of utility and helper classes, which simplifies agent development. VOLTTRON connects devices to applications implemented in the platform and in the Cloud, a data historian, and signals from the power grid. VOLTTRON incorporates a number of open source projects to build a flexible and powerful platform. The following is a summary of the various open source software that VOLTTRON uses:

- ∅MQ: The VOLTTRON message bus, which allows agents and services to exchange data, uses Zero MQ[1]. This free software is used by National Aeronautics and Space Administration (NASA), Cisco, etc. to provide scalable, reliable, and fast communication. The VOLTTRON team are active members of this open-source software community, both reporting and developing code to fix software bugs.
- PyModbus: The VOLTTRON Modbus[2] driver builds on PyModbus[3], which enables Python code to easily interact with Modbus devices.
- BACPypes[4]: The VOLTTRON BACnet[5] driver utilizes BACPypes to interact with devices supporting the BACnet protocol
- SQLite: The VOLTTRON BaseHistorian uses SQLite[6] for its local data cache. SQLite can also be used as a main data storage solution. SQLite provides a simple, reliable, and easy to use solution for local data storage.
- MySQL: VOLTTRON uses MySQL[7] as an option for a data historian, a database to store device data (historical trends) and agent analytic results.
- sMAP: VOLTTRON uses sMAP[8] as an option for a data historian, a database to store device data (historical trends) and agent analytic results.
- Other open-source Python modules being used are:
  - 'avro', 'configobj', 'gevent', 'flexible-jsonrpc', 'numpy', 'posix-clock', ‘pyopenssl', 'python-dateutil', 'requests', 'setuptools', 'simplejson', 'zope.interface', pandas, tornado, ply

[1] http://zeromq.org/
[2] http://www.modbus.org/
[3] http://code.google.com/p/pymodbus/
[4] http://bacpypes.sourceforge.net/
[5] http://www.bacnet.org/
[6] https://www.sqlite.org/
[7] https://www.mysql.com/
[8] http://www.cs.berkeley.edu/~stevedh/smap2/index.html

1.3.2 VOLTTRON Services

VOLTTRON's services utilize the above mentioned open source software in conjunction with other applications developed by collaborators; these services/applications include:

- **Actuator Agent**: This platform service is deployed in the form of an agent running on VOLTTRON. The Actuator agent manages the control of external devices by agents within VOLTTRON.
    - **Device control**: The Actuator agent will accept commands from other agents and issue the commands to the specified device. Currently Modbus and BACnet compatible device communication is supported.
    - **Device access scheduling**: This service allows the scheduling of agents' access to devices to prevent multiple agents from controlling the same device at the same time.
- **Drivers:** VOLTTRON driver agents communicate with devices being controlled by the platform. They isolate device-specific protocols from the rest of the platform by publishing device data to the message bus and taking commands from the message bus.
- **Historian**: Enables the storage of device data obtained by the Drivers and application analysis results in a database (currently SQLlite, MySQL, and sMAP databases are supported).
- **Management Interface:** Web based user interface that allows the administration of VOLTTRON nodes (and the agent/applications) running on the VOLTTRON nodes on one or more networks.
- **Message Bus**: All agents and services can publish and subscribe to topics on the message bus. This provides a single and uniform interface that abstracts the details of devices and agents from each other. At the most basic level, agents and components running in the platform produce and consume messages and/or events. The details of how agents produce events and how they process received events are left up to the agents.
- **Multi-Node Communication:** The MultiBuilding agent allows agents to publish and subscribe to the message bus of a remote VOLTTRON platform. This communication can be encrypted using 0MQ Curve. Note, this functionality can now be accessed directly through VIP
- **VOLTTRON Interconnect Protocol (VIP):** VIP is a protocol designed to increase the security of communications within and between VOLTTRON platforms. It allows for attribution of messages and restriction of access. It also makes it easier to address messages to agents on other platforms.
- **Weather Information**: This platform service is deployed in the form of an agent running on VOLTTRON. This agent periodically retrieves data from the Weather Underground site. It then reformats the data and publishes it to the platform on a weather topic accessible to other agents.
- VOLTTRON licensed enhancements (discussed more in Section 5).
    - **Agent Signing and Verification:** Agent code and configuration information is signed by several entities to ensure that it has not been tampered with in transit or while on the server.

- **Resource Management:** Agents present an execution contract with a resource requirements estimate. The platform only allows agents to run if it can support their requirements
- **Agent Mobility:** Agent can be sent to other platforms via an administrator command or they can request the move themselves. The receiving platform performs verification of the agent package and resource requirements before allowing it to execute.

Agents deployed on VOLTTRON can perform one or more roles, which can be broadly classified into the following groups:

- **Platform Agents:** These agents provide services to other agents running on the platform such as weather information, device scheduling, etc.

- **Proxy Agents**: These agents act as a bridge to remote applications that need access to the messages and data on the platform. A Proxy agent subscribes to topics of interest and forwards messages to the remote (or Cloud) application. These Cloud applications can then publish data to the platform via the Proxy agent.

- **Control Agents**: Using data from buildings and other agents, these agents make decisions and interact with devices and other resources to achieve a goal

- **Passive Agent**: These agents subscribe to certain data from the building systems and perform certain actions to create knowledge (faulty operation). The information and knowledge that these agents create is posted to the Historian or in a local file.

1.3.3 Conventions Used In Guide

The following conventions will be used in this guide:

- Terminal commands will be shown in a grey dotted box and will use Consolas font. For example:

  ```
  $ ls -l
  ```

  Note: A leading `#` symbol indicates a terminal command is run as root (or using sudo) while a leading `$` indicates the terminal command should be executed as a non-privileged user.

- File paths in the body of a paragraph will be enclosed in parenthesis and will use Consolas 10 point regular font. For example:

  The file can be located at (`/home/user/myfile.log`).

  The "~" symbol indicates the current user's home directory (`/home/<user>`) and (`<project directory>`) is the file path to the base (root) project directory.

- The contents of configuration files or other text files (including Python .py files) will be shown in a grey box with no border and will use Consolas font. For example:

  ```
  {
      'agentID': 'my-agent'
      'taskID': 'some task'
  }
  ```

- A file or script name will be Times New Roman 11 point italic font. For example:

  The file *example.py* is located at (`/home/user/example.py`).

- A user supplied value will be Times New Roman 11 point bold font. For example:

  parameter1 - this is a user supplied (configurable) parameter.

- A class or method (function) name will be in Consolas 10 point bold font. For example:

  `Myclass` inherits from **`Baseclass`**

- Python Code blocks will be Consolas 10 point font and are color coded using the default schema from the Eclipse Integrated Development Environment plugin PyDev.

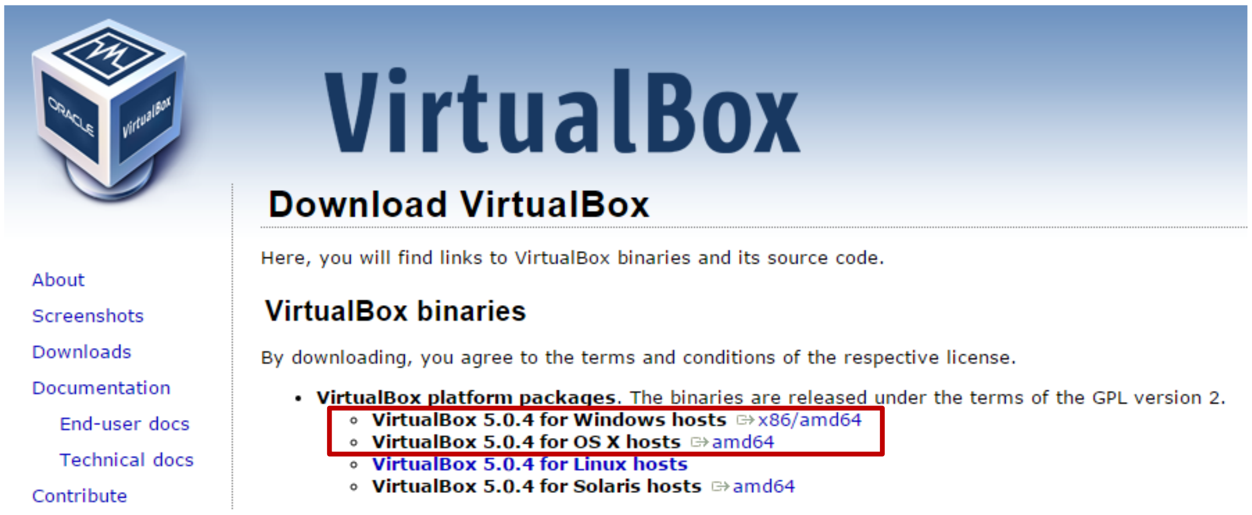
  The default PyDev color code is as follows:

  - Code **-** **`color`**
  - Decorator **-** `color`
  - Numbers **-** **`color`**
  - Keywords **-** **`color`**

- Strings - `color`
- Comments - `color`
- Mathematical operators - `color`

2 Deployment of VOLTTRON

VOLTTRON has been developed for deployment on Linux operating systems. To use VOLTTRON on a Mac or Windows system, VOLTTRON must be deployed on a virtual machine (VM). A VM is a software implementation of a machine (i.e., a computer) that executes programs like a physical machine. A system VM provides a complete system platform, which supports the execution of a complete operating system (OS). These usually emulate an existing architecture, and are built with the purpose of providing a platform to run programs where the real hardware is not available for use. This document will describe the steps necessary to install VOLTTRON on a Windows system using Oracle VirtualBox software (Figure 2).



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.

- **VirtualBox platform packages**. The binaries are released under the terms of the GPL version 2.
  - **VirtualBox 5.0.4 for Windows hosts** x86/amd64
  - **VirtualBox 5.0.4 for OS X hosts** amd64
  - **VirtualBox 5.0.4 for Linux hosts**
  - **VirtualBox 5.0.4 for Solaris hosts** amd64

Figure 2: VirtualBox Download Page

2.1 Installing Linux Virtual Machine

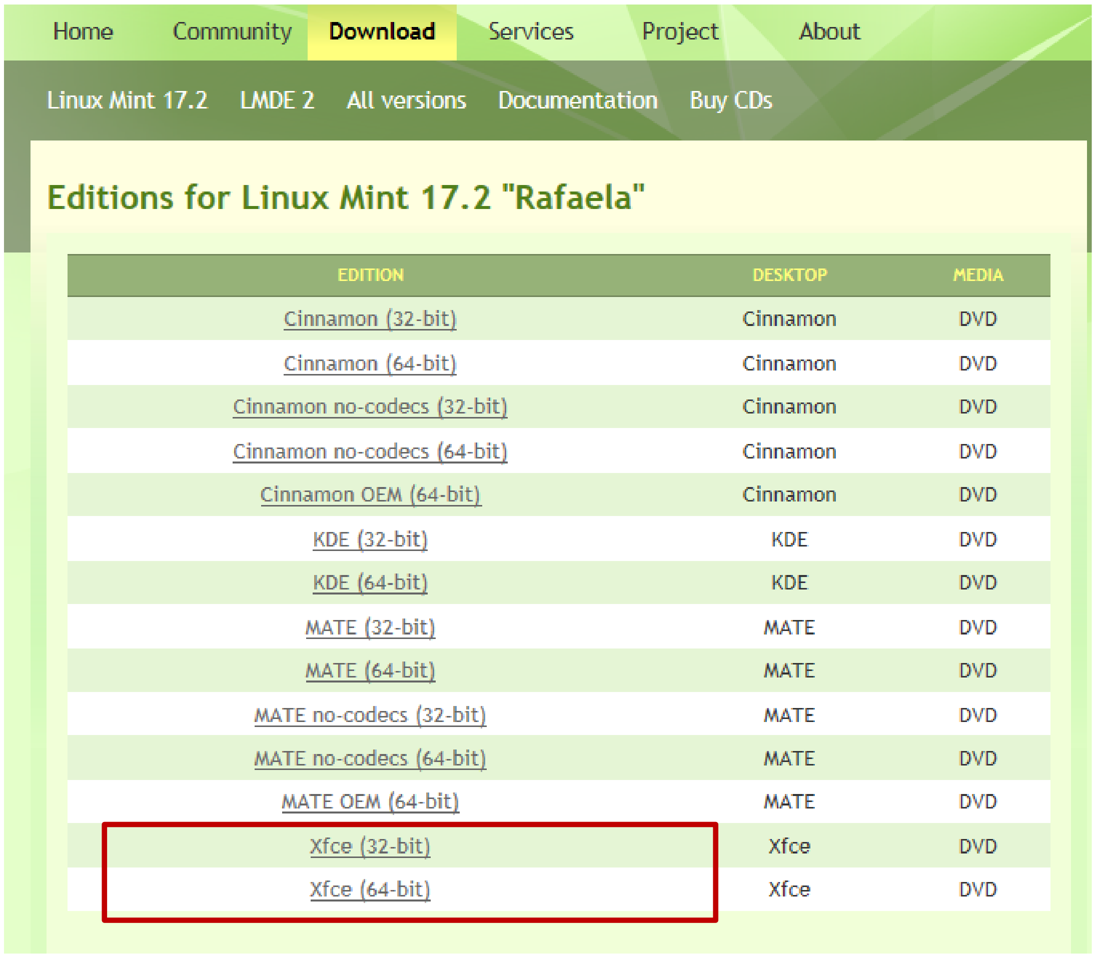
VirtualBox is free and can be downloaded from https://www.virtualbox.org/wiki/Downloads. Figure 2 shows the VirtualBox download page. The Windows and Mac host OS is shown boxed in red in the figure.

- To install on Windows choose: VirtualBox for Windows hosts x86/amd64

- To install on Mac choose: VirtualBox for OS X hosts x86/amd64

The latest version of VirtualBox, when this guide was constructed, was VirtualBox 5.0.4. VOLTTRON should be compatible with future releases of VirtualBox. After the installation file is downloaded, run and install the VirtualBox software. It will also be necessary to download a Linux operating system image for use on the VM. Ubuntu 14.04 LTS or Linux Mint 17 or later is the recommended Linux operating system for use with VOLTTRON. Any distribution of Linux should work with VOLTTRON (Debian, Ubuntu, Fedora, Arch Linux, etc.) but this document will describe the development of agents within VOLTTRON where Linux Mint 17.2 with the Xfce desktop is used. The other desktops associated with Linux Mint are compatible with VOLTTRON and should provide similar functionality to the Xfce desktop (Figure 3).

Linux Mint can be downloaded from the following URL http://www.linuxmint.com/release.php?id=25. Set up of the platform in Ubuntu is identical to the setup in Linux Mint[9] except for changes in the appearance of the desktop. A 32-bit version of Linux should be used when running VOLTTRON on a system with limited hardware (less than 2 GB of RAM).



Editions for Linux Mint 17.2 "Rafaela"

EDITION	DESKTOP	MEDIA
Cinnamon (32-bit)	Cinnamon	DVD
Cinnamon (64-bit)	Cinnamon	DVD
Cinnamon no-codecs (32-bit)	Cinnamon	DVD
Cinnamon no-codecs (64-bit)	Cinnamon	DVD
Cinnamon OEM (64-bit)	Cinnamon	DVD
KDE (32-bit)	KDE	DVD
KDE (64-bit)	KDE	DVD
MATE (32-bit)	MATE	DVD
MATE (64-bit)	MATE	DVD
MATE no-codecs (32-bit)	MATE	DVD
MATE no-codecs (64-bit)	MATE	DVD
MATE OEM (64-bit)	MATE	DVD
Xfce (32-bit)	Xfce	DVD
Xfce (64-bit)	Xfce	DVD

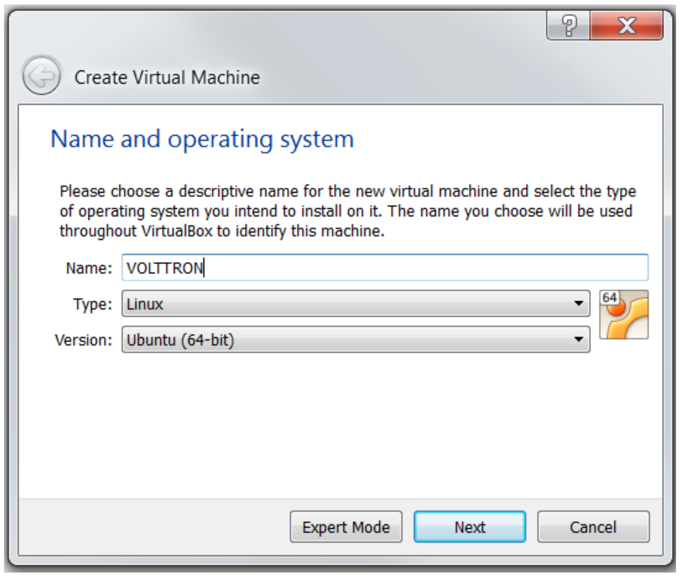
Figure 3: Linux Mint Download Page

2.2 Running and Configuring Virtual Machine

After the VirtualBox software is installed and the Linux Mint image has been downloaded, the virtual machine can be run and configured. The following steps describe how to configure the VM for deployment of VOLTTRON:

1. Start VirtualBox and click New icon in the top left corner of Oracle VM VirtualBox Manager window.
2. A selection box will appear; configure the selection as shown in Figure 4. Choose Next.

[9] Note that Linux Mint version could be different from the shown here. Also, on the download screen, you could pick any site, but preferably the site that is close to you.


Create Virtual Machine

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.

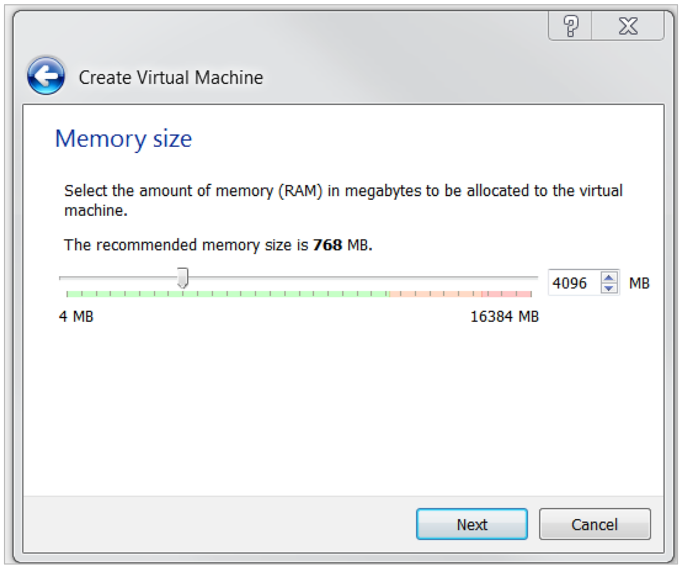
Name: VOLTTRON

Type: Linux

Version: Ubuntu (64-bit)

Figure 4: Creating a Virtual Machine

3. Choose the amount of memory to allocate to the VM, as shown in Figure 5. Note that this memory will be unavailable to the host while running the VM (i.e., a computer with 4 GB of memory, could probably spare 1 GB for the VM). Choose Next.


Create Virtual Machine

Memory size

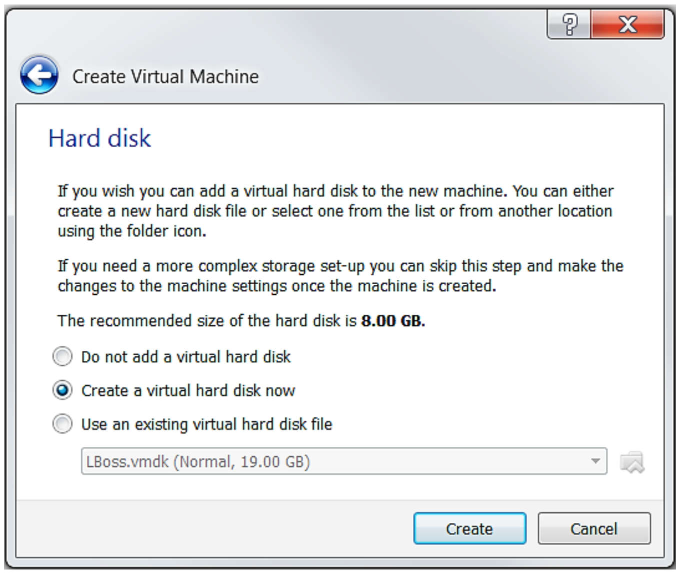
Select the amount of memory (RAM) in megabytes to be allocated to the virtual machine.

The recommended memory size is **768** MB.

4096 MB

Figure 5: Selecting Memory Size

4. Create hard drive for VM. Choose Create, as shown in Figure 6.



Create Virtual Machine

Hard disk

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.

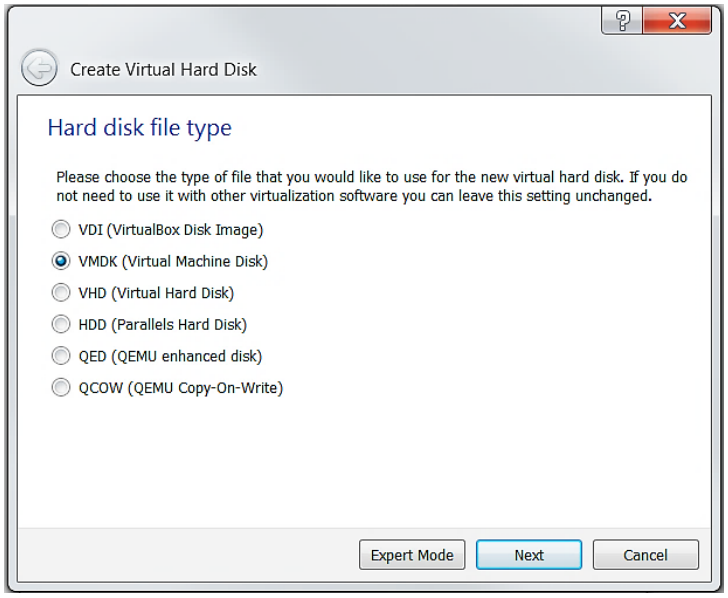
The recommended size of the hard disk is **8.00 GB.**

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

LBoss.vmdk (Normal, 19.00 GB)

Figure 6: Selecting Storage Size

5. Choose disk type. As shown in the Figure 7, select VMDK and then select Next.



Create Virtual Hard Disk

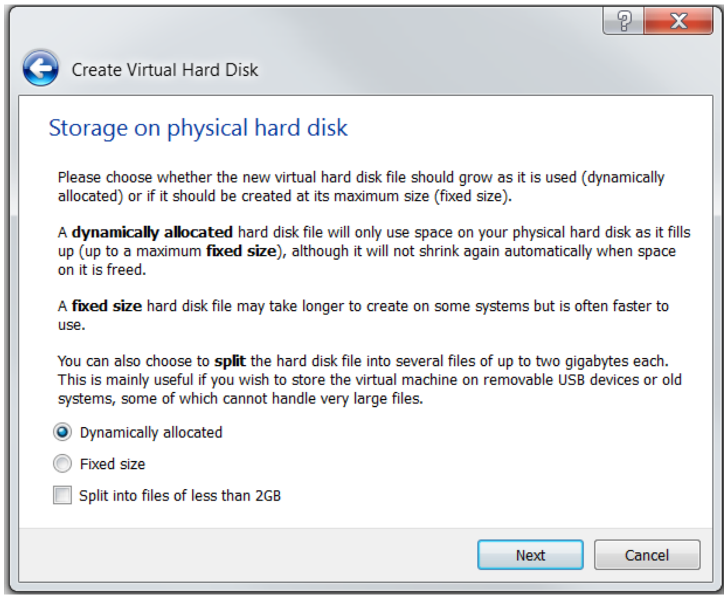
Hard disk file type

Please choose the type of file that you would like to use for the new virtual hard disk. If you do not need to use it with other virtualization software you can leave this setting unchanged.

- [ ] VDI (VirtualBox Disk Image)
- [✓ VMDK (Virtual Machine Disk)]
- [ ] VHD (Virtual Hard Disk)
- [ ] HDD (Parallels Hard Disk)
- [ ] QED (QEMU enhanced disk)
- [ ] QCOW (QEMU Copy-On-Write)

Figure 7: Creating Virtual Hard Drive

6. Choose Dynamically Allocated for the VM hard drive (Figure 8). This will allow the VM hard drive to only take storage space as needed, up to the size limit chosen in the previous step. Choose Continue.



Create Virtual Hard Disk

Storage on physical hard disk

Please choose whether the new virtual hard disk file should grow as it is used (dynamically allocated) or if it should be created at its maximum size (fixed size).

A **dynamically allocated** hard disk file will only use space on your physical hard disk as it fills up (up to a maximum **fixed size**), although it will not shrink again automatically when space on it is freed.

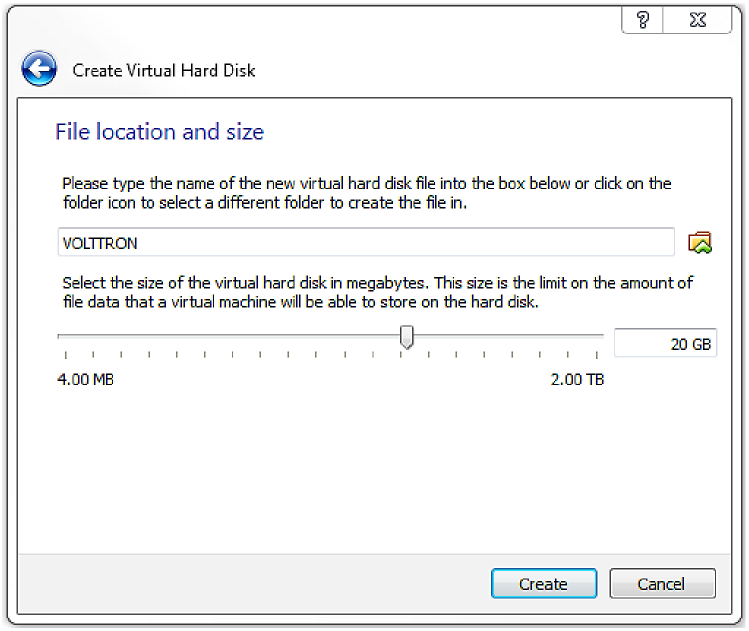
A **fixed size** hard disk file may take longer to create on some systems but is often faster to use.

You can also choose to **split** the hard disk file into several files of up to two gigabytes each. This is mainly useful if you wish to store the virtual machine on removable USB devices or old systems, some of which cannot handle very large files.

● Dynamically allocated
○ Fixed size
☐ Split into files of less than 2GB

Figure 8: Selection of Type of Hard Drive

7. Choose the file size for the VM virtual hard drive. Keep in mind that Linux Mint 17 is close to 4 GB just for the operating system (Figure 9). Choose Create.



Create Virtual Hard Disk

File location and size

Please type the name of the new virtual hard disk file into the box below or click on the folder icon to select a different folder to create the file in.

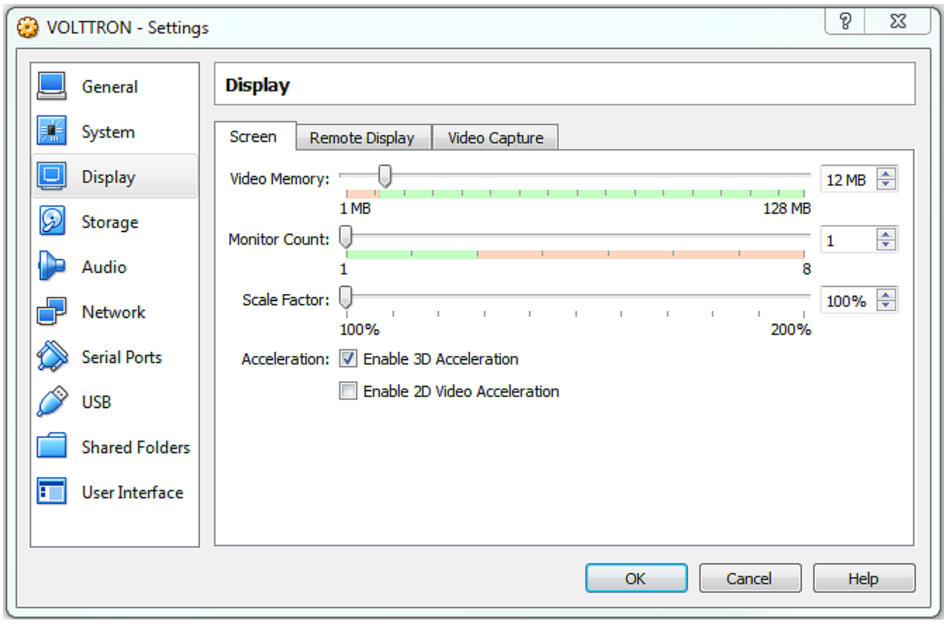
VOLTTRON

Select the size of the virtual hard disk in megabytes. This size is the limit on the amount of file data that a virtual machine will be able to store on the hard disk.

20 GB

Figure 9: Creating Virtual Hard Drive (continued)

8. 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. In the Display menu check Enable 3D Acceleration (Figure 10).

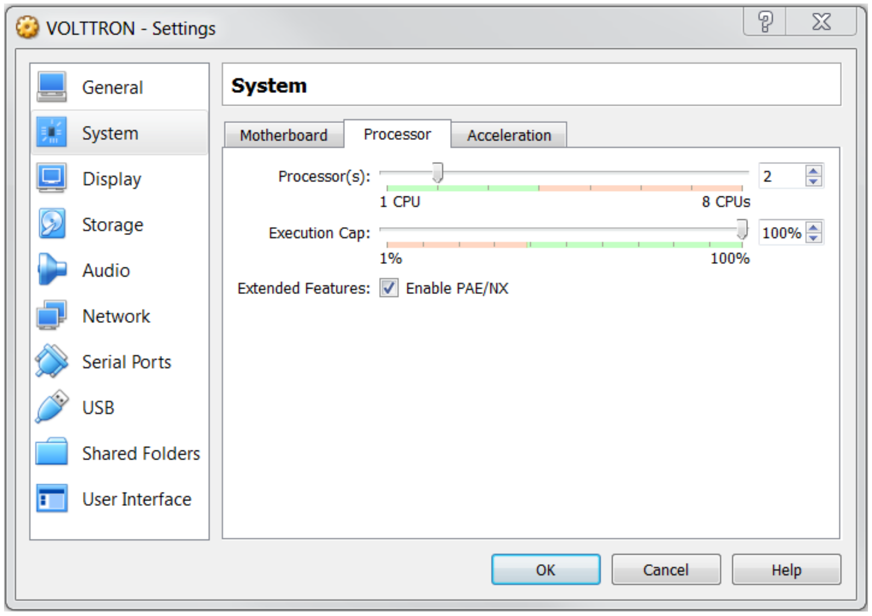


Display

Video Memory: 12 MB (1 MB – 128 MB)
Monitor Count: 1 (1 – 8)
Scale Factor: 100% (100% – 200%)
Acceleration: [✓] Enable 3D Acceleration
[ ] Enable 2D Video Acceleration

Figure 10: Selection of display type

9. In the Settings menu go to the Processor tab and Enable PAE/NX (Figure 11).

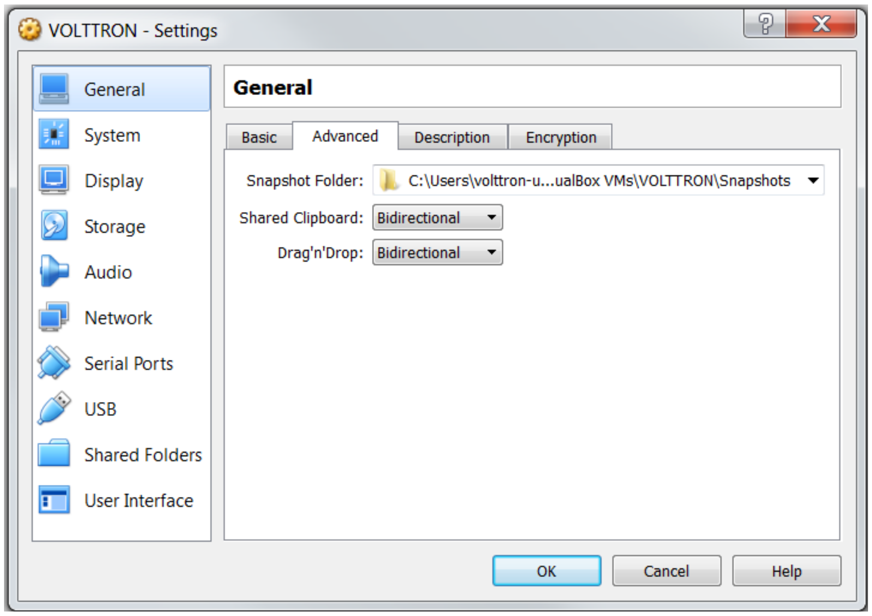


System

Processor(s): 2 (1 CPU – 8 CPUs)
Execution Cap: 100% (1% – 100%)
Extended Features: [✓] Enable PAE/NX

Figure 11: Selection of Processor Parameters

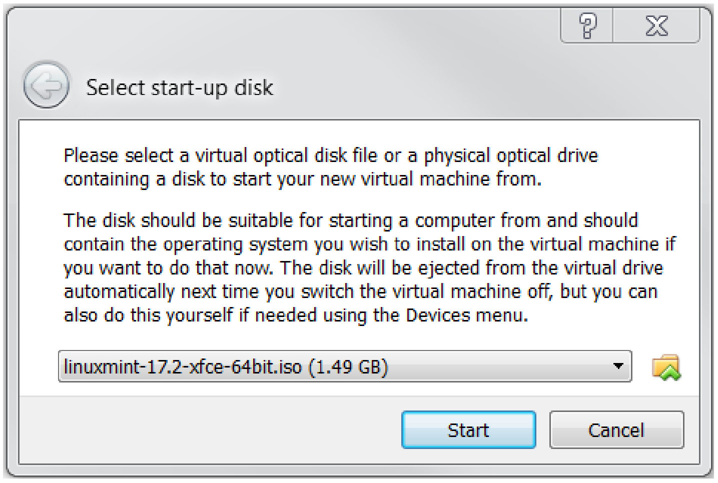
10. To enable bidirectional copy and paste, select the General tab in the VirtualBox Settings. Enable Shared Clipboard and Drag'n'Drop as Bidirectional, as shown in Figure 12.



Snapshot Folder: C:\Users\volttron-u...ualBox VMs\VOLTTRON\Snapshots
Shared Clipboard: Bidirectional
Drag'n'Drop: Bidirectional

Figure 12: Enable Bidirectional Copy and Paste (Shared Clipboard) and Drag'n'Drop

11. With the newly created VM selected, click start (or right click the VM and choose Start). To load the Linux image, select the Linux Mint image file (iso file) you downloaded, and then choose Start, as shown in Figure 13.



Please select a virtual optical disk file or a physical optical drive containing a disk to start your new virtual machine from.

The disk should be suitable for starting a computer from and should contain the operating system you wish to install on the virtual machine if you want to do that now. The disk will be ejected from the virtual drive automatically next time you switch the virtual machine off, but you can also do this yourself if needed using the Devices menu.

linuxmint-17.2-xfce-64bit.iso (1.49 GB)

Figure 13: Loading Linux Image

12. Choose Install Linux Mint (the install icon looks like a DVD media, as shown in Figure 14), proceed to configure installation (language, etc.). The VM will now have Linux Mint installed.

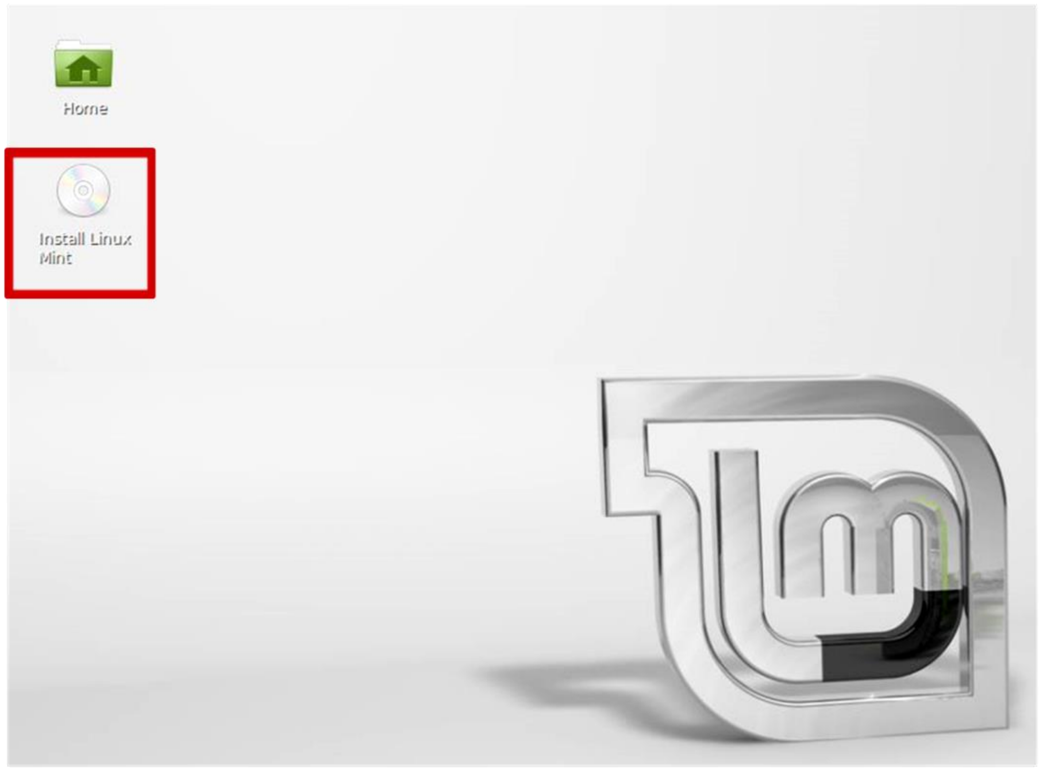
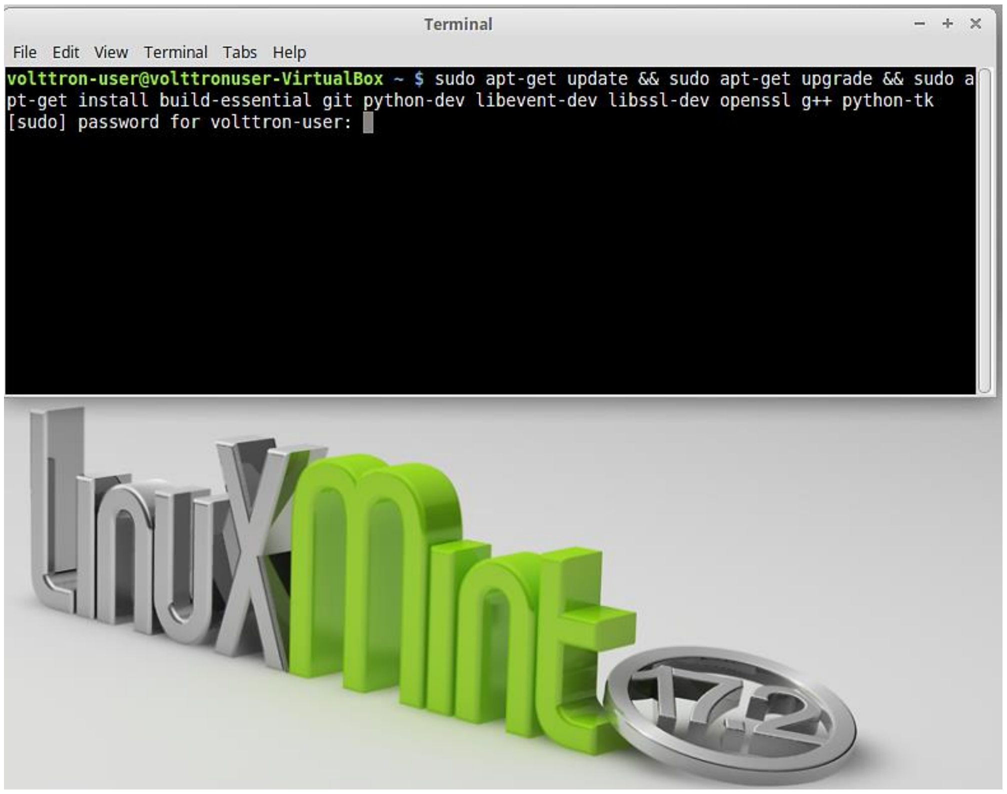


Figure 14: Installing Linux Mint Operating System

2.3 Installing Required Software

VOLTTRON requires the following Linux modules. To install them, open a terminal window and enter the following commands. Figure 15 shows the terminal command to install VOLTTRON software dependencies (terminal commands are bold and in a dotted box):



```
volttron-user@volttronuser-VirtualBox ~ $ sudo apt-get update && sudo apt-get upgrade && sudo a
pt-get install build-essential git python-dev libevent-dev libssl-dev openssl g++ python-tk
[sudo] password for volttron-user: 
```

Figure 15: Linux Mint Terminal Window

- Ensures the installer is up to date:

```
# apt-get update && apt-get upgrade
```

- This installs Git. The transactional network source code including VOLTTRON and other agent code is stored in a Git repository:

```
# apt-get install git
```

- This installs Python DevTools. This is a Python software development tool necessary for running VOLTTRON:

```
# apt-get install python-dev
```

- g++ is a C++ compatible runtime library:

```
# apt-get install g++
```

- build-essential is used to build and install Debian packages:

```
# apt-get install build-essential
```

- Required development library:

```
# apt-get install libevent-dev
# apt-get install libssl-dev
# apt-get install openssl
```

- Optional Python module that allows developers to utilize Python interface library:

```
# apt-get install python-tk
```

- One line command to grab all dependencies:

```
# apt-get update && apt-get upgrade && apt-get install build-essential
openssl git python-dev g++ libevent-dev libssl-dev python-tk
```

2.4 Building the VOLTTRON Platform

Ensure you have installed the required packages before proceeding. Enter the following commands:

1. The following command creates the base VOLTTRON directory, downloads the VOLTTRON source code, and creates a local copy on your machine:

```
$ git clone https://github.com/VOLTTRON/volttron volttron
```

This command creates a directory (`~/volttron`), which contains VOLTTRON and other VOLTTRON related files. The remainder of this guide will assume this file structure and **(`~/volttron`) or will often be referred to as the base platform directory or the base VOLTTRON directory (denoted as `<project directory>`).**

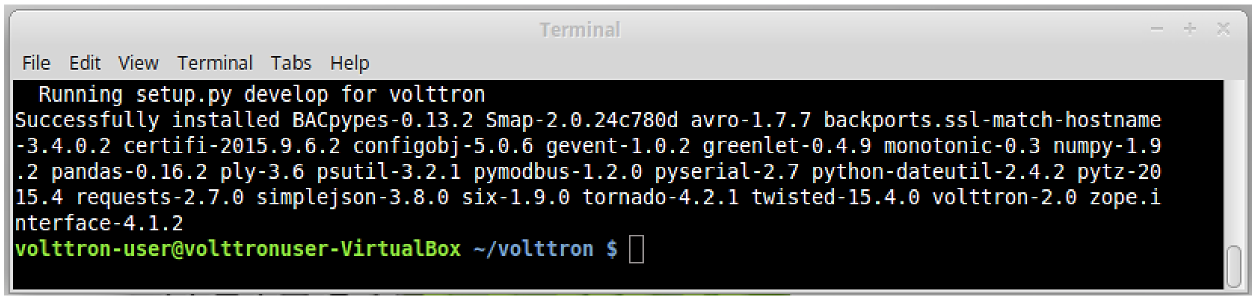
2. Go to the base VOLTTRON directory:

```
$ cd ~/volttron
```

3. VOLTTRON includes scripts that automatically pull down dependencies and build the necessary libraries. The *bootstrap.py* script has to be run only once. Some of the packages (especially Numpy) can be very verbose while they install, wait for the script to complete. To run the *bootstrap.py* script, from the terminal enter the following command:

```
$ python bootstrap.py
```

   ➢ Upon completion of the bootstrap process, the terminal window should appear similar to Figure 16.



```
  Running setup.py develop for volttron
Successfully installed BACpypes-0.13.2 Smap-2.0.24c780d avro-1.7.7 backports.ssl-match-hostname
-3.4.0.2 certifi-2015.9.6.2 configobj-5.0.6 gevent-1.0.2 greenlet-0.4.9 monotonic-0.3 numpy-1.9
.2 pandas-0.16.2 ply-3.6 psutil-3.2.1 pymodbus-1.2.0 pyserial-2.7 python-dateutil-2.4.2 pytz-20
15.4 requests-2.7.0 simplejson-3.8.0 six-1.9.0 tornado-4.2.1 twisted-15.4.0 volttron-2.0 zope.i
nterface-4.1.2
volttron-user@volttronuser-VirtualBox ~/volttron $ 
```

Figure 16: Linux Mint Terminal Window After Successful Completion of the 'bootstrap' Script

4. To test that the installation worked, activate the VOLTTRON platform by running the following command:

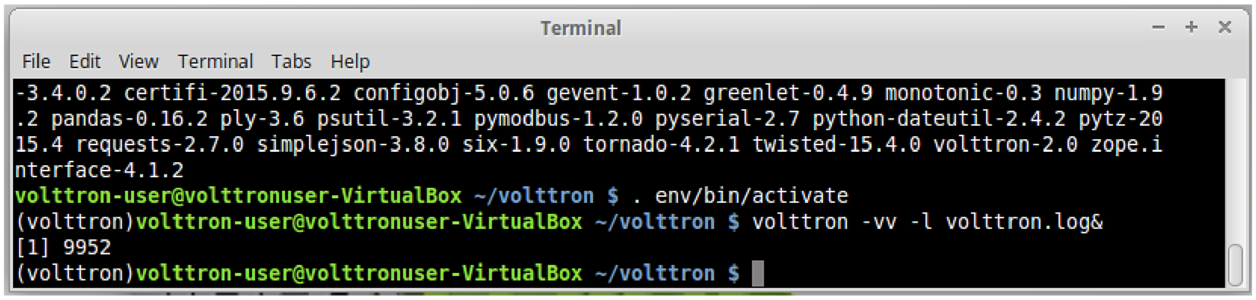
```
$ . env/bin/activate
```

Note there is a space between the "**.**" and "**env**".

5. Start the platform by running the following command:

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

➢ This command not only starts the VOLTTRON platform, but it creates a log file ("-l" option) called *volttron.log* in the base VOLTTRON directory and tells the platform to be very verbose ("-vv" option) when logging platform activity. After execution of these commands, the terminal window should appear similar to Figure 17:



```
-3.4.0.2 certifi-2015.9.6.2 configobj-5.0.6 gevent-1.0.2 greenlet-0.4.9 monotonic-0.3 numpy-1.9
.2 pandas-0.16.2 ply-3.6 psutil-3.2.1 pymodbus-1.2.0 pyserial-2.7 python-dateutil-2.4.2 pytz-20
15.4 requests-2.7.0 simplejson-3.8.0 six-1.9.0 tornado-4.2.1 twisted-15.4.0 volttron-2.0 zope.i
nterface-4.1.2
volttron-user@volttronuser-VirtualBox ~/volttron $ . env/bin/activate
(volttron)volttron-user@volttronuser-VirtualBox ~/volttron $ volttron -vv -l volttron.log&
[1] 9952
(volttron)volttron-user@volttronuser-VirtualBox ~/volttron $ 
```

Figure 17: Linux Mint Terminal After Successfully Activating and Starting the VOLTTRON Platform

➢ If VOLTTRON is being run via SSH and/or it is desired to ensure that VOLTTRON remains running after the SSH session or terminal is closed, then the following command should be used to start the platform:

```
$ volttron -vv -l volttron.log& > /dev/null & disown $-
```

6. Install mysql-connector-python (optional) if MySQL will be used as a platform historian.

```
$ pip install --allow-external mysql-connector-python mysql-connector-python
```

At this point, all required software has been installed and basic configuration has been completed.

2.5 VOLTTRON Home Directory and Configuration

By default, VOLTTRON project configuration files reside in VOLTTRON_HOME, which defaults to (`~/.volttron`). Note that some of the directories listed here are not created until agents are packaged and run.

- `$VOLTTRON_HOME/agents` - contains the agents installed on the platform
- `$VOLTTRON_HOME/certificates` - contains the certificates for use with the licensed (Section 6) VOLTTRON code.
- `$VOLTTRON_HOME/run` - contains files created by the platform during execution. The main ones are the ØMQ files created for publish and subscribe.
- `$VOLTTRON_HOME/ssh` – contains 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` - directory volttron-pkg command creates agent packages.

2.6 Platform Commands

With the exception of packaged agent wheel files, all VOLTTRON files for a platform instance are stored under a single directory known as VOLTTRON home. VOLTTRON home 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.

Configuration files use 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.

A default configuration file, (`$VOLTTRON_HOME/config`), may be created to override default options. If it exists, it will be automatically parsed before all other command-line options. To skip parsing the default configuration file, either move the file out of the way or set the SKIP_VOLTTRON_CONFIG environment variable.

All commands and subcommands 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".

Example of starting VOLTTRON platform:

```
$ env/bin/volttron -c config.ini -l volttron.log
```

Full options:

```
optional arguments:
  -c FILE, --config FILE
                        read configuration from FILE
  -l FILE, --log FILE   send log output to FILE instead of stderr
  -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
```

volttron-ctl commands:

```
install             install agent from wheel
tag                 set, show, or remove agent tag
remove              remove agent
list                list installed agent
status              show status of agents
clear               clear status of defunct agents
enable              enable agent to start automatically
disable             prevent agent from start automatically
start               start installed agent
stop                stop agent
run                 start any agent by path
shutdown            stop all agents
      with Volttron Restricted package installed and enabled
send                send and start agent on a remote platform (VOLTTRON Restricted)
```

volttron-pkg commands: usage: volttron-pkg [-h] {package,repackage,configure} ...

```
optional arguments:
  -h, --help        show this help message and exit

subcommands:
  valid subcommands

  {package,repackage,configure}

    package         create agent package (whl) from a directory or installed agent
    repackage       creates agent package from a currently installed agent
    configure       add a configuration file to an agent package
```

volttron-pkg commands (with Volttron Restricted package installed and enabled):

```
usage: volttron-pkg [-h] [-l FILE] [-L FILE] [-q] [-v] [--verboseness LEVEL]
                    {package,repackage,configure,create_ca,create_cert,sign,verify}
                    ...

VOLTTRON packaging and signing utility

optional arguments:
  -h, --help            show this help message and exit
  -l FILE, --log FILE   send log output to FILE instead of stderr
  -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

subcommands:
  valid subcommands

  {package,repackage,configure,create_ca,create_cert,sign,verify}

    package         Create agent package (whl) from a directory or installed agent
    repackage       Creates agent package from a currently installed agent
    configure           add a configuration file to an agent package
    sign                sign a package
    verify              verify an agent package
```

2.7 Agent Information Exchange: Messaging and Topics

Agents in VOLTTRON communicate with each other using a publish/subscribe mechanism built on the Zero MQ Python library. This allows for flexibility because topics can be created dynamically and the messages can be sent in 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(s)
  - For example an agent may publish results of an experiment or analysis on the “analysis” topic. Additional subtopics might contain information such as site, building, device, algorithm name, etc. The full topic string (including topic and subtopics) would be as follows:

    analysis/site1/building10/pump1/fault_detection
  - 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.
  - All agents should subscribe to the "platform" topic. This is the topic the VOLTTRON will use to send messages to agents, such as "shutdown".
- platform - Base topic used by the platform to inform agents of platform events
  - platform/shutdown - General shutdown command. All agents should exit upon receiving this. Message content will be a reason for the shutdown
  - platform/shutdown_agent - This topic will provide a specific agent ID. Agents should subscribe to this topic and exit if the ID in the message matches their ID.

Agents should set the "From" header. This will allow agents to filter on the "To" message sent back.

2.8 VOLTTRON Shell Scripts

A number of scripts have been developed that simplify launching and interacting with the VOLTTRON platform. These scripts automate the administration of VOLTTRON as much as possible and save time when interacting with the shell (Linux) and/or Python scripts. The scripts are located at (`<project path>/scripts`). One script that is particularly useful is the *pack_install* script (`<project`

`path>/scripts/core/pack_install.sh`). This script allows an agent to be configured, packaged, and installed with one command (execution of the *pack_install.sh* script with required arguments).

The following describes the usage of the *pack_install* script:

1. Prior to running the *pack_install* script ensure the VOLTTRON platform is activated (Section 2.4 step 4)
2. Prior to running the *pack_install* script ensure that VOLTTRON is running (Section 2.4 step 4)
3. Use the *pack_install* script to prepare an agent for execution. To run the *pack_install* script, from the base VOLTTRON directory, enter the following terminal command:

   ```
   ./scripts/core/pack_install.sh <agent directory> <agent config> <agent tag>
   ```

   - **agent directory -** path to outer agent directory (directory containing setup.py for the agent).
   - **agent config -** path to the agent's configuration file.
   - **agent tag -** name associated with agent (optional).

The *make-listener* script located at (`<project path>/scripts/core/make-listener`) can be modified for any agent/application. This script will stop the agent, remove the packaged agent, build (rebuild) the agent, install and configure the agent, tag the agent, and start the agent. The following fields within the script must be configured for each agent launched with the script:

- **SOURCE** - path to outer agent directory.
- **CONFIG** - path to the agent's configuration file.
- **TAG** - name associated with agent (optional).

2.9 Launching the Listener Agent

To test the VOLTTRON installation, build and deploy the Listener agent. If one plans on utilizing an integrated development environment (IDE) for agent development, please refer to Section 5.3 for information on installing and running agents in the Eclipse IDE. The Listener agent is a VOLTTRON example agent. The Listener agent is a very simple but functional agent that can be used as a starting point when one wishes to develop a new agent. The Listener agent logs all activity on the message bus for a particular instance of VOLTTRON. This agent can be helpful when debugging an application or for monitoring what is being published on the message bus by other agents.

The Listener agent could be packaged, configured, and installed using the *pack_install* script (Section 2.8). For completeness, the following instructions will not use the *pack_install* script but launch the agent manually. From the base VOLTTRON directory, enter the following commands in a terminal:

1. Package the agent:

   ```
   $ volttron-pkg package examples/ListenerAgent
   ```

2. Set the configuration file:

   ```
   $ volttron-pkg configure ~/.volttron/packaged/listeneragent-3.0-py2-none-
   any.whl examples/ListenerAgent/config
   ```

3. Install agent into platform (with the platform running):

```
$ volttron-ctl install ~/.volttron/packaged/listeneragent-3.0-py2-none-
any.whl
```

- 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/listeneragent-3.0-py2-none-any.whl as 416b532a-1e36-4e87-887f-04b6feea3f70 listeneragent-3.0`

4. Start the agent:

```
$ volttron-ctl start --name listeneragent-3.0
```

- Agent commands can also use the UUID as an identifier. The agent could be started with the following command:

```
$ volttron-ctl start --uuid 416b532a-1e36-4e87-887f-04b6feea3f70
```

This is helpful when managing multiple instances of the same agent.

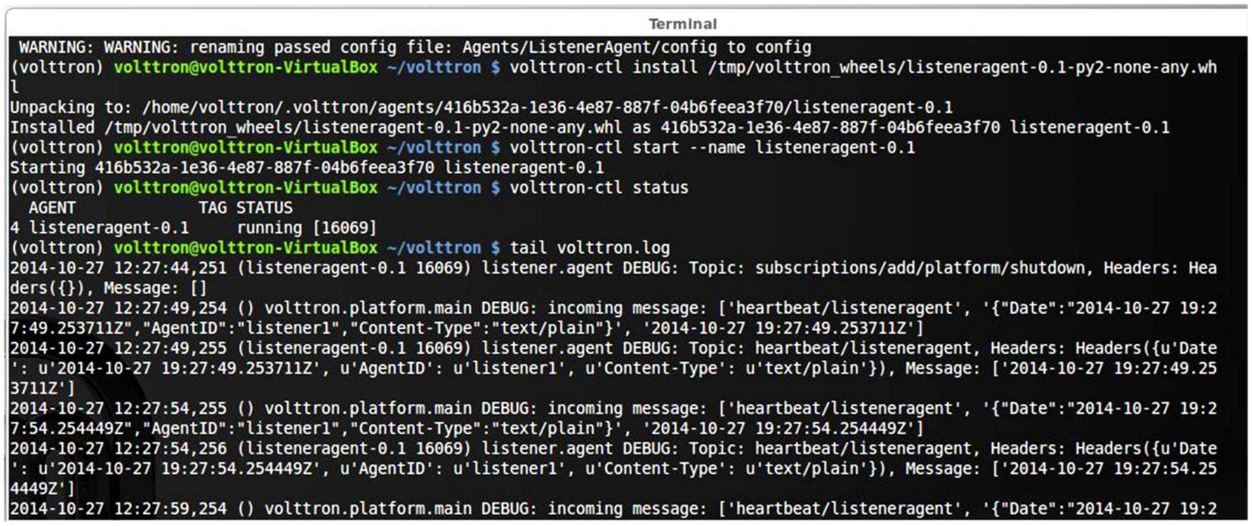
5. Verify that agent is running:

```
$ volttron-ctl status

$ tail -f volttron.log
```

If changes are made to the Listener 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:

```
$ volttron-ctl stop --name listeneragent-3.0

$ volttron-ctl remove --name listeneragent-3.0
```

Figure 18 shows an example of the output produced by the Listener agent.



```
 WARNING: WARNING: renaming passed config file: Agents/ListenerAgent/config to config
(volttron) volttron@volttron-VirtualBox ~/volttron $ volttron-ctl install /tmp/volttron_wheels/listeneragent-0.1-py2-none-any.wh
l
Unpacking to: /home/volttron/.volttron/agents/416b532a-1e36-4e87-887f-04b6feea3f70/listeneragent-0.1
Installed /tmp/volttron_wheels/listeneragent-0.1-py2-none-any.whl as 416b532a-1e36-4e87-887f-04b6feea3f70 listeneragent-0.1
(volttron) volttron@volttron-VirtualBox ~/volttron $ volttron-ctl start --name listeneragent-0.1
Starting 416b532a-1e36-4e87-887f-04b6feea3f70 listeneragent-0.1
(volttron) volttron@volttron-VirtualBox ~/volttron $ volttron-ctl status
  AGENT              TAG STATUS
4 listeneragent-0.1      running [16069]
(volttron) volttron@volttron-VirtualBox ~/volttron $ tail volttron.log
2014-10-27 12:27:44,251 (listeneragent-0.1 16069) listener.agent DEBUG: Topic: subscriptions/add/platform/shutdown, Headers: Hea
ders({}), Message: []
2014-10-27 12:27:49,254 () volttron.platform.main DEBUG: incoming message: ['heartbeat/listeneragent', '{"Date":"2014-10-27 19:2
7:49.253711Z","AgentID":"listener1","Content-Type":"text/plain"}', '2014-10-27 19:27:49.253711Z']
2014-10-27 12:27:49,255 (listeneragent-0.1 16069) listener.agent DEBUG: Topic: heartbeat/listeneragent, Headers: Headers({u'Date
': u'2014-10-27 19:27:49.253711Z', u'AgentID': u'listener1', u'Content-Type': u'text/plain'}), Message: ['2014-10-27 19:27:49.25
3711Z']
2014-10-27 12:27:54,255 () volttron.platform.main DEBUG: incoming message: ['heartbeat/listeneragent', '{"Date":"2014-10-27 19:2
7:54.254449Z","AgentID":"listener1","Content-Type":"text/plain"}', '2014-10-27 19:27:54.254449Z']
2014-10-27 12:27:54,256 (listeneragent-0.1 16069) listener.agent DEBUG: Topic: heartbeat/listeneragent, Headers: Headers({u'Date
': u'2014-10-27 19:27:54.254449Z', u'AgentID': u'listener1', u'Content-Type': u'text/plain'}), Message: ['2014-10-27 19:27:54.25
4449Z']
2014-10-27 12:27:59,254 () volttron.platform.main DEBUG: incoming message: ['heartbeat/listeneragent', '{"Date":"2014-10-27 19:2
```

Figure 18: Sample Output from the Listener Agent

3 VOLTTRON Core Services

Agents that perform critical services that enable and enhance the functionality of VOLTTRON as a control and analytics platform have been segregated from other agent's. Core service agents are now located at:

- `<project directory>/volttron/services/core`

Other agents that are designed to perform a specific function directly for the user and are not critical to the functionality of VOLTTRON have been reclassified as application agents and have been moved to:

- `<project directory>/volttron/applications`

The core VOLTTRON services include device communication, data storage, multi-node communication, and a graphical frontend management service. The remainder of this section will give details on these services including instruction on how to configure and launch these services.

3.1 Configuring and Launching the Weather Agent

The Weather agent, another VOLTTRON service agent, retrieves weather information from the Weather Underground site and shares it with agents running on the platform. The first step to launching the Weather agent is to obtain a developer key from Weather Underground.

3.1.1 Obtaining a Developer Key from Weather Underground

Follow these steps to create a Weather Underground account and obtain a developer key.

- Go to Weather Underground site (Figure 19) the following URL http://www.wunderground.com/weather/api/

- Select, Sign Up for FREE

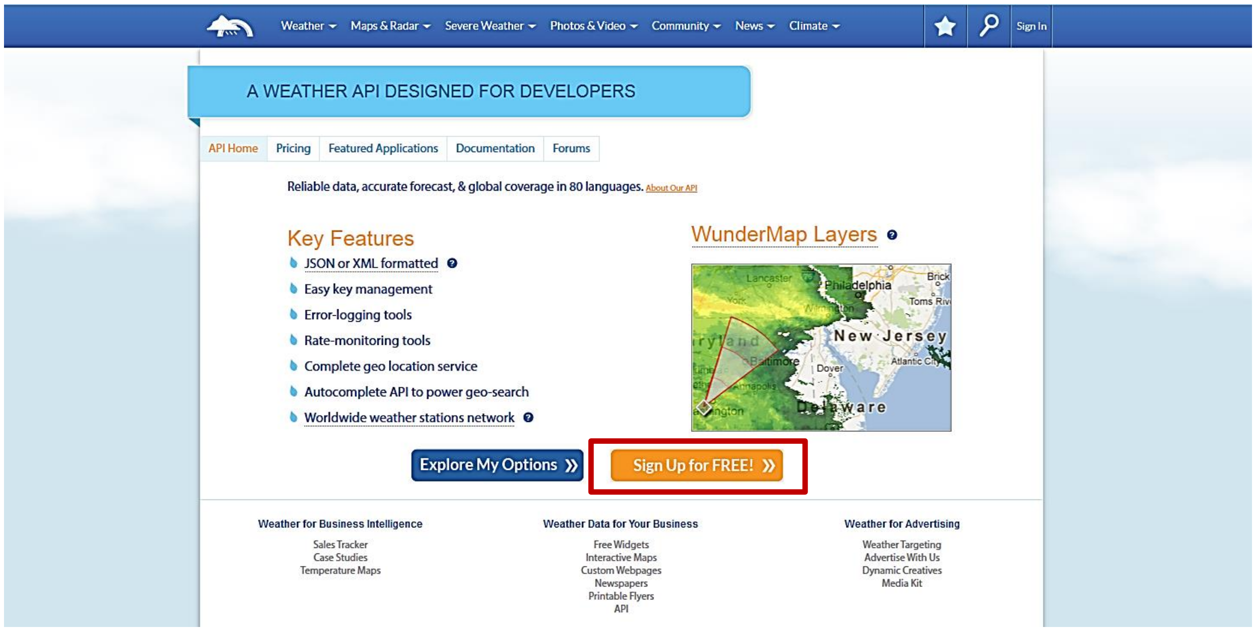
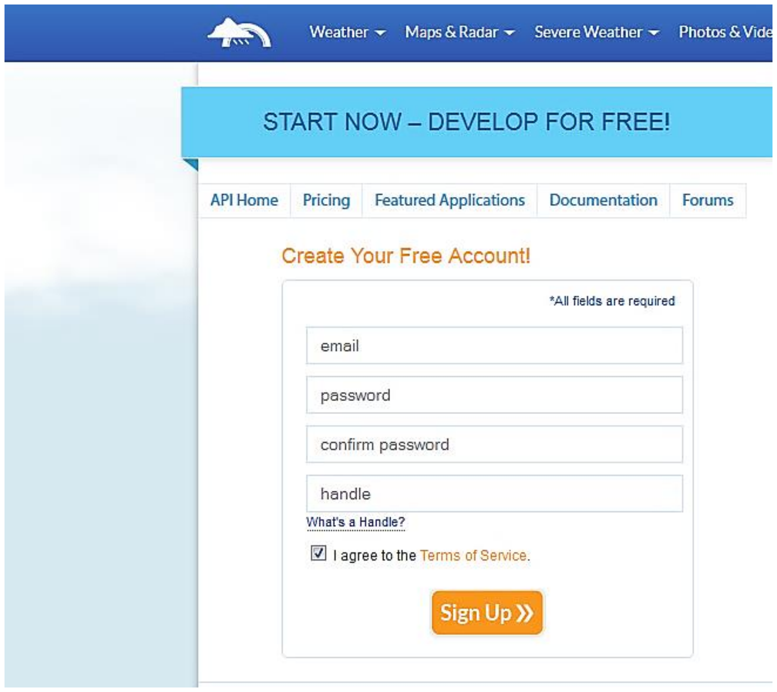


Figure 19: Weather Underground Website

- The window should now look similar to Figure 20. Enter your information to create an account.



Create Your Free Account!

*All fields are required

email

password

confirm password

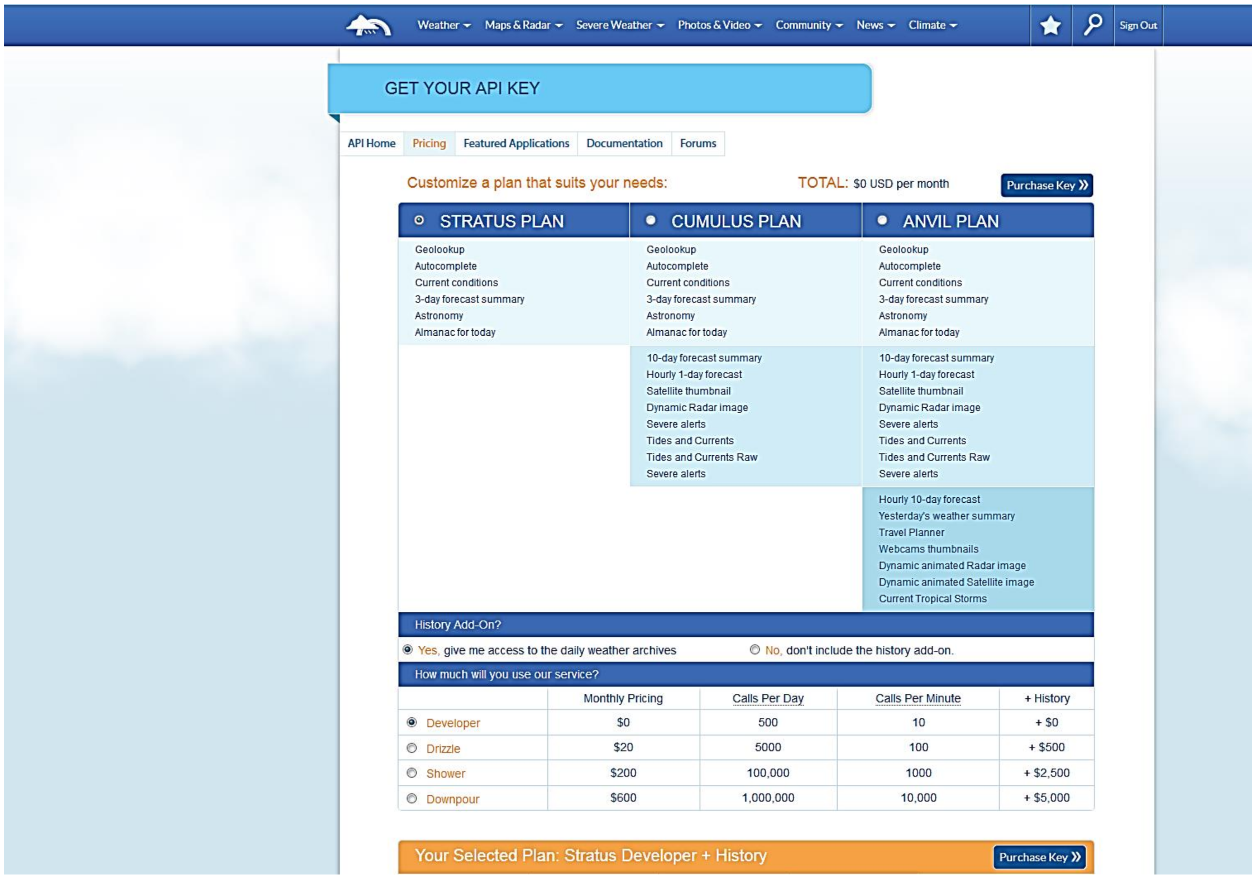
handle

What's a Handle?

☑ I agree to the Terms of Service.

Figure 20: Setting up a Developer Account

- Select a plan that meets your needs. Login to with your username and password and click on "Explore my options button." For most applications, the free plan will be adequate. The window should appear similar to Figure 21:



GET YOUR API KEY

Customize a plan that suits your needs: TOTAL: $0 USD per month

STRATUS PLAN	CUMULUS PLAN	ANVIL PLAN
Geolookup Autocomplete Current conditions 3-day forecast summary Astronomy Almanac for today	Geolookup Autocomplete Current conditions 3-day forecast summary Astronomy Almanac for today	Geolookup Autocomplete Current conditions 3-day forecast summary Astronomy Almanac for today
	10-day forecast summary Hourly 1-day forecast Satellite thumbnail Dynamic Radar image Severe alerts Tides and Currents Tides and Currents Raw Severe alerts	10-day forecast summary Hourly 1-day forecast Satellite thumbnail Dynamic Radar image Severe alerts Tides and Currents Tides and Currents Raw Severe alerts
		Hourly 10-day forecast Yesterday's weather summary Travel Planner Webcams thumbnails Dynamic animated Radar image Dynamic animated Satellite image Current Tropical Storms

History Add-On?

[✓ Yes, give me access to the daily weather archives] No, don't include the history add-on.

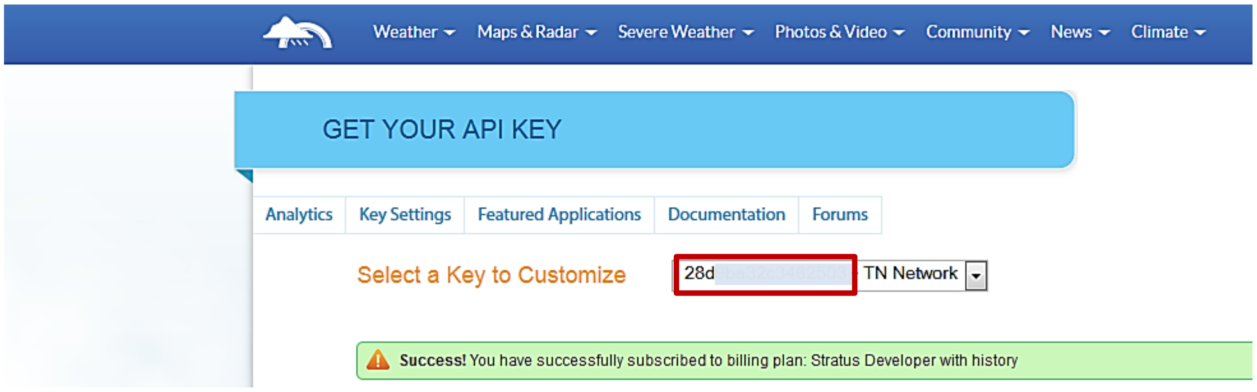
How much will you use our service?

	Monthly Pricing	Calls Per Day	Calls Per Minute	+ History
[✓ Developer]	$0	500	10	+ $0
Drizzle	$20	5000	100	+ $500
Shower	$200	100,000	1000	+ $2,500
Downpour	$600	1,000,000	10,000	+ $5,000

Your Selected Plan: Stratus Developer + History

Figure 21: Creating a WeatherUnderground API Key

- You now have access to your Weather Underground API key. An example API key is shown in the red box of Figure 22:



GET YOUR API KEY

Select a Key to Customize 28d[redacted] TN Network

Success! You have successfully subscribed to billing plan: Stratus Developer with history

Figure 22: Weather Underground API Key

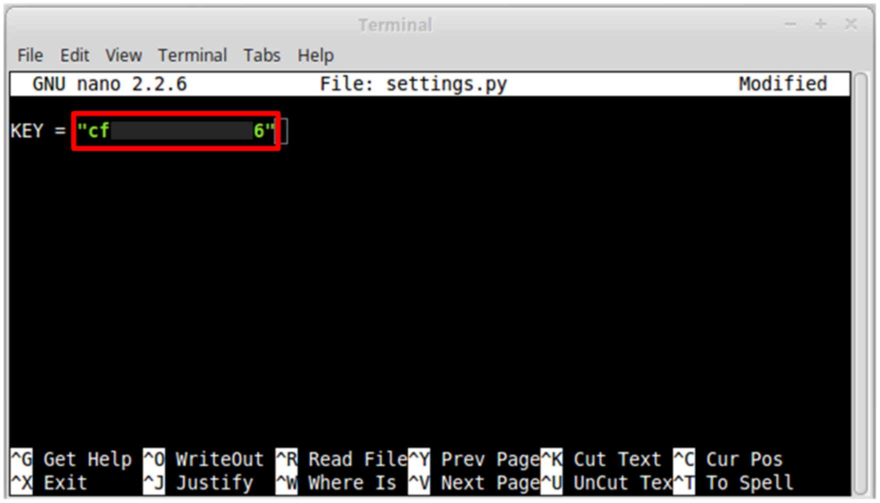
3.1.2 Configuring Weather Agent with API Key and Location

The following steps will show how to configure the Weather agent with the developer key from Weather Underground and how to enter a zip code to get weather data from that zip code.
Edit (`<project directory>/services/core/WeatherAgent/weather/settings.py`) with your Weather Underground key. From the base VOLTTRON directory, enter the following terminal commands:

1. Open *settings.py* at with a text editor or nano:

```
$ nano services/core/weather/settings.py
```

2. Enter a Weather Underground Developer key, as shown in Figure 23:

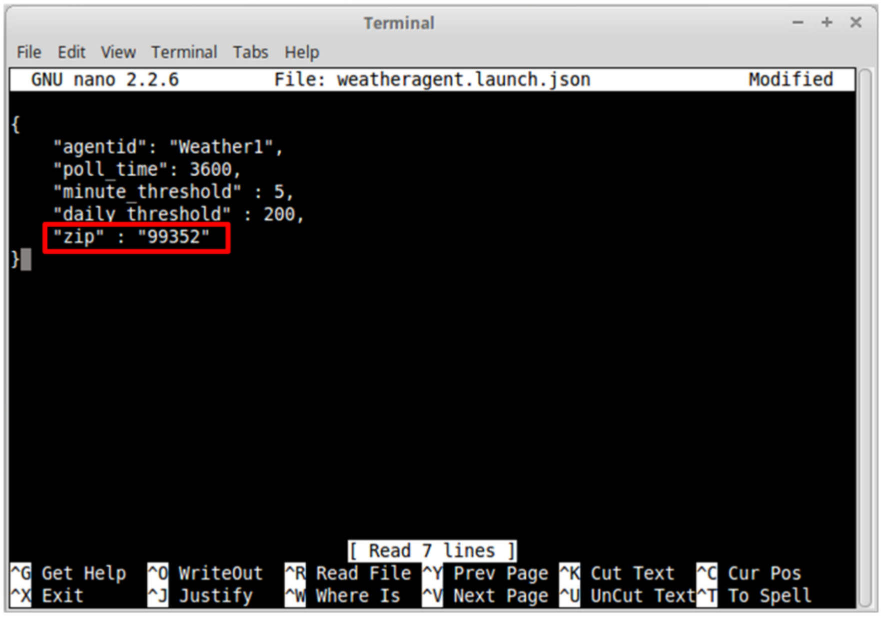


```
 GNU nano 2.2.6             File: settings.py                  Modified

KEY = "cf[redacted]6"

^G Get Help ^O WriteOut ^R Read File^Y Prev Page^K Cut Text ^C Cur Pos
^X Exit     ^J Justify  ^W Where Is ^V Next Page^U UnCut Tex^T To Spell
```

Figure 23: Entering the Weather Underground Developer Key

3. Open the Weather agent's configuration file and edit the "zip" field, as shown in Figure 24:

```
$ nano WeatherAgent/weatheragent.config
```



```
 GNU nano 2.2.6          File: weatheragent.launch.json           Modified

{
    "agentid": "Weather1",
    "poll_time": 3600,
    "minute_threshold" : 5,
    "daily_threshold" : 200,
    "zip" : "99352"
}

                              [ Read 7 lines ]
^G Get Help  ^O WriteOut  ^R Read File ^Y Prev Page ^K Cut Text  ^C Cur Pos
^X Exit      ^J Justify   ^W Where Is  ^V Next Page ^U UnCut Text^T To Spell
```

Figure 24: Entering Zip Code for the Location

3.1.3 Launching the Weather Agent

To launch the Weather agent, enter the following commands from the base VOLTTRON directory:

1. Run the *pack_install* script on the Weather agent:

```
$ ./scripts/core/pack_install.sh services/core/WeatherAgent
services/core/WeatherAgent/weatheragent.config myweather-service
```

- 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/weatheragent-3.0-py2-none-any.whl as a84c91a7-9b0d-491f-aa12-c6abc676a55b weatheragent-3.0`

2. Start the agent:

```
$ volttron-ctl start --tag myweather-service
```

- Agent commands can also use the UUID as an identifier. The agent could be started with the following command:

```
$ volttron-ctl start --uuid a84c91a7-9b0d-491f-aa12-c6abc676a55b
```

  This is helpful when managing multiple instances of the same agent.

3. Verify that agent is running:

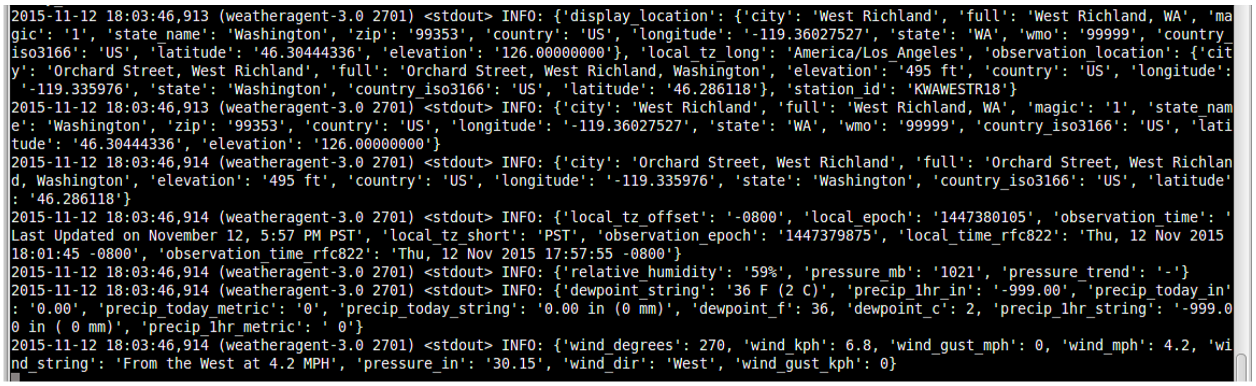
```
$ volttron-ctl status

$ tail -f volttron.log
```

If changes are made to the Weather 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:

```
$ volttron-ctl stop --tag myweather-service

$ volttron-ctl remove --tag myweather-service
```

Then re-build and start the updated agent**.**

Figure 25 shows example output from the Weather agent.

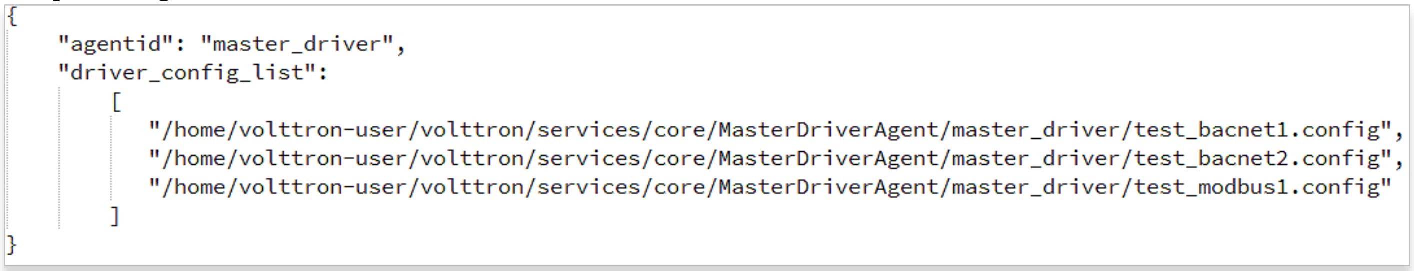


```
2015-11-12 18:03:46,913 (weatheragent-3.0 2701) <stdout> INFO: {'display_location': {'city': 'West Richland', 'full': 'West Richland, WA', 'ma
gic': '1', 'state_name': 'Washington', 'zip': '99353', 'country': 'US', 'longitude': '-119.36027527', 'state': 'WA', 'wmo': '99999', 'country_
iso3166': 'US', 'latitude': '46.30444336', 'elevation': '126.00000000'}, 'local_tz_long': 'America/Los_Angeles', 'observation_location': {'cit
y': 'Orchard Street, West Richland', 'full': 'Orchard Street, West Richland, Washington', 'elevation': '495 ft', 'country': 'US', 'longitude':
 '-119.335976', 'state': 'Washington', 'country_iso3166': 'US', 'latitude': '46.286118'}, 'station_id': 'KWAWESTR18'}
2015-11-12 18:03:46,913 (weatheragent-3.0 2701) <stdout> INFO: {'city': 'West Richland', 'full': 'West Richland, WA', 'magic': '1', 'state_nam
e': 'Washington', 'zip': '99353', 'country': 'US', 'longitude': '-119.36027527', 'state': 'WA', 'wmo': '99999', 'country_iso3166': 'US', 'lati
tude': '46.30444336', 'elevation': '126.00000000'}
2015-11-12 18:03:46,914 (weatheragent-3.0 2701) <stdout> INFO: {'city': 'Orchard Street, West Richland', 'full': 'Orchard Street, West Richlan
d, Washington', 'elevation': '495 ft', 'country': 'US', 'longitude': '-119.335976', 'state': 'Washington', 'country_iso3166': 'US', 'latitude'
: '46.286118'}
2015-11-12 18:03:46,914 (weatheragent-3.0 2701) <stdout> INFO: {'local_tz_offset': '-0800', 'local_epoch': '1447380105', 'observation_time': '
Last Updated on November 12, 5:57 PM PST', 'local_tz_short': 'PST', 'observation_epoch': '1447379875', 'local_time_rfc822': 'Thu, 12 Nov 2015
18:01:45 -0800', 'observation_time_rfc822': 'Thu, 12 Nov 2015 17:57:55 -0800'}
2015-11-12 18:03:46,914 (weatheragent-3.0 2701) <stdout> INFO: {'relative_humidity': '59%', 'pressure_mb': '1021', 'pressure_trend': '-'}
2015-11-12 18:03:46,914 (weatheragent-3.0 2701) <stdout> INFO: {'dewpoint_string': '36 F (2 C)', 'precip_1hr_in': '-999.00', 'precip_today_in'
: '0.00', 'precip_today_metric': '0', 'precip_today_string': '0.00 in (0 mm)', 'dewpoint_f': 36, 'dewpoint_c': 2, 'precip_1hr_string': '-999.0
0 in ( 0 mm)', 'precip_1hr_metric': ' 0'}
2015-11-12 18:03:46,914 (weatheragent-3.0 2701) <stdout> INFO: {'wind_degrees': 270, 'wind_kph': 6.8, 'wind_gust_mph': 0, 'wind_mph': 4.2, 'wi
nd_string': 'From the West at 4.2 MPH', 'pressure_in': '30.15', 'wind_dir': 'West', 'wind_gust_kph': 0}
```

Figure 25: Example Output from the Weather Agent

3.2 Device Communication: Configuring and Launching Master Driver

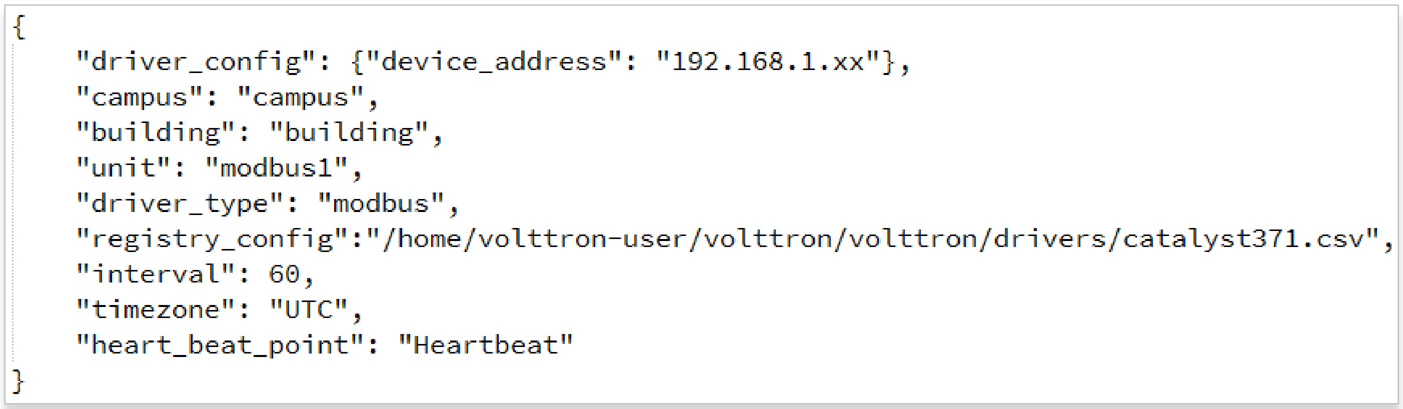
The Master Driver agent (MasterDriverAgent) is the linchpin for communicating with BACnet or Modbus devices in VOLTTRON. It replaces the functionality of the sMAP driver used in previous releases of VOLTTRON and separates device communication from the process of recording and storing device data in a database. Recording of device data to a database is now handled by the Historian agent (documented in Section 3.4). Configuring the Master driver consists of creating a configuration file for each device and pointing the Master Driver at the device configuration file(s). Figure 26 shows an example configuration file for the Master Driver:



```
{
    "agentid": "master_driver",
    "driver_config_list":
        [
            "/home/volttron-user/volttron/services/core/MasterDriverAgent/master_driver/test_bacnet1.config",
            "/home/volttron-user/volttron/services/core/MasterDriverAgent/master_driver/test_bacnet2.config",
            "/home/volttron-user/volttron/services/core/MasterDriverAgent/master_driver/test_modbus1.config"
        ]
}
```

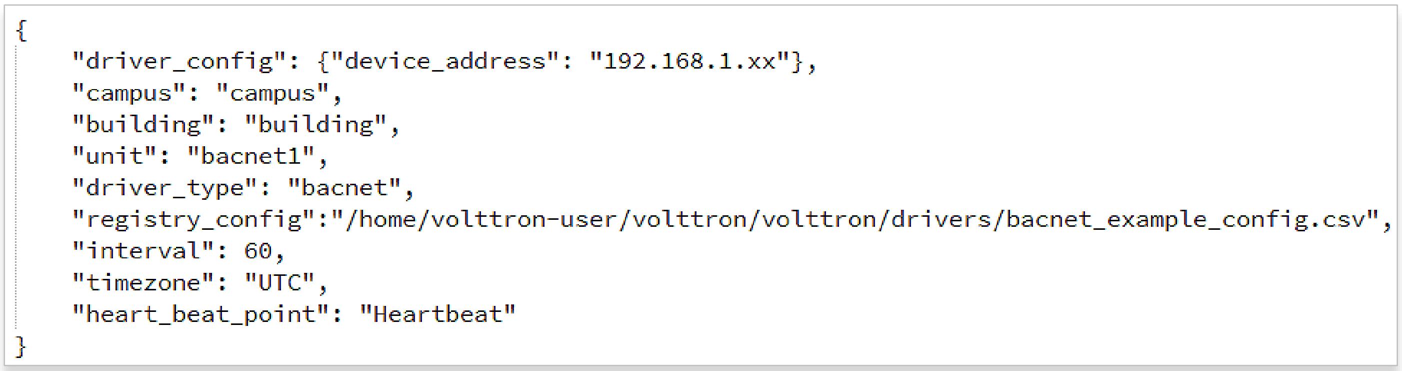
Figure 26: Example Configuration File for Master Driver Agent

The Master Driver configuration file contains a list (**driver_config_list**) where each entry in this list is the path to a configuration file for a BACnet or Modbus device. The device configuration files contain information specific to a single BACnet or Modbus device (e.g., IP address, data polling interval for each device, etc.). Figure 27 shows an example of a Modbus device configuration file and Figure 28 shows an example of a BACnet device configuration file:



```
{
    "driver_config": {"device_address": "192.168.1.xx"},
    "campus": "campus",
    "building": "building",
    "unit": "modbus1",
    "driver_type": "modbus",
    "registry_config":"/home/volttron-user/volttron/volttron/drivers/catalyst371.csv",
    "interval": 60,
    "timezone": "UTC",
    "heart_beat_point": "Heartbeat"
}
```

Figure 27: Example of Modbus Device Configuration File



```
{
    "driver_config": {"device_address": "192.168.1.xx"},
    "campus": "campus",
    "building": "building",
    "unit": "bacnet1",
    "driver_type": "bacnet",
    "registry_config":"/home/volttron-user/volttron/volttron/drivers/bacnet_example_config.csv",
    "interval": 60,
    "timezone": "UTC",
    "heart_beat_point": "Heartbeat"
}
```

Figure 28: Example of BACnet Device Configuration File

The following list gives a description of each of the parameters in the device configuration file:

- **driver_config** - Driver specific settings go here. See the documentation for specific drivers for details.
- **campus** - Campus portion of the device topic. At least one must be specified, all device topics must be unique.
- **building** - Building portion of the device topic (Optional).
- **unit** - Unit portion of the device topic (Optional).
- **path** - Additional topic bits after unit. Useful for specifying sub devices (Optional).
- **driver_type** - Type of driver to use for this device.
- **registry_config** - Configuration file for registers on the device. See the documentation for specific drivers for details.
- **interval** - Timeframe to scrape the device and publish the results.
- **heart_beat_point** - Point (must exist in the registry) on device that is toggled to indicate communication with VOLTTRON platform is functioning.

The Modbus and BACnet registry files tell the driver what the Modbus/BACnet address is for each sensor or control point, what data type it can hold, and if the register is writeable or read-only. The device configuration file tells the Master driver where to find the Modbus and BACnet registry files.

3.2.1 Driver Agents and Interacting with the Master Driver Agent

All VOLTTRON drivers are implemented through the Master Driver agent and are technically sub-agents running in the same process as the Master Driver agent. Each of these driver sub-agents is responsible for creating an interface to a single device. Creating that interface is facilitated by an instance of an interface class. Currently there are two interface classes included: Modbus and BACnet.

In the Master Driver agent directory one will see a directory called interfaces (Master Driver agent directory tree follows):

```
├── master_driver
│   ├── agent.py
│   ├── driver.py
│   ├── __init__.py
│    ├── interfaces
│    │    ├── __init__.py
│    │    ├── bacnet.py
│    │    └── modbus.py
│    └── socket_lock.py
├── master-driver.agent
└── setup.py
```

The files bacnet.py and modbus.py implement the interface class for each respective protocol. (The BACnet interface is mostly just a pass-through to the BACnet Proxy agent, but the Modbus interface is self-contained.)

Looking at those two files is a good introduction into how they work. The file name is used when configuring a driver to determine which interface to use. The name of the interface class in the file must be called “Interface”.

3.2.1.1 Interface Basics

A complete interface consists of two parts: One or more Register classes and the Interface class. The BaseInterface class uses a Register class to describe the registers of a device to the driver sub-agent. This class is commonly sub-classed to store protocol specific information for the Interface class to use. For example, the BACnet Interface uses a sub-classed BaseRegister to store the instance number, object type, and property name of the point on the device represented by the Register class. The Modbus Interface uses several different Register classes to deal with the different types of registers on Modbus devices and their different needs.

The register class contains the following attributes:

- **read_only** - True or False
- **register_type** - "bit" or "byte", used by the driver sub-agent to help deduce some metadata about the point.
- **point_name** - name of the point on the device. Used by the base interface for reference.
- **units** - units of the value, metadata for the driver
- **description** - metadata for the driver
- **python_type** - data (Python) type of the point, used to produce metadata. This must be set explicitly, otherwise it default to int (integer).

```python
class Register(BaseRegister):
    def __init__(self, instance_number, object_type, property_name,
                 read_only, pointName, units, description = ''):
        super(Register, self).__init__("byte", read_only, pointName,
                                       units, description = '')
        self.instance_number = int(instance_number)
        self.object_type = object_type
        self.property = property_name
```

Note that this implementation is incomplete. It does not properly set the register_type or python_type. The Interface class is instantiated by the driver sub-agent to do work. The Interface class contains the following attributes:

- **config_dict** - dictionary of key values pairs from the configuration file's driver_config section.

- **registry_config_str** - the contents of the **registry_config** entry in the driver configuration file. It is up to the Interface class to parse this file according to the needs of the driver.

Here is an example taken from the BACnet driver:

```
    def configure(self, config_dict, registry_config_str):
        # Parse the configuration string.
        self.parse_config(registry_config_str)
        self.target_address = config_dict["device_address"]
        self.proxy_address = config_dict.get("proxy_address",
                                             "platform.bacnet_proxy")
        #Establish routing to the device if needed.
        self.ping_target(self.target_address)
```

And here is the **`parse_config`** method (see section 3.2.2.3 BACnet Registry Configuration File):

```
    def parse_config(self, config_string):
        if config_string is None:
            return
        #Python's CSV file parser wants a file like object.
        f = StringIO(config_string)

        #Parse the CVS file contents.
        configDict = DictReader(f)

        for regDef in configDict:
            #Skip lines that have no address yet.
            if not regDef['Point Name']:
                continue

            io_type = regDef['BACnet Object Type']
            read_only = regDef['Writable'].lower() != 'true'
            point_name = regDef['Volttron Point Name']
            index = int(regDef['Index'])
            description = regDef['Notes']
            units = regDef['Units']
            property_name = regDef['Property']

            register = Register(index,
                                io_type,
                                property_name,
                                read_only,
                                point_name,
                                units,
                                description = description)
            self.insert_register(register)
```

Once the register is created, it must be added with the **`insert_register`** method (final line of **`parse_config`**).

The **`get_point`** method must be implemented by an Interface class. This method obtains the value of a point from a device and returns it. Here is a simple example from the BACnet driver. In this case, it only has to pass the work on to the BACnet Proxy agent for handling:

```
    def get_point(self, point_name):
        register = self.get_register_by_name(point_name)
        point_map = {point_name:[register.object_type,
                                 register.instance_number,
                                 register.property]}
        result = self.vip.rpc.call(self.proxy_address, 'read_properties',
                                   self.target_address, point_map).get()
        return result[point_name]
```

Failure should be indicated by a useful exception being raised. (In this case, the exception raised by the BACnet proxy is left un-handled. This could be improved with better handling when register that does not exist is requested). The register instance for the point can be retrieved with **get_register_by_name** method.

The **set_point** method must be implemented by an Interface class. This function sets the value of a point on a device and ideally returns the actual that was written to the device (confirmation of command). Here is a simple example from the BACnet driver. In this case, the **set_point** method only has to pass the work on to the BACnet Proxy agent for handling:

Failure should be indicated by a useful exception. In this example, only an exception for attempting to write to a read-only point is capture with a useful message. Any other exceptions were left unhandled.

The **scrape_all** method must be implemented by the Interface class. The following is a simple example from the BACnet driver:

```
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)
    args = [self.target_address, value,
            register.object_type,
            register.instance_number,
            register.property]
    result = self.vip.rpc.call(self.proxy_address,
                               'write_property', *args).get()
    return result
```

The **get_registers_by_type** method allows one to obtain a list of registers by their type and whether they are read only or writeable (BACnet currently only uses "byte", "bit" is ignored). As the procedure for handling all the different types in BACnet is the same, we can bundle them all up into a single request from the proxy.

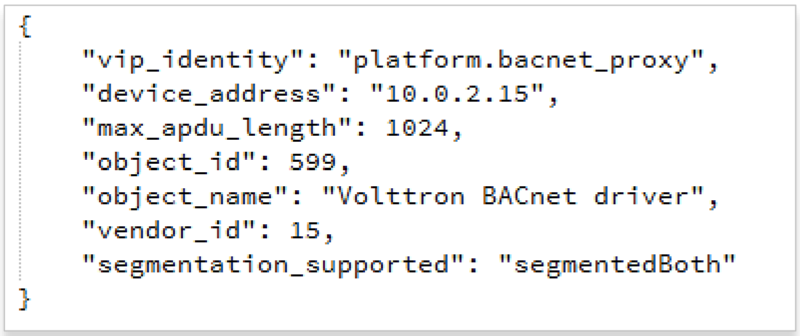
3.2.2 BACnet Configuration

The following section will describe the steps to communicate with BACnet devices within VOLTTRON.

3.2.2.1 BACnet Proxy Agent: Configuration

Communicating with BACnet devices requires that the BACnet Proxy (BACnetProxy) agent is configured and running. Communication with BACnet device(s) on a network occurs via a single virtual

BACnet device. Previous versions of VOLTTRON used one virtual device per device on the network. This only worked in a limited number of circumstances. (This problem is fixed in the legacy sMAP drivers in VOLTTRON 3.0 only). In the new driver architecture, the BACnet Proxy agent enables communicating with BACnet devices and manages the virtual BACnet device. Figure 29 shows an example configuration file for the BACnet Proxy agent:



```
{
    "vip_identity": "platform.bacnet_proxy",
    "device_address": "10.0.2.15",
    "max_apdu_length": 1024,
    "object_id": 599,
    "object_name": "Volttron BACnet driver",
    "vendor_id": 15,
    "segmentation_supported": "segmentedBoth"
}
```

Figure 29: Example Configuration File for BACnet Proxy Agent

The configuration parameters for the BACnet Proxy are defined as follows:

Communication settings

- **vip_identity** - The VIP identity of the agent. Defaults to "platform.bacnet_proxy". This should only be changed if multiple proxies need to be run for communication with multiple BACnet networks.
- **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 (IP address for interface).
    - In some cases one needs to specify the subnet mask of the virtual device or a different port number to listen on. The full format of the address is:

      ADDRESS/SUBNET_MASK:PORT

      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, one would use the address:

      192.168.1.2/24

      If your BACnet network is on a different port (47809) than the default BACnet port (47808), one would use the address:

      192.168.1.2:47809

      If one needs to specify both:

      192.168.1.2/24:47809

BACnet communication settings provide the BACnet Proxy agent with important network information that allows for the correct routing of BACnet device communications.

Device settings

- **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.

  - This setting determines the largest APDU accepted by the BACnet virtual device. Valid options are 50, 128, 206, 480, 1024, and 1476. Defaults to 1024 (Optional).

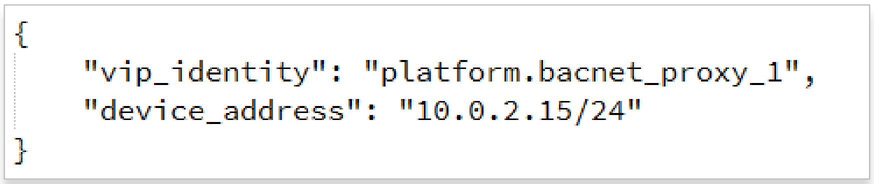
- **object_id** - ID of the device object of the virtual BACnet device. Defaults to 599 (Optional).
- **object_name** - Name of the object. Defaults to "VOLTTRON BACnet driver" (Optional).
- **vendor_id** - Vendor ID of the virtual BACnet device. Defaults to 15 (Optional).
- **segmentation_supported** - (From bacpypes documentation) a vast majority of BACnet communications traffic fits in 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.
  - Possible setting are **segmentedBoth** (default), **segmentedTransmit**, **segmentedReceive**, or **noSegmentation** (Optional).

BACnet device 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. Consequently, there is little reason to change the default settings outside of the **max_apdu_length** (the default is not the largest possible value).

3.2.2.2 BACnet Proxy Agent: Communication with Multiple BACnet Networks

If two BACnet devices are connected to different ports, they are considered to be on different BACnet networks. To communicate with both devices, one will need to run one BACnet proxy agent per network. Each proxy will need to be bound to different ports appropriate to each BACnet network and will need a different VIP identity specified. When configuring drivers, one will need to specify which proxy to use by specifying the VIP identity.

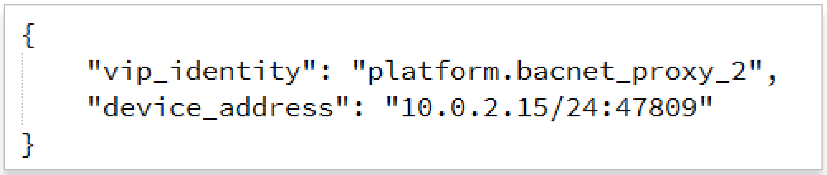
For example, a proxy connected to the default BACnet network (Figure 30):



```
{
    "vip_identity": "platform.bacnet_proxy_1",
    "device_address": "10.0.2.15/24"
}
```

Figure 30: Configuring BACnet Proxy for Multiple BACnet Networks

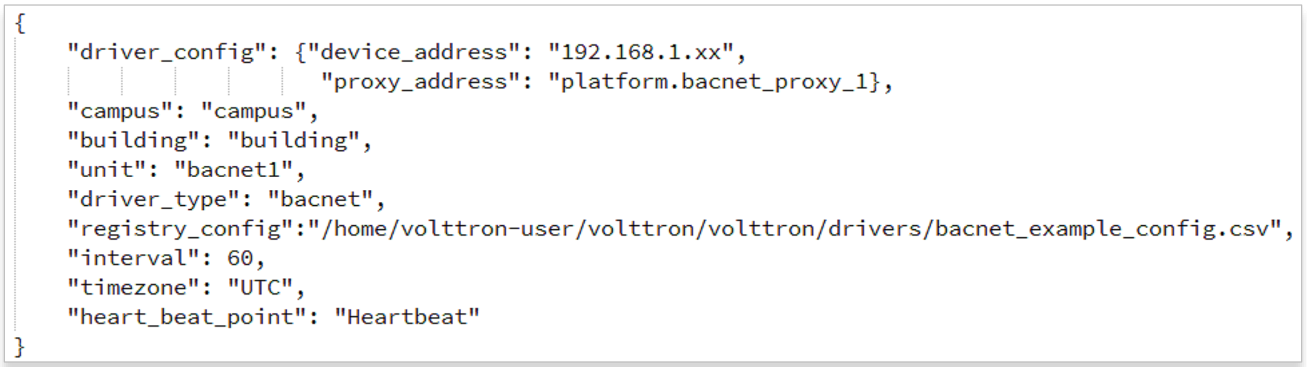
Another BACnet network on port 47809 (Figure 31):



```
{
    "vip_identity": "platform.bacnet_proxy_2",
    "device_address": "10.0.2.15/24:47809"
}
```

Figure 31: Configuring BACnet Proxy for Multiple BACnet Networks (continued)

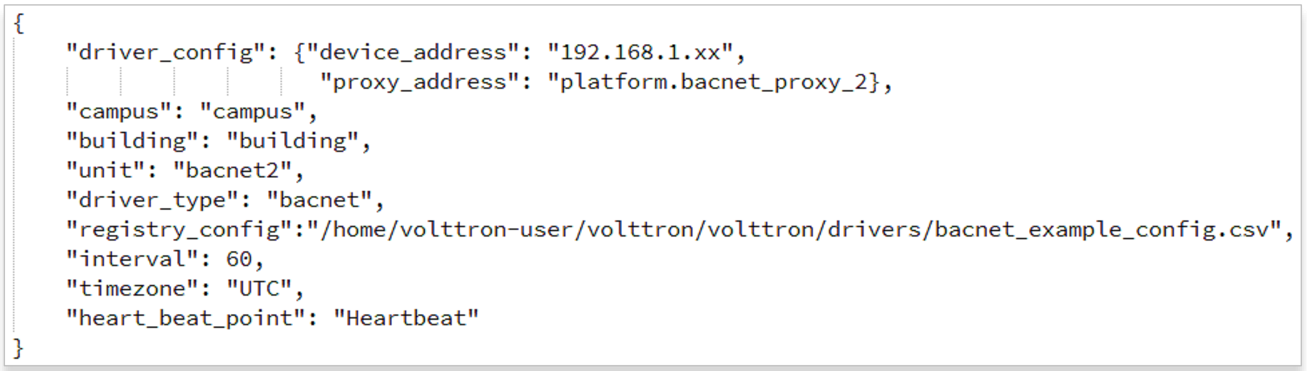
A device on the first network (Figure 32):



```
{
    "driver_config": {"device_address": "192.168.1.xx",
                      "proxy_address": "platform.bacnet_proxy_1},
    "campus": "campus",
    "building": "building",
    "unit": "bacnet1",
    "driver_type": "bacnet",
    "registry_config":"/home/volttron-user/volttron/volttron/drivers/bacnet_example_config.csv",
    "interval": 60,
    "timezone": "UTC",
    "heart_beat_point": "Heartbeat"
}
```

Figure 32: Configuring BACnet Proxy for Multiple BACnet Networks (continued)

A device on the second network (Figure 33):



```
{
    "driver_config": {"device_address": "192.168.1.xx",
                      "proxy_address": "platform.bacnet_proxy_2},
    "campus": "campus",
    "building": "building",
    "unit": "bacnet2",
    "driver_type": "bacnet",
    "registry_config":"/home/volttron-user/volttron/volttron/drivers/bacnet_example_config.csv",
    "interval": 60,
    "timezone": "UTC",
    "heart_beat_point": "Heartbeat"
}
```

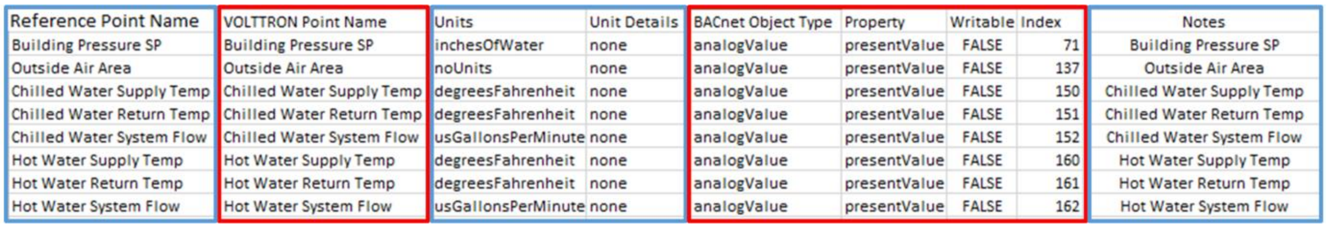
Figure 33: Configuring BACnet Proxy for Multiple BACnet Networks (continued)

Notice that both configuration files use the same BACnet registry configuration (`<project directory/volttron/drivers/bacnet_example_config.csv`). This is fine as long as the registry configuration is appropriate for both devices. The following section describes how to generate and configure a device BACnet registry configuration file.

On some BACnet networks, starting the Master Driver agent before the BACnet Proxy may cause device communication to fail.

3.2.2.3 BACnet Registry Configuration File

To utilize BACnet communications for a device, a key of the registers must be constructed. This key is a file, in comma separated value format, that contains the point name that is published on the message bus, the I/O type (BACnet register), and the point/register address on the device (point address). An example BACnet registry file is shown in Figure 34.



Reference Point Name	VOLTTRON Point Name	Units	Unit Details	BACnet Object Type	Property	Writable	Index	Notes
Building Pressure SP	Building Pressure SP	inchesOfWater	none	analogValue	presentValue	FALSE	71	Building Pressure SP
Outside Air Area	Outside Air Area	noUnits	none	analogValue	presentValue	FALSE	137	Outside Air Area
Chilled Water Supply Temp	Chilled Water Supply Temp	degreesFahrenheit	none	analogValue	presentValue	FALSE	150	Chilled Water Supply Temp
Chilled Water Return Temp	Chilled Water Return Temp	degreesFahrenheit	none	analogValue	presentValue	FALSE	151	Chilled Water Return Temp
Chilled Water System Flow	Chilled Water System Flow	usGallonsPerMinute	none	analogValue	presentValue	FALSE	152	Chilled Water System Flow
Hot Water Supply Temp	Hot Water Supply Temp	degreesFahrenheit	none	analogValue	presentValue	FALSE	160	Hot Water Supply Temp
Hot Water Return Temp	Hot Water Return Temp	degreesFahrenheit	none	analogValue	presentValue	FALSE	161	Hot Water Return Temp
Hot Water System Flow	Hot Water System Flow	usGallonsPerMinute	none	analogValue	presentValue	FALSE	162	Hot Water System Flow

Figure 34: An Example BACnet Registry File

The data fields boxed in red in Figure 34 are important for communication with the device(s) and/or control of the device(s). The fields boxed in blue are for informational purposes and are not required but are often helpful, especially when using a registry key constructed by a third party. Save this file at the location specified by the **registry_config** parameter in the device configuration file (as shown in Figure 27 and Figure 33).

For more details on the BACnet registry file, visit the VOLTTRON Wiki:

https://github.com/VOLTTRON/volttron/wiki/BACnet-Driver

For information on auto-generating a BACnet registry file, visit the VOLTTRON Wiki:

https://github.com/VOLTTRON/volttron/wiki/AutoBacnetConfigGeneration

The following list provides a brief explanation of each field in the BACnet registry file:

- **Volttron Point Name -** the name used to access the point. References to this point will use this name.
- **Units -** used for metadata 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, and 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. If "TRUE", an actuation point will be created in addition to the normal point representing periodically scraped data. The following should be noted:
  - Only points labeled "TRUE" can be accessed through the Actuator agent (Actuator agent).
  - Incorrectly labeled points will cause an error to be returned if the user attempts to write to the point.
  - Currently the BACnet Driver will write at priority 16 (the lowest possible) when write requests are made.
- **Index** - Object ID of the BACnet object. Essentially, this is the "address" for the data point on the device.
- **Notes** - Additional information a vendor or user configured when setting up the device.

3.2.2.4 BACnet Proxy Agent: Launching the Agent

To launch the BACnet Proxy agent, enter the following commands from the base VOLTTRON directory:

1. Run *pack_install* script on BACnet Proxy agent:

```
$ ./scripts/core/pack_install.sh services/core/BACnetProxy
services/core/BACnetProxy/bacnet-proxy.agent bacnet-proxy
```

- 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/bacnet_proxyagent-0.1-py2-none-any.whl as ceb5ec9c-52d9-479a-916c-f957359b49ff bacnet_proxyagent-0.1`

2. Start the agent:

```
$ volttron-ctl start --tag bacnet-proxy
```

3. Verify that agent is running:

```
$ volttron-ctl status

$ tail -f volttron.log
```

If changes are made to the BACnet Proxy 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:

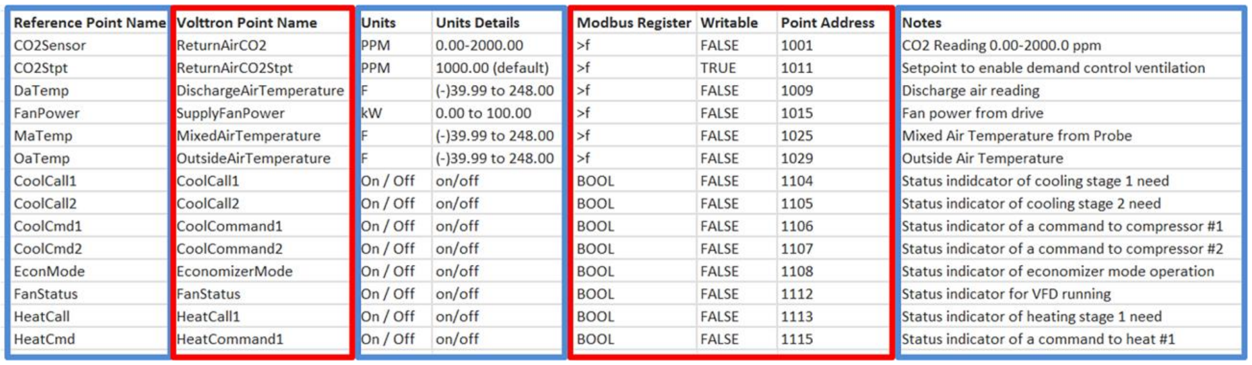
```
$ volttron-ctl stop --tag bacnet-proxy

$ volttron-ctl remove --tag bacnet-proxy
```

3.2.3 Modbus Configuration

Configuration of Modbus device drivers is very similar to BACnet device driver configuration. A Modbus device configuration file is shown in Figure 27. The **driver_config** dictionary in a Modbus device configuration file can contain the following parameters:

- **device_address** - IP address of the device.
- **port** - port the device is listening on. Defaults to 502.
- **slave_id** - slave ID of the device. Defaults to 0.

To utilize Modbus communications for a device, a key of the registers must be constructed. This key is a file, in comma separated value format, that contains the point name that is published on the message bus, the I/O type (Modbus register), and the point/register address on the device (Figure 35).



Reference Point Name	Volttron Point Name	Units	Units Details	Modbus Register	Writable	Point Address	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	Setpoint to enable demand control ventilation
DaTemp	DischargeAirTemperature	F	(-)39.99 to 248.00	>f	FALSE	1009	Discharge air reading
FanPower	SupplyFanPower	kW	0.00 to 100.00	>f	FALSE	1015	Fan power from drive
MaTemp	MixedAirTemperature	F	(-)39.99 to 248.00	>f	FALSE	1025	Mixed Air Temperature from Probe
OaTemp	OutsideAirTemperature	F	(-)39.99 to 248.00	>f	FALSE	1029	Outside Air Temperature
CoolCall1	CoolCall1	On / Off	on/off	BOOL	FALSE	1104	Status indidcator of cooling stage 1 need
CoolCall2	CoolCall2	On / Off	on/off	BOOL	FALSE	1105	Status indicator of cooling stage 2 need
CoolCmd1	CoolCommand1	On / Off	on/off	BOOL	FALSE	1106	Status indicator of a command to compressor #1
CoolCmd2	CoolCommand2	On / Off	on/off	BOOL	FALSE	1107	Status indicator of a command to compressor #2
EconMode	EconomizerMode	On / Off	on/off	BOOL	FALSE	1108	Status indicator of economizer mode operation
FanStatus	FanStatus	On / Off	on/off	BOOL	FALSE	1112	Status indicator for VFD running
HeatCall	HeatCall1	On / Off	on/off	BOOL	FALSE	1113	Status indicator of heating stage 1 need
HeatCmd	HeatCommand1	On / Off	on/off	BOOL	FALSE	1115	Status indicator of a command to heat #1

Figure 35: An Example Modbus Registry File

The data fields boxed in red in Figure 35 are important for communication with the device(s) and/or control of the device(s). The fields boxed in blue are for informational purposes and are not required but are often helpful, especially when using a registry key constructed by a third party. Save this file at the location specified by the **registry_config** parameter in the device configuration file (as shown in Figure 27 and Figure 33).

For more details on the Modbus registry file, visit the VOLTTRON Wiki:

https://github.com/VOLTTRON/volttron/wiki/Modbus-Driver

The following list provides a brief explanation of each field in the BACnet registry file:

- **Volttron Point Name -** the name to used access the point. References to this point will use this name.
- **Units -** used for metadata 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 http://docs.python.org/2/library/struct.html for full documentation. If the supplied format string produces a string, it must only produce one value.
    - ">f" - A big endian 32-bit floating point number.
    - ">l" - A big endian 32-bit integer.
- **Writable -** either “TRUE” or “FALSE”. Determines if the point can be written to. If “TRUE”, an actuation point will be created in addition to the normal point representing periodically scraped data. The following should be noted:
  - Only points labeled “TRUE” can be accessed through the Actuator agent.
  - Incorrectly labeled points will cause an error to be returned if the user attempts to write to the point.
- **Point Address -** Modbus address of the point. Cannot include any offset value; it must be the value of the Modbus address.
- **Notes -** Additional information a vendor or user configured when setting up the device.

3.2.4 eMaster Driver Agent: Launching the Agent

To launch the Master Driver agent, enter the following commands from the base VOLTTRON directory:

1. Run *pack_install* script on Master Driver agent:

```
$ ./scripts/core/pack_install.sh services/core/MasterDriverAgent
services/core/MasterDriverAgent/master-driver.agent master-driver
```

   - 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/master_driveragent-0.1-py2-none-any.whl as ceb7ec9c-52d9-479a-279c-f957359b49ef master_driveragent-0.1`

2. Start the agent:

```
$ volttron-ctl start --tag master-driver
```

3. Verify that agent is running:

```
$ volttron-ctl status

$ tail -f volttron.log
```

If changes are made to the BACnet Proxy 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:

```
$ volttron-ctl stop --tag master-driver

$ volttron-ctl remove --tag master-driver
```

Then rebuild and start the updated agent.

3.3 Device Control: Configuring and Launching the Actuator Agent

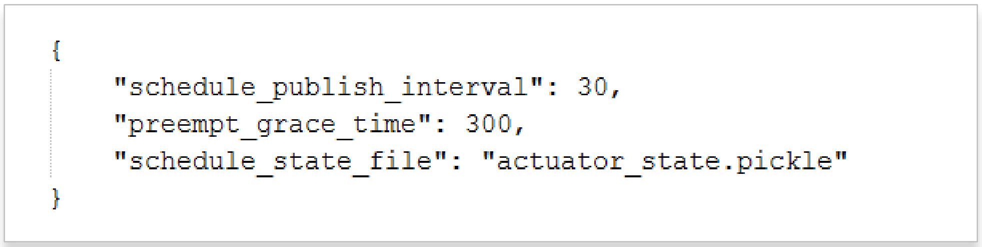
The value contained in the registers on your Modbus or BACnet device will be published to the message bus at a regular interval (the read interval set in the BACnet or Modbus device configuration file). For on demand data or active control of the device, the Actuator agent must be configured and launched. The Actuator agent performs the following platform services:

- device control: The Actuator agent will accept commands from other agents and issue the commands to the specified device. Currently, communication with Modbus and BACnet compatible devices is supported.
- device access scheduling: This service allows the scheduling of agents' access to devices to prevent multiple agents from controlling the same device at the same time.

3.3.1 Configuring the Actuator Agent

Before launching the Actuator agent, we must create or modify the Actuator agent's configuration file (Figure 36). Preemptible, as used in the context of describing device interaction scheduling with the

Actuator agent, means that one agent, the preempted agent, will give up access to a device to allow another agent of higher priority to interact with the device.



```
{
    "schedule_publish_interval": 30,
    "preempt_grace_time": 300,
    "schedule_state_file": "actuator_state.pickle"
}
```

Figure 36: Example Actuator Agent Configuration File

The configuration parameters shown in Figure 36 are defined as follows:

- **schedule_publish_interval** - The interval in seconds between schedule announcements for devices being managed by the Actuator agent (periodically the Actuator agent will publish the state of all currently used devices).
- **preempt_grace_time** - The amount of time given to an application of “low” and “preemptible” priority when a higher priority application requests access to the device.
- **schedule_state_file** - Saved schedule information for each device being managed by the Actuator agent.

The Actuator agent will be able to facilitate communication and control of devices that are configured within the Master Driver agent.

3.3.2 Scheduling a Task

To have active control of a device, an agent can request a block of time be scheduled on the device. An agent can request a task be scheduled by publishing to the “devices/actuators/schedule/request” or calling the Actuator via RPC with the following header:

```
{
    'type': 'NEW_SCHEDULE',
    'requesterID': <Agent ID>, # The name of the requesting agent.
    'taskID': <unique task ID>, # unique (to all tasks) ID for scheduled task.
    'priority': <task priority>, #('HIGH, 'LOW', 'LOW_PREEMPT').
}
```

The schedule request message should be formatted as follows:

```
[
    ["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...
]
```

When constructing a schedule request for a device, the following should be noted:

- Everything in the header is required.
- 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 start or end (or both) of a requested time slot on a device may touch other time slots without overlapping and will not be considered in conflict.
- A request must not conflict with itself.

A schedule block of time and the associated task can have three possible priorities:

HIGH - This task cannot be preempted under any circumstance. This task may preempt other conflicting preemptible 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 Actuator agent configuration file, defaults to 60 seconds) before being revoked. This task may **not** preempt other tasks.

An agent can request access to a device with the following RPC call:

```
result = self.vip.rpc.call(
   'platform.actuator',    #Target agent
   'request_new_schedule', #Method to call
   agent_id,               #Requestor
   "some task",            #TaskID
   'LOW',                  #Priority
   schedule_request        #Request message
   ).get(timeout=10)
```

3.3.3 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': <Agent ID>, #The name of the requesting agent.
    'taskID': <unique task ID>, #ID of task being canceled.
}
```

When canceling a task, the following should be noted:

- The requesterID and taskID must match the original values from the original request header.
- After a task's time has passed, there is no need to cancel it. Doing so will result in a TASK_ID_DOES_NOT_EXIST error.

An agent may cancel a scheduled task as follows:

```
cancel_result = self.vip.rpc.call(
                                  'platform.actuator',         # Target agent
                                  'request_cancel_schedule', # Method
                                  agent_id,                  # Requestor
                                  'some task'                # TaskID
                                  ).get(timeout=10)
```

3.3.4 Actuator Error Reply

If something goes wrong, the Actuator agent will reply to both "get" and "set" on the "error" topic for an actuator. Note: via RPC, errors will be returned in the result object.

```
devices/actuators/error/<full device path>/<actuation point>
```

with this header:

```
{
    'requesterID': <Agent ID>
}
```

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>
}
```

3.3.5 Task Preemption and Schedule Failure

In response to a task schedule request, the Actuator agent will respond on the topic:

```
devices/actuators/schedule/response
```

with this header:

```
{
    'type': <'NEW_SCHEDULE', 'CANCEL_SCHEDULE'>
    'requesterID': <Agent ID from the request>,
    'taskID': <Task ID from the request>
}
```

And, the following message:

```
{
    'result': <'SUCCESS', 'FAILURE', 'PREEMPTED'>,
    'info': <Failure reason, if any>,
    'data': <Data about the failure or cancellation, if any>
}
```

3.3.5.1 Preemption Message

If a higher priority task preempts another scheduled task, the Actuator agent will publish the following message (the field type within the header will be contain CANCEL_SCHEDULE):

```
{
    'agentID': <Agent ID of preempting task>,
    'taskID': <Task ID of preempting task>
}
```

3.3.5.2 Failure Reasons

In most cases, the Actuator agent will try to give good feedback as to why a request failed.

3.3.5.3 Failure Responses from Actuator Agent

The following list contains possible error messages an agent may receive from the Actuator agent. This field corresponds to the info within the Actuator agent response message:

- INVALID_REQUEST_TYPE - Request type was not NEW_SCHEDULE or CANCEL_SCHEDULE.
- MISSING_TASK_ID - Failed to supply a task ID.
- MISSING_AGENT_ID - Agent ID not supplied.
- TASK_ID_ALREADY_EXISTS - The supplied task ID 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 information string.
- CONFLICTS_WITH_EXISTING_SCHEDULES - Schedule conflicts with an existing schedule that it cannot preempt. The data item for the results will contain information 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_ID_DOES_NOT_EXIST - Trying to cancel a task that 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.

3.3.6 Actuator Agent Interaction via PubSub

Once a 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" receive the same response from the Actuator agent.

3.3.6.1 Getting Values

While the device drivers will 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>
```

with this header:

```
{
    'requesterID': <Agent ID>
}
```

Via RPC, the call is:

```
point_value = self.vip.rpc.call(
    'platform.actuator',                # Target agent
    'get_point',                        # Method
    'campus/building/device/some_point' # point
    ).get(timeout=10)
```

3.3.6.2 Setting Values

Values 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:

```
{
    'requesterID': <Agent ID>
}
```

The content of the message is the new, desired value for the actuation point.

An example RPC call is:

```
result = self.vip.rpc.call(
    'platform.actuator',                    # Target agent
    'set_point',                            # Method
    agent_id,                               # Requestor
    'campus/building/unit3/some_point',     # Point to set
    '0.0'                                   # New value
    ).get(timeout=10)
```

3.3.6.3 Actuator Reply

When using pubsub, the Actuator agent will reply to both “get” and “set” on the value topic for an actuator point:

```
devices/actuators/value/<full device path>/<actuation point>
```

with this header:

```
{
    'requesterID': <Agent ID>
}
```

The message contains the value of the actuation point in JSON. The message can be parsed using jsonapi.loads method to parse to Python dictionary (from volttron.platform.agent import utils -> utils.jsonapi).

3.3.6.4 Common Error Types

The following list contains possible error messages an agent may receive from the Actuator agent. This field corresponds to the information within the Actuator agent response message:

- 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.

Other error types involve problem with communication between the Actuator agent and device drivers.

3.3.7 Device Schedule State Announcements

Periodically the Actuator agent will publish the schedule state (what agent has scheduled access to control a device with the Actuator agent) of all currently used 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': <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.

3.3.8 Launching the Actuator Agent

After the Actuator agent has been configured, the agent can be launched. To launch the Actuator agent from the base VOLTTRON directory, enter the following commands in a terminal window:

1. Run *pack_install* script on Actuator agent:

```
$ ./scripts/core/pack_install.sh services/core/ActuatorAgent
services/core/ActuatorAgent/actuator-deploy.service actuator
```

   - 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/master_driveragent-0.1-py2-none-any.whl as ceb7ec9c-52d9-479a-279c-f957359b49ef actuatoragent-0.1`

2. Start the agent:

```
$ volttron-ctl start --tag actuator
```

3. Verify that agent is running:

```
$ volttron-ctl status
$ tail -f volttron.log
```

If changes are made to the Actuator 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:

```
$ volttron-ctl stop --tag actuator
$ volttron-ctl remove --tag actuator
```

Then rebuild and start the updated agent.

The Actuator agent can now be used to interact with Modbus or BACnet devices or simulated devices. Any device, existing or not, can be scheduled. This can be a beneficial debugging tool, especially when testing the functionality of an agent under development.

3.3.9 Tips for Working with the Actuator Agent

The following is a list of tips for working with the Actuator agent:

- An agent can watch the window value from device state announcements to perform scheduled actions within a time slot.
  - If an agent's task is LOW_PREEMPT priority, it can watch for device state announcements, where the window is less than or equal to the grace period (default 60 seconds).
- When considering whether 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?
        - No. Consider LOW and reschedule if preempted.
        - Yes. Use HIGH.
    - Is it problematic to prematurely stop a task once started?
        - No. Consider LOW_PREEMPT and watch the device state announcements for a small window value.
        - Yes. Consider LOW or HIGH.
- If an agent is only observing but needs to assure that no other task is going on while taking readings, it can schedule the time to prevent other agents from "messing" with a devices' state. The device state announcements can be used as a reminder as to when to start watching.

3.4 Data Storage: Platform Historians

Data storage for device sensor measurements and control inputs/outputs as well as data from control and analytics agents is essential for system degradation detection, optimal control of building systems and experimental or control sequence validation. The following section will detail the information necessary to interact with the BaseHistorian (foundation other historians are built on) to create a working historian and detail the configuration and use of the built-in historian services within VOLTTRON.

3.4.1 Developing Historian Agents

VOLTTRON provides a convenient base class (BaseHistorian) for developing Historian agents. The base class enables the following functionality:

- automatically subscribes to all device data topics and agent analysis topics
- caches published data to disk until it is successfully recorded to a Historian
- creates the public facing interface for querying results
- creates a simple interface for concrete implementations (specific Historian implementation that interacts with a database or storage medium) to make a working Historian solution

VOLTTRON provides three built-in Historian (database) options, which are MySQL, SQLite, and sMAP.

3.4.1.1 The BaseHistorian

All historians must inherit from the BaseHistorian class located at:

```
<project directory>/volttron/platform/agent/base_historian.py
```

The BaseHistorian sets up the subscriptions to device data (“devices” topic), agent analysis (“analysis” topic) and any miscellaneous logging information (“datalogger” topic) in its **`starting`** method. The BaseHistorian executes in a greenlet (pseudo-concurrent programming) on a loop that continually adds data to a que (self._event_que) that is made available for the concrete historian implementation (e.g., SQLHistorian).

The concrete historian will need to connect to whatever database that will be used. This can be done in the startup method denoted with the decorator as follows:

```
@Core.receiver('onstart')
def starting(self, sender, **kwargs):
```

This will be the first method run after the agent’s `__init__`. The agent’s connection to the database does not need to be closed until the agent is shutdown. The BaseHistorian agent’s methods (functions) are as follows:

`publish_to_historian``(self, to_publish_list):`

This method is called by the BaseHistorian agent 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': timstamp,
        'source': 'scrape',
        'topic': 'campus/building/unit/point',
        'value': 90,
        'meta': {'units':'F'}
    }
    {
        ...
    }
]
```

- _id - ID of record. All IDs in the list are unique. This is used for internal record tracking.
- timestamp - Python date time object of the time data was published at time zone UTC
- source - source of the data. Can be scrape, analysis, log, or actuator.
- topic - topic data was published on. 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.

`query_topic_list(self)`:

- This method checks the database to obtain the current data topics in the database. New data topic will be added to this list.

```
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 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.

There are numerous database solutions that could be integrated with the platform to act a historian. Each of these databases will have different settings and configurations. Use the SQLHistorian as an example of how a historian properly interacts with the platform and the BaseHistorian agent. The SQLHistorian is located at:

```
<project directory>/services/core/SQLHistorian
```

3.4.2 SQLite Data Historian

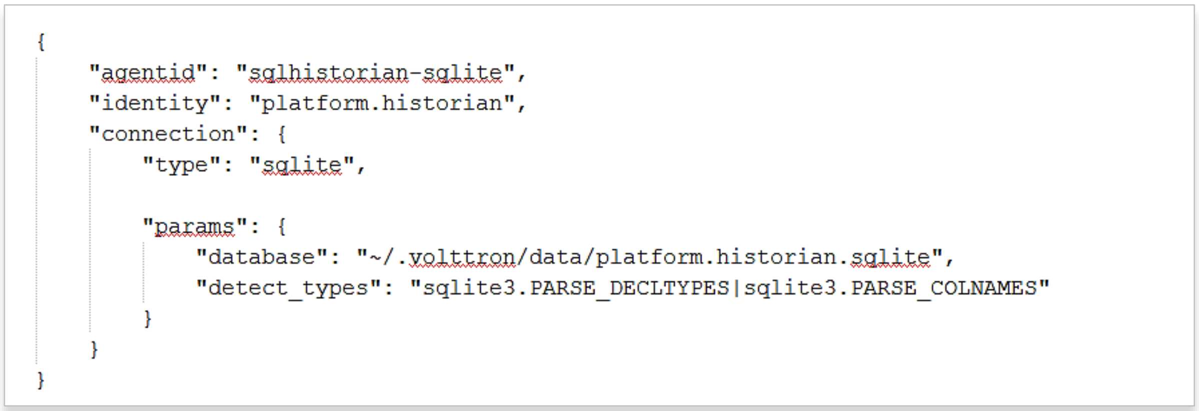
Implementation of SQLite as the VOLTTRON Historian is very simple because the database and tables are automatically created by the SQL Historian (SQLHistorian) when it is launched. VOLTTRON's SQLite implementation supports a local database and works well as a cache or backup to a remote historian (e.g., MySQL or sMAP).

3.4.2.1 Configuring and Launching SQLHistorian Using SQLite

The following steps will describe the process for configuring and launching the SQL Historian. In this section, the SQL Historian will use SQLite.

1. Configure the SQL Historian to use SQLite (Figure 37). The SQLite configuration file for the SQLHistorian is located at:

```
<project directory>/services/core/SQLHistorian/config.sqlite.platform.historian
```



```
{
    "agentid": "sqlhistorian-sqlite",
    "identity": "platform.historian",
    "connection": {
        "type": "sqlite",

        "params": {
            "database": "~/.volttron/data/platform.historian.sqlite",
            "detect_types": "sqlite3.PARSE_DECLTYPES|sqlite3.PARSE_COLNAMES"
        }
    }
}
```

Figure 37: SQL Historian (SQLite) Configuration

The following list provides a brief explanation of each field in the SQL Historian configuration registry file:

- **identity** - Configuring the identity as **platform.historian** indicates that this historian is the primary historian for that VOLTTRON instance and can be queried from a VOLTTRON Central management platform (Section 3.5 for information on VOLTTRON Central). If the platform is not the primary historian or VOLTTRON Central is not being used, then the identity can be any string value.
- **type** - Database type. Currently the SQL Historian supports MySQL (mysql) and SQLite (sqlite).
- **database** - File to use as SQLite database. Default file (with path):

  `$VOLTTRON_HOME/data/platform.historian.sqlite`
- **detect_types** - allows SQLite to intelligently type queried data (SQLite stores all data as strings).

2. Run *pack_install* script on SQL Historian agent:

```
$ ./scripts/core/pack_install.sh services/core/SQLHistorian
services/core/SQLHistorian/config.sqlite.platform.historian lite-historian
```

- 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/sqlhistorianagent-3.0-py2-none-any.whl as 31a6b4bf-f366-4be3-bbb0-39bbcf9e4421 sqlhistorianagent-3.0`

3. Start the agent:

```
$ volttron-ctl start --tag lite-historian
```

4. Verify that agent is running:

```
$ volttron-ctl status
$ tail -f volttron.log
```

If changes are made to the SQL Historian 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:

```
$ volttron-ctl stop --tag lite-historian
$ volttron-ctl remove --tag lite-historian
```

Then rebuild and start the updated agent.

3.4.3 MySQL Data Historian

Implementation of MySQL as the VOLTTRON Historian requires the installation of a MySQL server and the creation of a database. The following steps describe this process:

1. Install the MySQL server. From a terminal, enter the following commands:

```
# apt-get install mysql-server
```

2. During the installation process, a prompt in the terminal will request that the user set the root password for the MySQL server. Set the root password as desired (for this example, the root password will be volttron and the user password will be user_password).

3. Enable MySQL server to auto-start on system boot:

   - Upstart (Ubuntu < 15.04):

   ```
   # update-rc.d mysql defaults
   ```

   - Systemd:

   ```
   # systemctl enable mysql
   ```

4. Start a MySQL shell:

```
# mysql -p
```

Enter the root password at the prompt.

5. Create a user for the database and assign a password to user:

```
mysql> CREATE USER 'volttron_user'@'IP' IDENTIFIED BY 'user_password';
```

The IP value in the previous terminal command should be the IP address for the VOLTTRON instance(s) that will use the database. Multiple IPs can be set by using the wildcard character "%" (e.g., "192.168.%" would allow any connection from volttron_user at an IP starting with "192.168").

6. Create a database (database is named volttron_guide in this example):

```
mysql> CREATE DATABASE volttron_guide;

mysql> USE volttron_guide;
```

7. Create a table called "topics" and a table called "data" within the database volttron_guide:

```
mysql> CREATE TABLE IF NOT EXISTS data (ts timestamp NOT NULL, topic_id
INTEGER NOT NULL, value_string TEXT NOT NULL, UNIQUE(ts, topic_id));

mysql> CREATE INDEX data_idx ON data (ts ASC);

mysql> CREATE TABLE IF NOT EXISTS topics (topic_id INTEGER PRIMARY KEY,
topic_name TEXT NOT NULL, UNIQUE(topic_name));
```

8. Grant the newly created user administrative (root) privileges for the "topics" table and the "data" table:

```
mysql> GRANT ALL PRIVILEGES ON volttron_guide.topics TO 'volttron_user'@'IP'
IDENTIFIED BY user_password;

mysql> GRANT ALL PRIVILEGES ON volttron_guide.data TO 'volttron_user'@'IP'
IDENTIFIED BY user_password;]
```

9. Use the exit command to leave the MySQL shell.

```
mysql> exit
```

Detailed information on MySQL, including syntax and use, can be found at the MySQL Documentation Library:

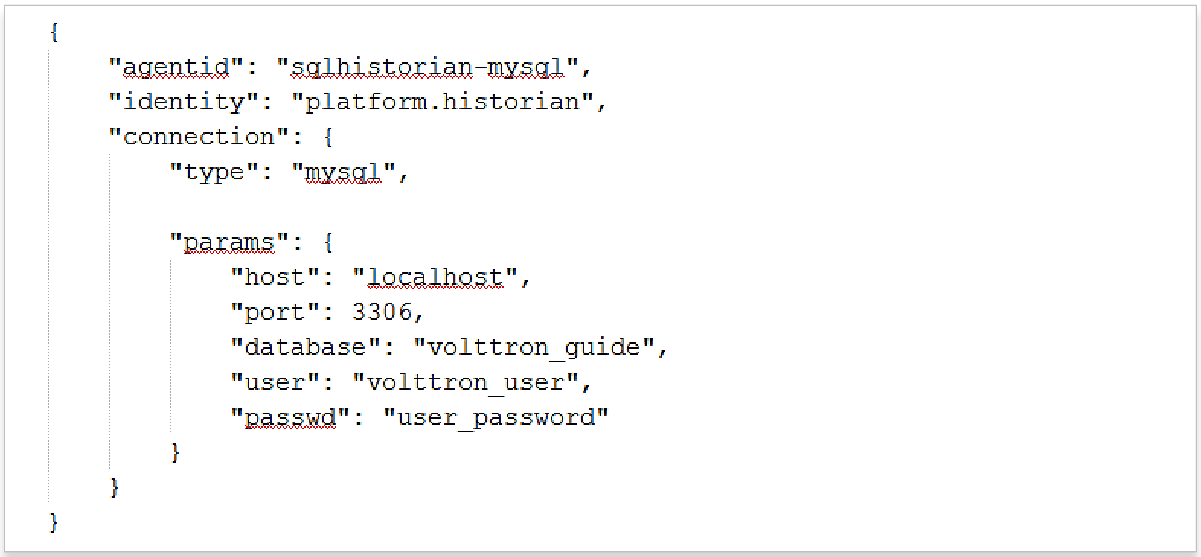
https://dev.mysql.com/doc/index.html

3.4.3.1 Configuring and Launching SQLHistorian Using MySQL

The following steps will describe the process for configuring and launching the SQL Historian. In this section, the SQL Historian will be configured to communicate with a MySQL server:

1. Configure the SQL Historian to use the database created in section 3.4.3 (Figure 38). The MySQL configuration is located at:

   `<project directory>/services/core/SQLHistorian/config.mysql.platform.hsitorian`



```
{
    "agentid": "sqlhistorian-mysql",
    "identity": "platform.historian",
    "connection": {
        "type": "mysql",

        "params": {
            "host": "localhost",
            "port": 3306,
            "database": "volttron_guide",
            "user": "volttron_user",
            "passwd": "user_password"
        }
    }
}
```

Figure 38: SQLHistorian (MySQL) Configuration

The following list provides a brief explanation of each field in the SQL Historian configuration file:

- **agentid** – Arbitrary string identifier for historian.
- **identity** - Configuring the identity as platform.historian indicates that this historian is the primary historian for that VOLTTRON instance and can be queried from a VOLTTRON Central management platform (see Section 3.5 for information on VOLTTRON Management Central). If the platform is not the primary historian or VOLTTRON Central is not being used, then the identity can be any string value.
- **type** - Database type. Currently supports MySQL (mysql) and SQLite (sqlite).
- **host** - IP address for the MySQL server. If the server is running on the same machine as the VOLTTRON instance running the SQLHistorian use localhost.
- **port** - Port MySQL server listens on. Default is 3306 can be modified by editing.
    - ➢ `/etc/mysql/my.cnf` (file can also be located at: `/etc/mysql/mysql.d/my.cnf`)

  The MySQL server must be restarted for modification to *my.cnf* to take effect.
- **database** - Name of the MySQL database created in section 3.4.3 step 6.
- **user** - Name of the user accessing the MySQL database (created in section 3.4.3 step 5).
- **passwd** - Password for user accessing the MySQL database (created in section 3.4.3 step 5).

2. Run *pack_install* script on SQL Historian agent:

```
$ ./scripts/core/pack_install.sh services/core/SQLHistorian
services/core/SQLHistorian/config.mysql.platform.historian mysql-historian
```

- 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/sqlhistorianagent-3.0-py2-none-any.whl as b0c80e2f-8e9c-4fa6-bdcf-0643b377a50e sqlhistorianagent-3.0`

3. Start the agent:

```
$ volttron-ctl start --tag mysql-historian
```

4. Verify that agent is running:

```
$ volttron-ctl status
$ tail -f volttron.log
```

If changes are made to the SQL Historian 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:

```
$ volttron-ctl stop --tag sql-historian
$ volttron-ctl remove --tag sql-historian
```

Then rebuild and start the updated agent.

3.4.4 sMAP Data Historian

sMAP is one data storage option built-in to VOLTTRON. The sMAP Historian (sMAPHistorian) provides the means to interact with a sMAP server to store device data and agent analyses.

3.4.4.1 Installing the sMAP Server (Optional)

If one has access to an existing sMAP server, then the sMAP Historian can be configured to use that server for data storage.

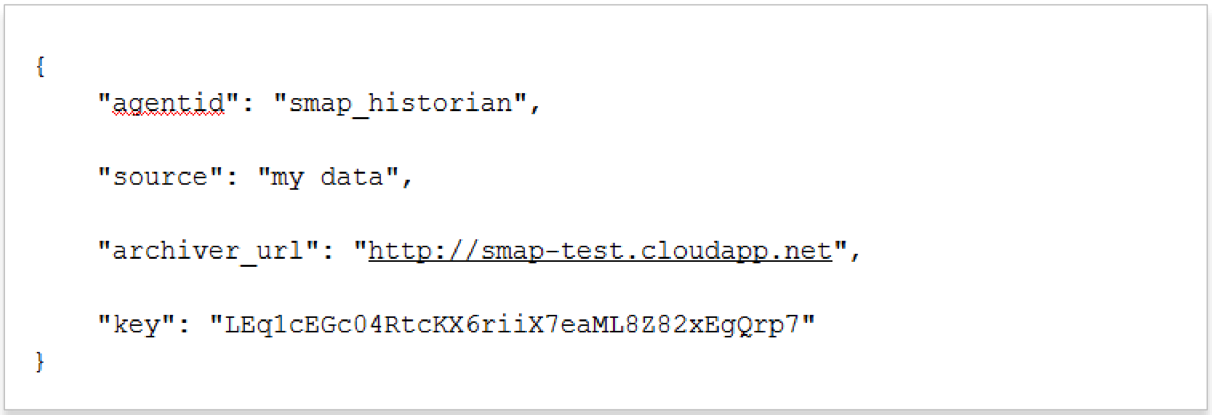
To install your own sMAP instance, follow the installation instructions found at the following URL:

http://pythonhosted.org/Smap/en/2.0/install.html

3.4.4.2 Configuring and Launching the sMAPHistorian

The following steps will describe the process for configuring and launching the sMAP Historian. These instructions assume the user has a preexisting sMAP server.

1. Configure the sMAP Historian (Figure 39):



```
{
    "agentid": "smap_historian",

    "source": "my data",

    "archiver_url": "http://smap-test.cloudapp.net",

    "key": "LEq1cEGc04RtcKX6riiX7eaML8Z82xEgQrp7"
}
```

Figure 39: sMAP Configuration File

The required information is described as follows:

- **archiver_url** - URL for sMAP server.
- **key** - security key created by the sMAP administration interface while configuring the sMAP server.
- **source** - the base of topic for the sMAP database.
- **agentid** - arbitrary string identifier for historian.

5. Run *pack_install* script on sMAP Historian agent:

```
$ ./scripts/core/pack_install.sh services/core/sMAPHistorian
services/core/sMAPHistorian/smap-historian.agent smap-historian
```

   - 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/smaphistorianagent-3.0-py2-none-any.whl as 439790a3-b81a-429b-920f-ab2cc282ca2a smaphistorianagent-3.0`

6. Start the agent:

```
$ volttron-ctl start --tag smap-historian
```

7. Verify that agent is running:

```
$ volttron-ctl status

$ tail -f volttron.log
```

If changes are made to the sMAP Historian 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:

```
$ volttron-ctl stop --tag smap-historian

$ volttron-ctl remove --tag smap-historian
```

Then rebuild and start the updated agent.

3.4.5 OpenEIS Data Analytics Historian

An Analytics Historian (OpenEIS) 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: Users Guide* (Kim et al. 2014) in Section 6 *Create a New Data Map* and Section 7 *Create a New Data Set*. 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"
    }
},
# 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"
#    ]
#        }
    }
}
```

Enabling communication of VOLTTRON with the OpenEIS enables users to take advantage of the data analytic applications on OpenEIS.

3.5 Platform Management: VOLTTRON Management Central (VMC)

VOLTTRON Management Central (VMC) is web-based user interface (UI) for managing VOLTTRON instances on one or more machines. VMC uses a layered security approach that ensures communications are secure and only authorized parties can access the VOLTTRON clients or server (VMC).

3.5.1 VOLTTRON Management Central: Client (Node) Communication Configuration

This section will describe the necessary configuration to allow a VOLTTRON node to communicate with a VOLTTRON Management Central server:

1. Ensure the platform is activated and start VOLTTRON (if it is not already running). From the base VOLTTRON directory, enter the following commands in a terminal:

```
$ . env/bin/activate
$ volttron -vv -l volttron.log&
```

2. Locate the VOLTTRON server key for the VOLTTRON client (node):

```
$ cat volttron.log |grep public
```

The terminal output will be similar to the following:

```
2015-10-01 00:03:19,369 () volttron.platform.main INFO: public key:
RcbUt99IJk3J1nD101WfRrk7VYOwOICAtfHwJrZw1zs
```

The server key here is: **RcbUt99IJk3J1nD101WfRrk7VYOwOICAtfHwJrZw1zs**

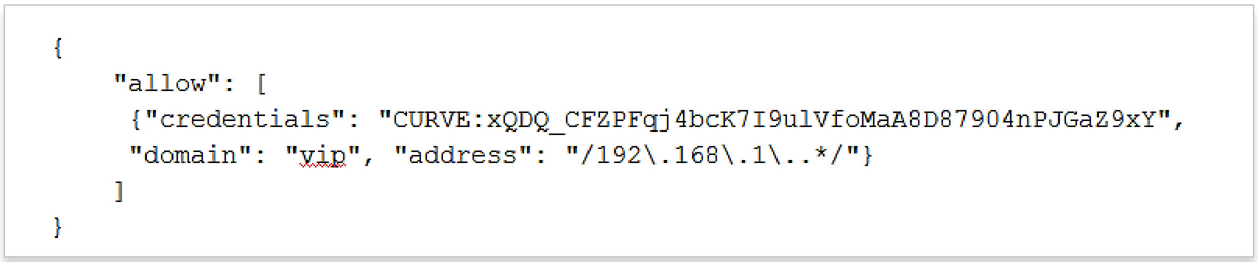
3. Generate a key-pair for server authorization:

```
$ volttron-ctl keypair
```

The terminal output will be similar to the following:

```
public: xQDQ_CFZPFqj4bcK7I9ulVfoMaA8D87904nPJGaZ9xY
secret: Hv_MZlsmZ8bpYcLPq1PyPCHoJ9fjWNaAyGZQJoB2rxM
```

4. Add the public key to: `$VOLTTRON_HOME/auth.json`



```
{
    "allow": [
     {"credentials": "CURVE:xQDQ_CFZPFqj4bcK7I9ulVfoMaA8D87904nPJGaZ9xY",
     "domain": "vip", "address": "/192\.168\.1\..*/"}
    ]
}
```

Figure 40: Example of auth.json File

- **credentials -** The public key used to authenticate other VOLTTRON platforms authorized to access the VOLTTRON node being configured.
- **domain -** Optional argument that can be used if multiple VIP domains are in use. If this argument is used, the node can only be accessed by specified VIP domains.
- **address -** Authorized IP address(s) that may connect to the VOLTTRON node. The example in Figure 40 uses regular expressions to allow wildcard matching. This example allows any local IP on the ".1" subnet (192.168.1.wildcard).

5. Configure the VIP address the node will use for communication. Enter the following command:

```
$ ip link
```

This will show you all the network interfaces.

```
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 16436 qdisc noqueue state UNKNOWN mode
DEFAULT group default
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP
mode DEFAULT group default qlen 1000
    link/ether 02:0e:06:80:e1:6c brd ff:ff:ff:ff:ff:ff
5: wlan0: <BROADCAST,MULTICAST> mtu 1500 qdisc noop state DOWN mode DEFAULT
group default qlen 1000
    link/ether 00:90:4c:11:22:33 brd ff:ff:ff:ff:ff:ff
```

Typically, if connected by Ethernet, the interface will start with "e" and if connected by WIFI, the interface will start with a "w." Choose the correct interface and enter the following command (this example will use a wired connection, the interface is eth0).

```
$ ip addr show eth0
```

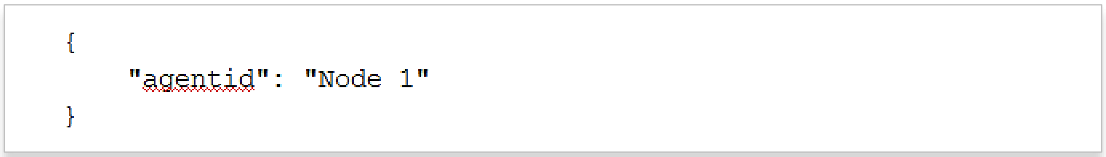
```
link/ether 02:0e:06:80:e1:6c brd ff:ff:ff:ff:ff:ff
inet 192.168.1.213/24 brd 192.168.1.255 scope global eth0
inet6 fe80::e:6ff:fe80:e16c/64 scope link
    valid_lft forever preferred_lft forever
```

The IP address for the node is on the second line after "inet." This information must be added to the platform config in (`$VOLTTRON_HOME/config`). Create this file and add the following content:

```
[volttron]
vip-address=tcp://192.168.1.213:29216
```

This is the IP address for the node and the preferred communication port. Any port available can be used (29216 is typically unused by other applications).

6. Configure the Platform agent. Edit the Platform agent's configuration file (Figure 41) as follows (`<project directory>/services/core/Platform/config`):



```
{
    "agentid": "Node 1"
}
```

Figure 41: Platform Agent Configuration File

   The value for **agentid** is configurable (choose any string that does not contain special character such as *^&).

7. Start the Platform agent and enable the Platform agent to auto start with the platform (if desired). Run the following commands:

```
$ ./scripts/core/pack_install.sh services/core/Platform
services/core/Platform/config platform-node1

$ volttron-ctl start --tag platform-node1

$ volttron-ctl enable --tag platform-node1
```

The following sub-section will describe how to register the platform with a VOLTTRON Management Central instance.

3.5.2 Configure VOLTTRON Management Central (VMC)

The following steps will show one how to configure and launch a VOLTTRON Management Central (VMC) instance.

1. Generate a password hash for each user. The password is used to authenticate users when logging onto VMC:

```
$ echo -n user-pass | sha512sum | awk '{print $1}'
```

   The user hash will appear similar to:

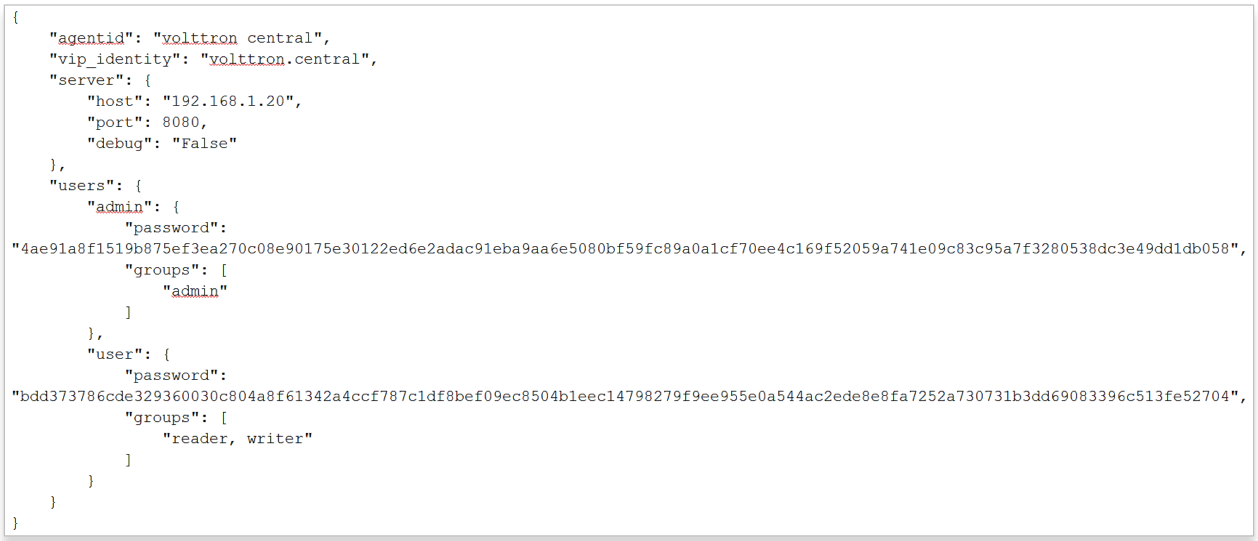
```
bdd373786cde329360030c804a8f61342a4ccf787c1df8bef09ec8504b1eec14798279f9ee955e
0a544ac2ede8e8fa7252a730731b3dd69083396c513fe52704
```

```
echo -n admin-pass | sha512sum | awk '{print $1}'
```

   The admin hash will appear similar to:

```
4ae91a8f1519b875ef3ea270c08e90175e30122ed6e2adac91eba9aa6e5080bf59fc89a0a1cf70
ee4c169f52059a741e09c83c95a7f3280538dc3e49dd1db058
```

2. Edit the VMC configuration file (Figure 42). The VMC configuration file is located at (`<project directory>/services/core/VolttronCentral/config`).



```
{
    "agentid": "volttron central",
    "vip_identity": "volttron.central",
    "server": {
        "host": "192.168.1.20",
        "port": 8080,
        "debug": "False"
    },
    "users": {
        "admin": {
            "password":
"4ae91a8f1519b875ef3ea270c08e90175e30122ed6e2adac91eba9aa6e5080bf59fc89a0a1cf70ee4c169f52059a741e09c83c95a7f3280538dc3e49dd1db058",
            "groups": [
                "admin"
            ]
        },
        "user": {
            "password":
"bdd373786cde329360030c804a8f61342a4ccf787c1df8bef09ec8504b1eec14798279f9ee955e0a544ac2ede8e8fa7252a730731b3dd69083396c513fe52704",
            "groups": [
                "reader, writer"
            ]
        }
    }
}
```

Figure 42: VOLLTRON Management Central Agent Configuration File

- **server** - Contains connection information for the server. The configurable parameters are:
  - **host** - IP of VMC server.
  - **port** - communication port.
  - **debug** - generate additional messages and warnings for use with troubleshooting and testing.
- **users** - Contains the name, password, and privileges of all users allowed to access the VMC instance.
  - **user** - login name of user allowed access to VMC.
  - **password** - password hash (created in previous step). When logging into VMC, the user will use the non-hashed password.
  - **groups** - permission access on VMC for user on VMC. Values are admin, writer, and reader.

3. Start and enable the Platform agent and the VMC agent (ensure that VOLTTRON is running):

```
$ ./scripts/core/pack_install.sh services/core/Platform
services/core/Platform/config platform-vc
$ volttron-ctl start --tag platform-vc
$ volttron-ctl enable --tag platform-vc

$ ./scripts/core/pack_install.sh services/core/VolttronCentral
services/core/VolttronCentral/config vmc
$ volttron-ctl start --tag vmc
$ volttron-ctl enable --tag vmc
```

VMC is now ready to be used.

4. Login to the VMC using a web browser. Enter the following URL:

http://192.168.1.20:8080

This should be the IP address and port used in the VMC configuration file.

5. Login to VMC. Use the user and password (non-hashed password) configured in the VMC configuration file (step 1 of the current section).

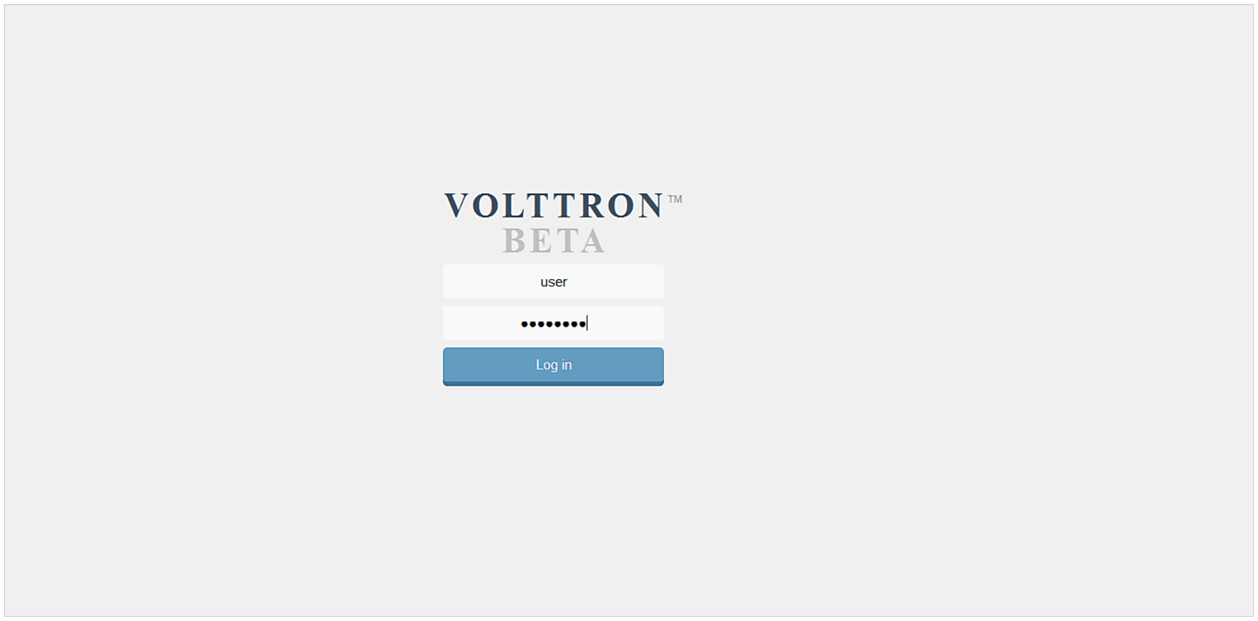


Figure 43: VMC Web Interface (Login)

6. Register the VOLTTRON node configured in Section 3.5.1.

   - Select Platforms button in top right corner of browser window (Figure 44):

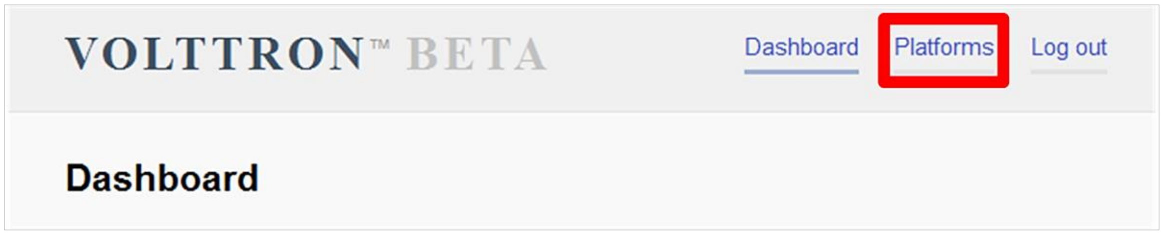
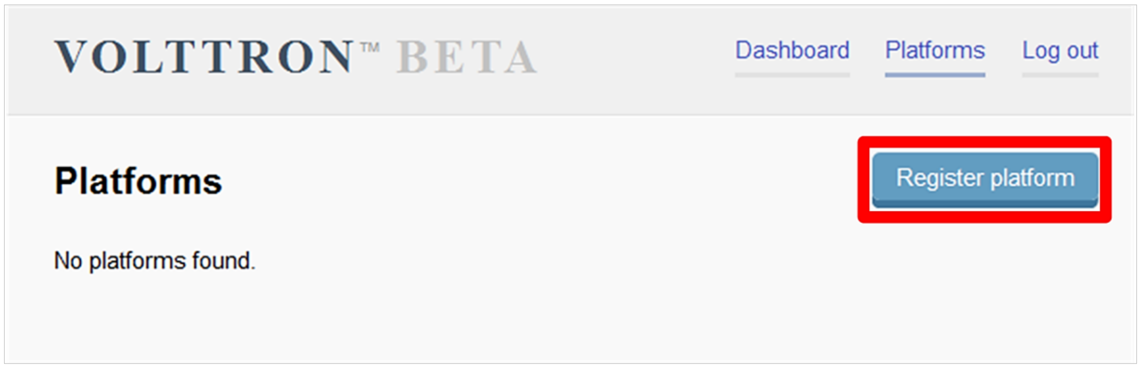


Figure 44: VMC Web Interface (continued)

- Select "Register Platform" button (Figure 45).



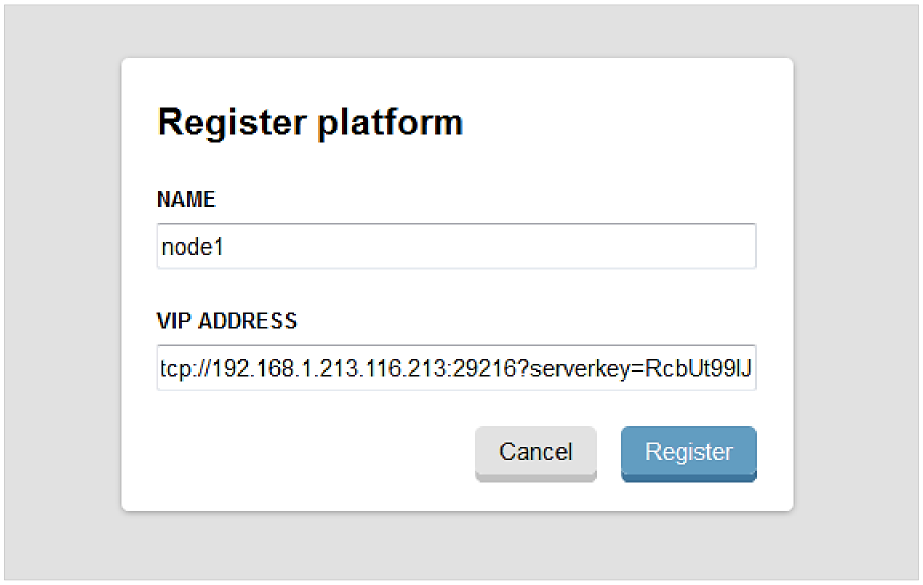
Platforms

No platforms found.

Figure 45: VMC Web Interface (continued)

- Enter a name for the VOLTTRON node and the VIP address. The VIP address is constructed as follows:
  - `tcp://192.168.1.213.116.213:29216?serverkey=RcbUt99IJk3J1nD101WfRrk7VYOwOICAtfHwJrZw1zs&publickey=xQDQ_CFZPFqj4bcK7I9ulVfoMaA8D87904nPJGaZ9xY&secretkey=Hv_MZlsmZ8bpYcLPq1PyPCHoJ9fjWNaAyGZQJoB2rxM`
    - `node address` - the IP and port configured in section 3.5.1 step 5.
    - `serverkey` - the server key was located in section 3.5.1 step 2.
    - `publickey` - the public key was configured in section 3.5.1 step 2.
    - `secretkey` - the secret key was located in section 3.5.1 step 2.
    - The "?" and "&" symbols are required and delimit the fields required for platform registration.

Figure 46 shows Register platform window.

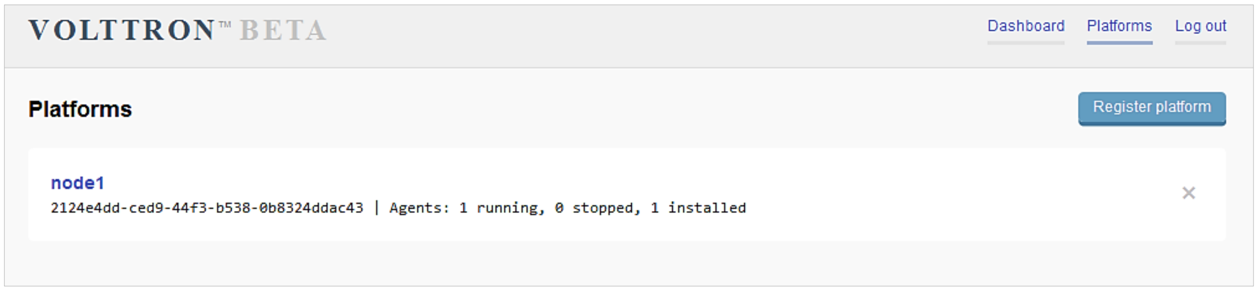


NAME: node1

VIP ADDRESS: tcp://192.168.1.213.116.213:29216?serverkey=RcbUt99IJ

Figure 46: VMC Web Interface (Platform Registration)

The VOLTTRON node will now be listed in the VMC. This VOLTTRON node can now be managed and monitored from the VMC (Figure 47).



Platforms

node1

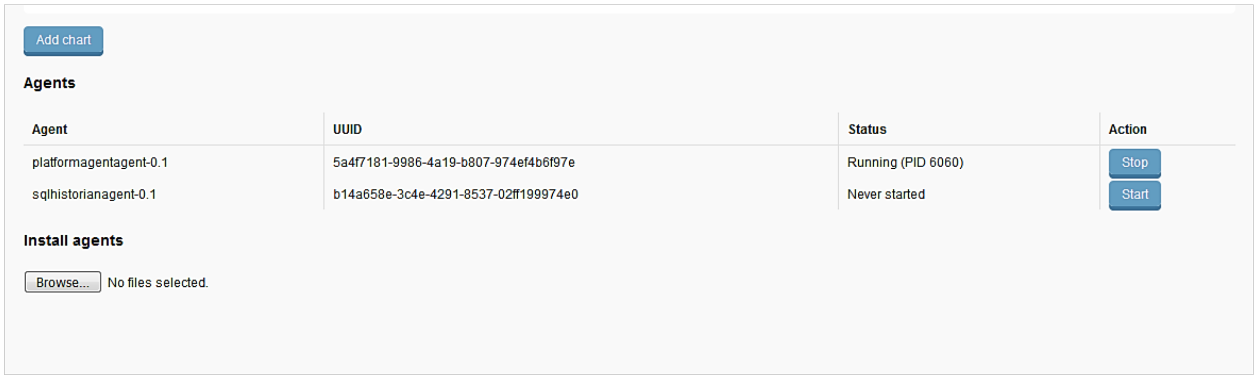
2124e4dd-ced9-44f3-b538-0b8324ddac43 | Agents: 1 running, 0 stopped, 1 installed

Figure 47: VMC Web Interface (Node Registered)

7. Agents must still be packaged and installed prior to managing them on VMC. From the VOLTTRON (logged in locally or via SSH to the node), run the pack install script on the SQL Historian:

```
$ . scripts/core/pack_install.sh services/core/SQLHistorian
services/core/SQLHistorian/config.sqlite.platform.historian historian
```

Select the platform (node1 in Figure 47). The two agents installed on node1 should be shown at the bottom of the page (Figure 48):



Agents

Agent	UUID	Status	Action
platformagentagent-0.1	5a4f7181-9986-4a19-b807-974ef4b6f97e	Running (PID 6060)	Stop
sqlhistorianagent-0.1	b14a658e-3c4e-4291-8537-02ff199974e0	Never started	Start

Install agents

No files selected.

Figure 48: VMC Web Interface (Installed Agents)

8. Click the start button for the sqlhistorianagent-0.1 agent. The charts at the top of the screen should begin to populate with data as the agent becomes active (Figure 49). The SQLHistorian will publish system usage data (e.g., CPU) when it is configured with the identity **platform.historian**. The identity is set in the agent's configuration file (*config.sqlite.platform.historian* from step 7 in this section).

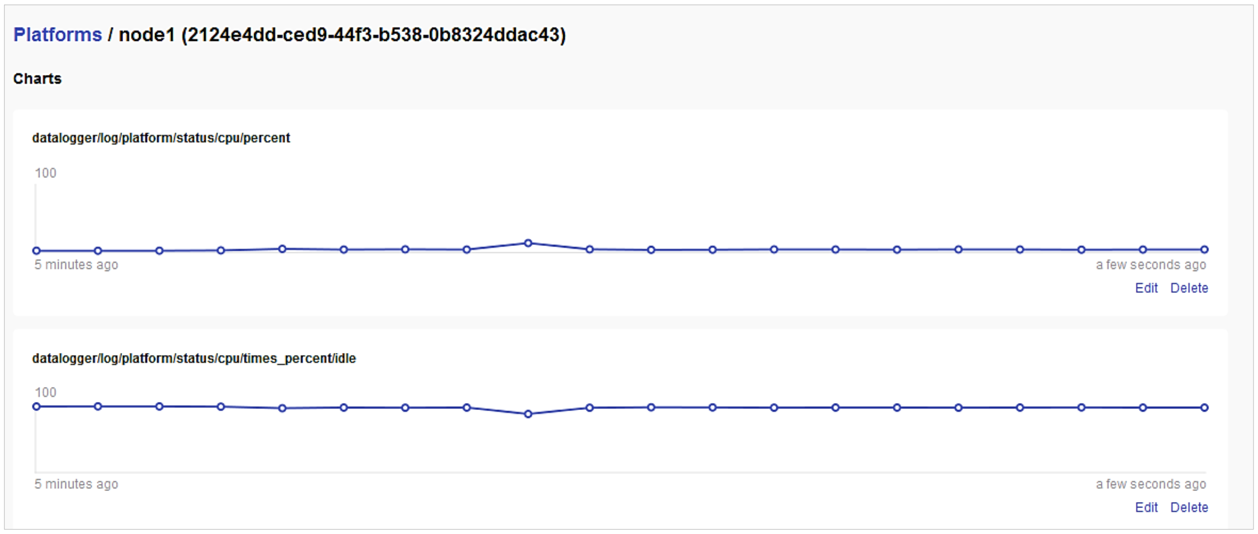

Platforms / node1 (2124e4dd-ced9-44f3-b538-0b8324ddac43)
Charts
datalogger/log/platform/status/cpu/percent
100
5 minutes ago
a few seconds ago
Edit
Delete
datalogger/log/platform/status/cpu/times_percent/idle
100
5 minutes ago
a few seconds ago
Edit
Delete

Figure 49: VMC Web Interface (System Usage Charts)

To add a chart, click the Add Chart button. The published topic that the chart pulls data from will need to be provided. The selected topic should be numeric data. One may also select the refresh interval and chart type, as well as pin the chart to the dashboard (Figure 50).

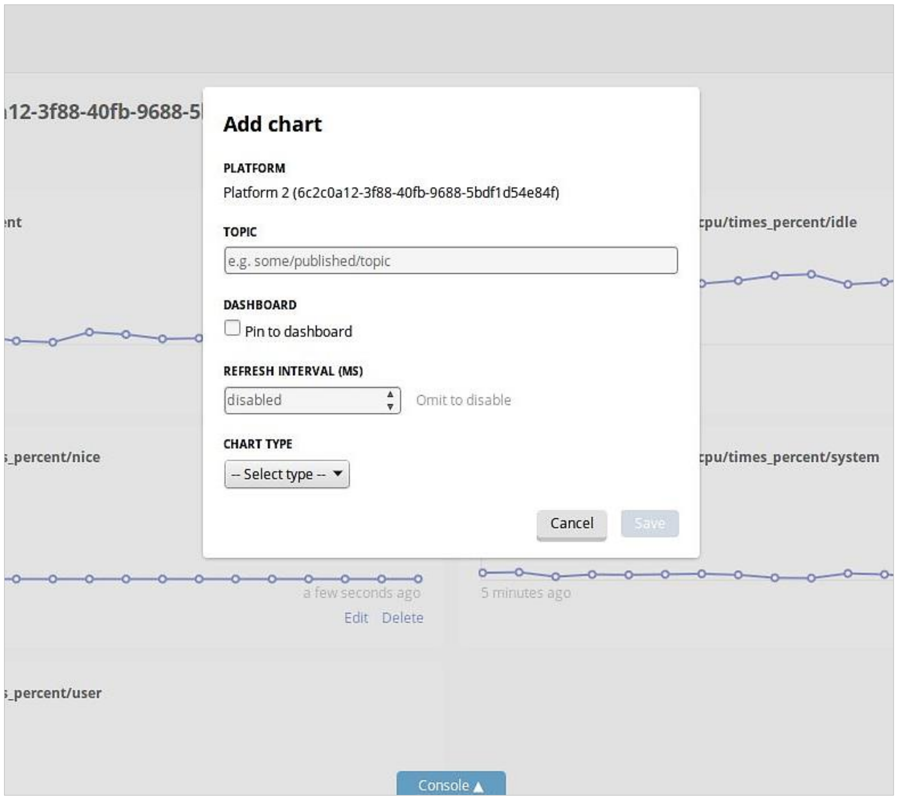


Figure 50: VMC Web Interface (Adding Charts)

A VMC demonstration script is packaged with VOLTTRON. The script highlights the key features for VMC described in this section. Information on running the management demonstration is available on the VOLTTRON Wiki:

https://github.com/VOLTTRON/VOLTTRON3.0-docs/wiki/VOLTTRON-Central-Demo

4 Sample Applications/Agents

This section summarizes the use of the sample applications that are pre-packaged with VOLTTRON. For detailed information on these applications, refer to the report Transactional Network Platform: Applications.[10]

4.1 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, and 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 will detail 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.

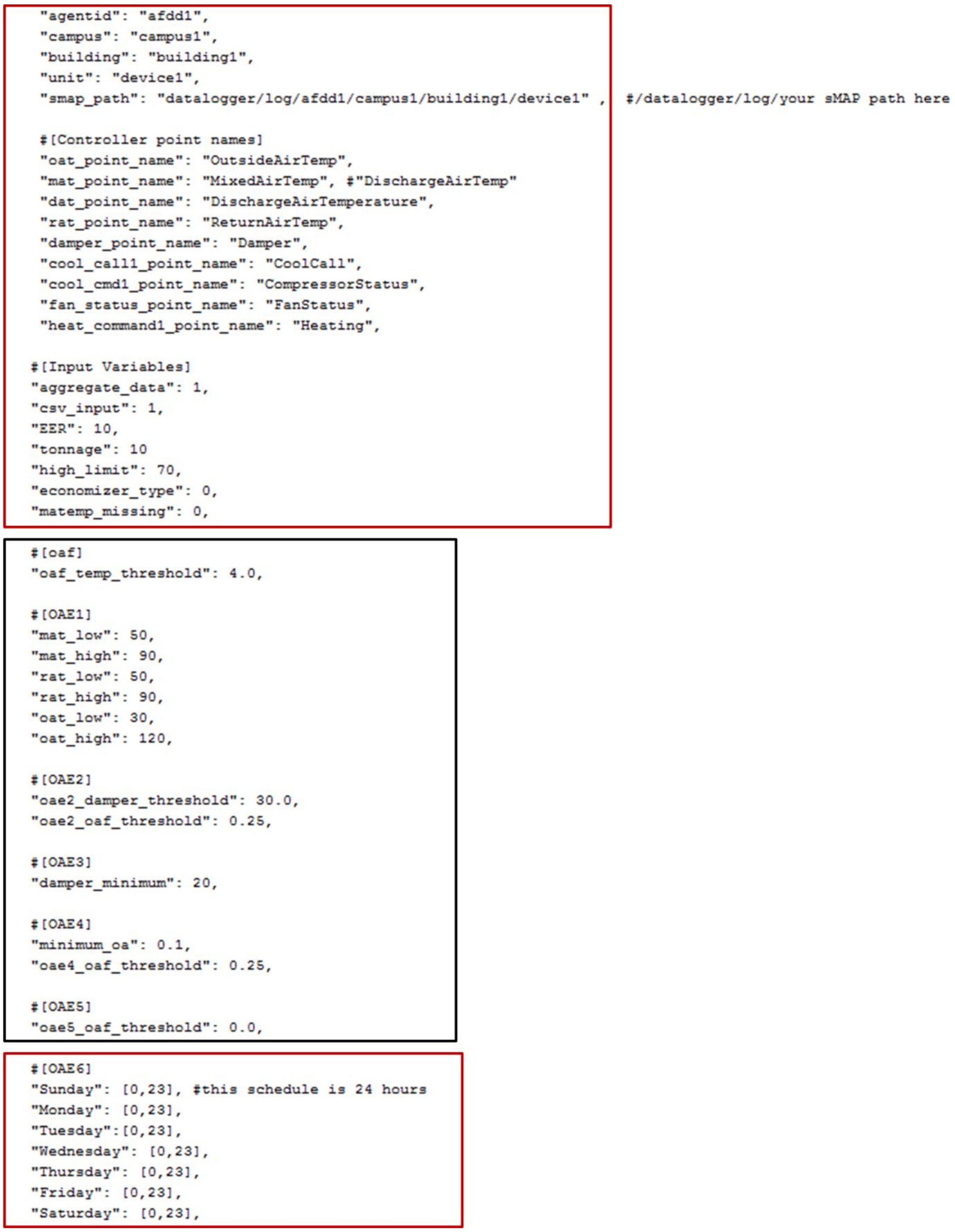
Note: A proactive version of the Passive AFDD exists as a PNNL application (AFDDAgent). This application requires active control of the RTU for fault detection and diagnostics to occur. The Passive AFDD was created to allow more users a chance to run diagnostics on their HVAC equipment without the need to actively modify the controls of the system.

4.1.1 Configuring the Passive AFDD Agent

Before launching the Passive AFDD agent, several parameters require configuration. The AFDD utilizes the same JSON style configuration file that the Actuator, Listener, and Weather agents use, which is documented in the previous sections of this document. The threshold parameters used for the fault detection algorithms are pre-configured and will work well for most RTUs or AHUs. Figure 51 shows an example configuration file for the AFDD agent.

The parameters boxed in black (in Figure 51) 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:

[10] http://www.pnl.gov/main/publications/external/technical_reports/PNNL-22941.pdf



```
 "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 51: Example Passive AFDD Agent Configuration File

- **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 Section 3 .
- **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, then this value will 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)

4.1.2 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 as follows:

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:

```
$ volttron-ctl start --tag passive-afdd
```

3. Verify that the agent is running:

```
$ volttron-ctl 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:

```
$ volttron-ctl stop --tag passive-afdd
$ volttron-ctl 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
31	Unit is not currently cooling or conditions are not favorable for economizing (No Fault)
32	Insufficient outdoor air when economizing (Fault)
33	Outdoor-air damper is not fully open when the unit should be economizing (Fault)
36	OAD is open but conditions were not favorable for OAF calculation (No Fault)
37	Missing data necessary for fault detection (No Fault)
38	OAD is open when economizing but OAF calculation led to an unexpected value (No Fault)
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 calculation (No Fault)
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 Fault)
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 calculation (No Fault)
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 calculation (No Fault)
69	Unit is off (No Fault)

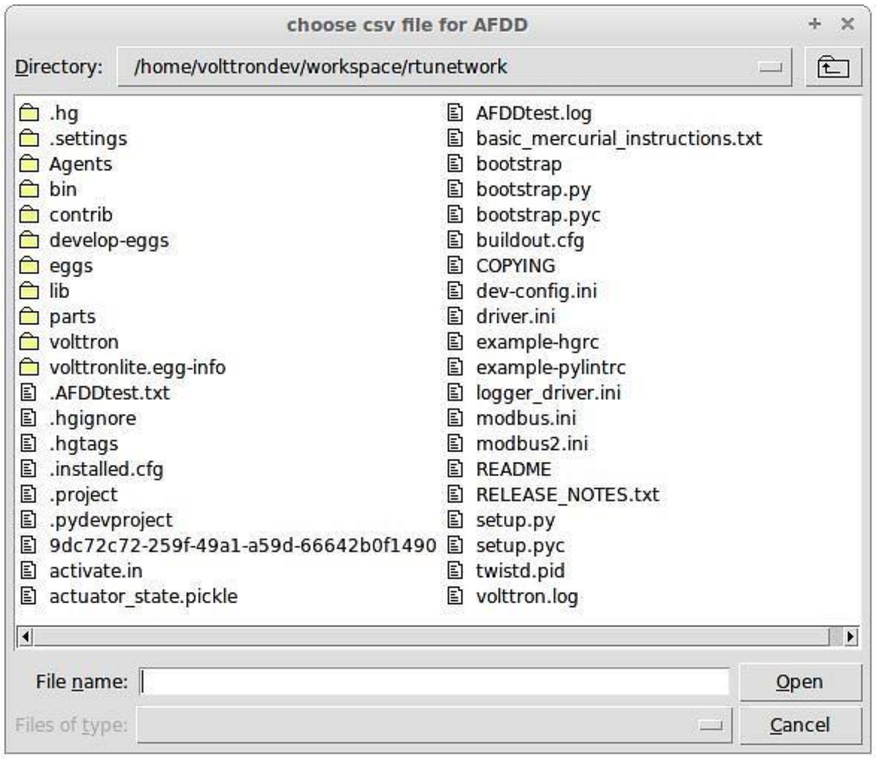
	AFDD-6 (Schedule)
70	Unit is operating correctly based on input on/off time (No Fault)
71	Unit is operating at a time designated in schedule as "off" time
77	Missing data

4.1.2.1 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, as described in Section 4.1.2.
- 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 52 shows the dialog box that is used to input the csv data file.


choose csv file for AFDD

Directory: /home/volttrondev/workspace/rtunetwork

```
.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
```

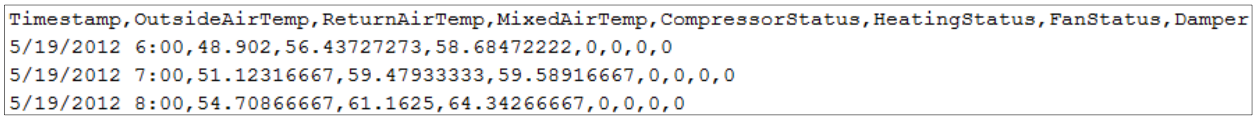
File name:

Files of type:

Figure 52: 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 53 shows a sample input data in a csv format.



```
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 53: Sample of CSV Data for Passive AFDD Agent

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 51, boxed in red.

4.2 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 5^{o}F below the normal. Rather than change the set point to a value that is 3 to 5^{o}F below the normal all at once, the set point is gradually lowered over a period of time.

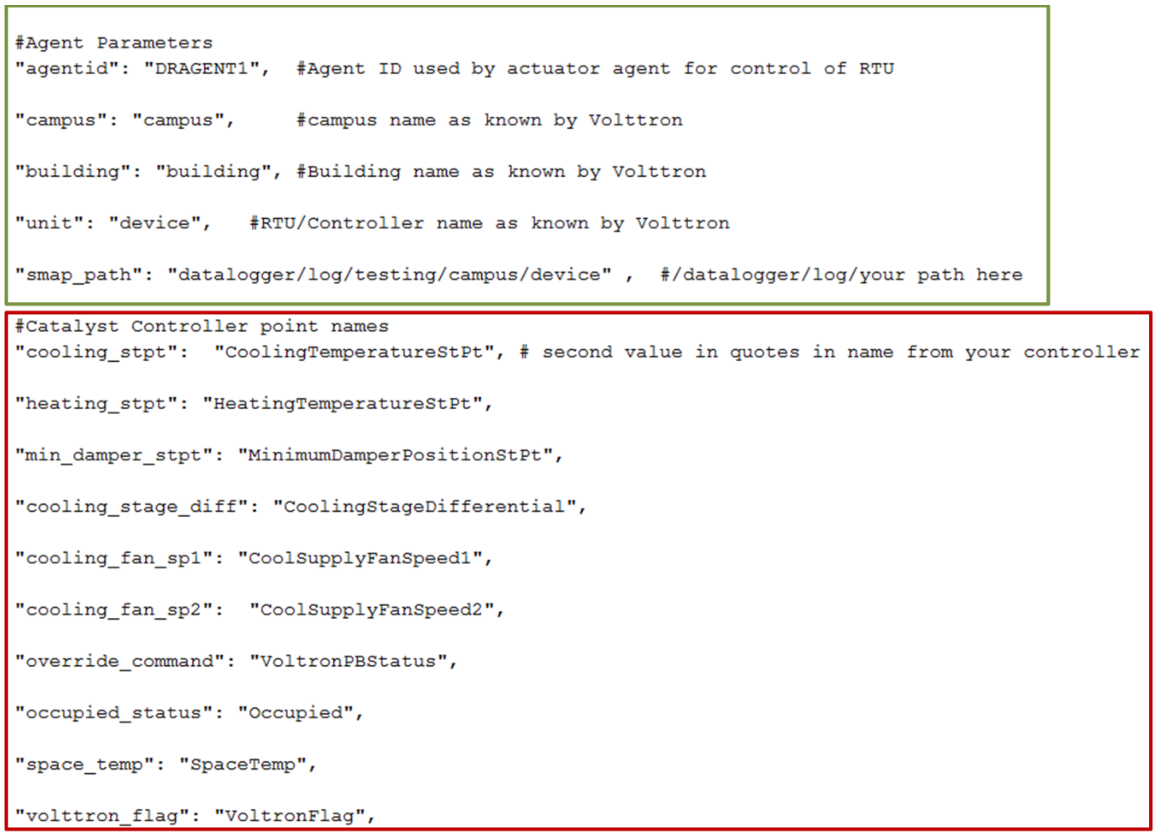
Step 2 – Event: During the CPP event, the cooling set point is raised to a value that is 4 to 5^{o}F above the normal, the damper is commanded to a position that is slightly below the normal minimum (half of the normal minimum), the fan speed is slightly reduced (by 10% to 20% of the normal speed, if 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.

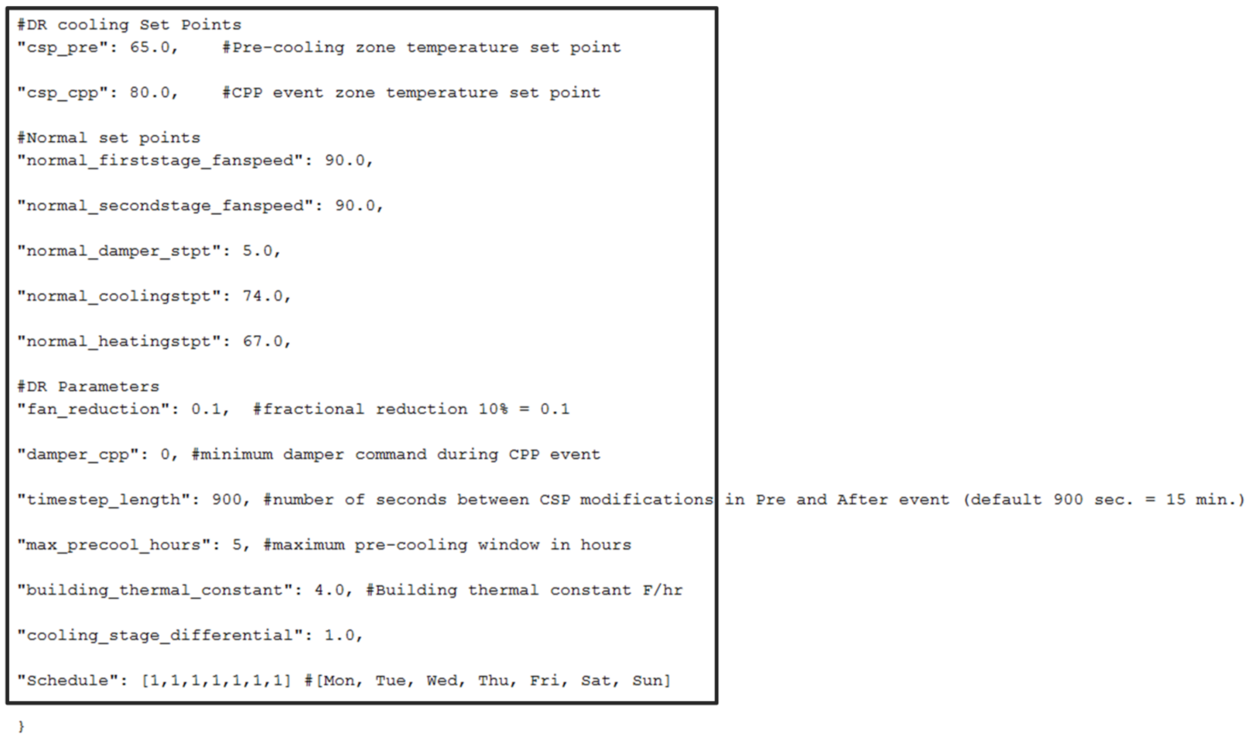
4.2.1 Configuring DR Agent

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 (Section 3.3). Figure 54 shows a sample configuration file for the DR agent:



```
#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 54: Example Configuration File for the DR Agent

The parameters boxed in black (Figure 54) 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]).

4.2.2 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 demand response 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:

```
$ . 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.

4.2.3 DR Agent Output to sMAP

The Demand Response agent will output to the sMAP location prescribed in your sMAP configuration file. The specific "branch" within this sMAP database is specified in the DR agent's configuration file. The DR agent will output the start time for the CPP event and the end time for the CPP event. These will be specified by a value of "1" for the start time and "2" for the end time. If the CPP event is cancelled or a user override is initiated, the DR agent will push a value of "3" to sMAP.

4.2.4 Launching the Demand Response Agent

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:

```
$ volttron-ctl start --tag dr-agent
```

3. Verify that agent is running:

```
$ volttron-ctl 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:

```
$ volttron-ctl stop --tag dr-agent
$ volttron-ctl remove --tag dr-agent
```

Then re-build and start the updated agent**.**

5 Agent Development in VOLTTRON

VOLTTRON supports agents written in any language. The only requirement is that they can communicate over the message bus. However, Python-based agents were the focus of the original work, and an array of utilities has been created to speed development in that language.

The **`volttron.platform.vip.agent`** package contains these utilities (`~/volttron/volttron/platform/vip/agent`). Information on using these utilities can be found on the VOLTTRON Wiki and within comments in the code. These comments contain explanations and examples of usage.

The following sections provide an example agent and then give an overview of creating a simple agent from scratch.

5.1 Example Agent Walkthrough

The Listener agent 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 because it utilizes publish and subscribe mechanisms and contains the basic structure necessary to build a very simple agent.

5.2 Explanation of Listener Agent

The Listener agent inherits from the Agent class for its base functionality. Please see (`<project directory>/volttron/platform/agent/base.py`) for the details of these classes.

```
from __future__ import absolute_import
from datetime import datetime
import logging
import sys

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, topics
from . import settings
# If developing inside Eclipse, use pydev-launch.py
# or change this to: import settings
```

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 the agents `__init__` method:

```
def __init__(self, config_path, **kwargs):
    super(ListenerAgent, self).__init__(**kwargs)
    self.config = utils.load_config(config_path)
```

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 setup(self, sender, **kwargs):
    # Demonstrate accessing a value from the config file
    _log.info(self.config['message'])
    self._agent_id = self.config['agentid']
```

The Listener agent subscribes to all topics published on the message bus. Subscribe/publish interactions with the message bus are handled by the PubSub module located at:

- ➢ `<project directory>/volttron/platform/vip/agent/subsystems/pubsub.py`

The Listener agent uses an empty string to subscribe to all messages published. It checks for the sender being pubsub.compat in case there are any VOLTTRON 2.0 agents running on the platform.

```
@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)
    _log.debug(
        "Peer: %r, Sender: %r:, Bus: %r, Topic: %r, Headers: %r, "
        "Message: %r", peer, sender, bus, topic, headers, message)
```

The Listener agent uses the `@Core.periodic` decorator to execute the **`publish_heartbeat`** method every **HEARTBEAT_PERIOD** seconds where **HEARTBEAT_PERIOD** is specified in the *settings.py* file:

```
# Demonstrate periodic decorator and settings access
@Core.periodic(settings.HEARTBEAT_PERIOD)
def publish_heartbeat(self):
    '''Send heartbeat message every HEARTBEAT_PERIOD seconds.

    HEARTBEAT_PERIOD is set and can be adjusted in the settings module.
    '''
    now = datetime.utcnow().isoformat(' ') + 'Z'
    headers = {
        'AgentID': self._agent_id,
        headers_mod.CONTENT_TYPE: headers_mod.CONTENT_TYPE.PLAIN_TEXT,
        headers_mod.DATE: now,
    }
    self.vip.pubsub.publish(
        'pubsub', 'heartbeat/listeneragent', headers, now)
```

The **`vip.pubsub.publish`** method is called with the following arguments:

- pubsub **-** Always the first argument to **`vip.pubsub.publish`**.

- topic **-** Topic of message or data. In this example the Listener is publishing to the "heartbeat/listeneragent" topic
- headers **-** Contains information such as date published, content type (plain text, utf-8, asci, etc), and the agent ID for publishing agent.
- message **-** Can be a single value (float, integer, or string), list of objects, or dictionary. Agents subscribing to the message should know the format of message so they can parse and use the information. In this example the Listener agent is publishing a string message containing the current time ("now") in UTC ISO format (note: Python datetime objects should be converted to strings before passing to **`vip.pubsub.publish`**).

To see the Listener agent in action, please see Section 2.9.

5.3 Explanation of SimpleForwarder Agent

Communication between platforms can now be accomplished using VIP. The following example illustrates setting up of an agent to subscribe to a topic(s) on a local message bus and send that to a remote VOLTTRON node using VIP.

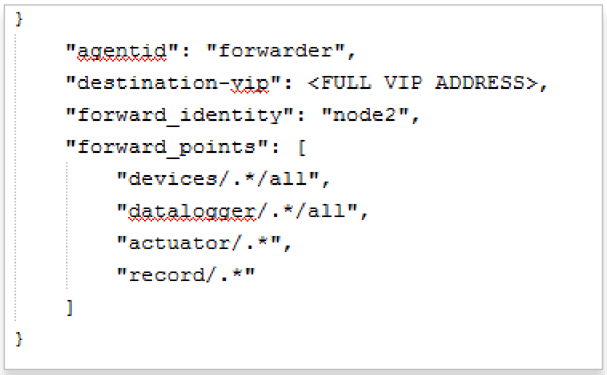
The following are the imports for the SimpleForwarder agent. The top block is Python system libraries and the bottom block of imports are the VOLTTRON imports.

```
import logging
import re
import sys
import gevent
from gevent.core import callback
from __builtin__ import list

from volttron.platform.vip.agent import Core, Agent
from volttron.platform.agent import utils
from volttron.platform.messaging import topics, headers as headers_mod
```

Next, the agent loads and reads the configuration file.

```
def simpleforwarder(config_path, **kwargs):
    # load configuration file specified during agent launch.
    config = utils.load_config(config_path)

    # read destination VIP address from config file.
    destination_vip = config.get('destination-vip')
    identity = config.get('identity', kwargs.pop('identity', None))

    # read list of topics to publish to remote message bus.
    forward_identity = config.get('forward_identity', None)
    forward_points = config.get('forward_points', [])
    point_ex = [re.compile(v) for v in forward_points]
    pub_list = \
        [item.split('/')[0] if '/' in item else item for item in forward_points]
     assert point_ex
    assert destination_vip
```

Figure 55 shows the SimpleForwarder agent configuration file:



```
}
    "agentid": "forwarder",
    "destination-vip": <FULL VIP ADDRESS>,
    "forward_identity": "node2",
    "forward_points": [
        "devices/.*/all",
        "datalogger/.*/all",
        "actuator/.*",
        "record/.*"
    ]
}
```

Figure 55: SimpleForwarder Agent Configuration File

The SimpleForwarder agent configuration file is located at:

➢ `<project directory>/examples/SimpleForwarder/forwarder.config`

The configuration file contains the following arguments:

- **agentid -** Identification tag for agent (typically for interaction with platform and other agents).
- **topic -** Full VIP address for the destination platform. Should appear similar to (see Section 3.5.1):
    - "tcp://192.168.1.213.116.213:29216?serverkey=RcbUt99IJk3J1nD101WfRrk7VYOwOICAtfHwJrZw1zs&publickey=6ZFlyV9w_RppG743NGM1ajFyiC3IMA_aCmacER_8OzE&secretkey=qTKwxXcUHPFCoSF2Fxf5iQyNOJTAg2IS4lOFnfbPvoA"
- **forward_identity -** Identification tag for agent on remote platform.
- **forward_points -** List of topics to forward to remote the platform. This can be formatted as regular expressions and will match accordingly.

Next, the SimpleForwarder agent will run its startup method. The startup method is tagged with the decorator `@Core.receiver('onstart').`

```
@Core.receiver('onstart')
def starting(self, sender, **kwargs):
    '''
    Subscribes to the platform message bus on the actuator, record,
    datalogger, and device topics to capture data.
    '''
    _log.info('Starting forwarder to {}'.format(destination_vip))

    # instantiate an Agent that will publish to the remote message bus.
    agent = Agent(identity=forward_identity, address=destination_vip)
    event = gevent.event.Event()

    # agent.core.run set the event flag to true when agent is running.
    # spawn a greenlet thread for Agent.
    gevent.spawn(agent.core.run, event)

    # Wait until the agent is fully initialized and ready to
    # send and receive messages.
    event.wait()

    self._target_platform = agent

    # subscribe to configured topics
    for item in pub_list:
        self.vip.pubsub.subscribe(peer='pubsub', prefix=item,
                                  callback=self.data_received)
```

The startup method will instantiate an Agent instance to publish to the remote platform (one Agent instance is needed for each additional VIP address published to). The "for" loop at the end of the function setups a function callback for each topic in the agent configuration file (**forward_points** entry in *forwarder.config*).

The **`data_received`** method will check if received topic matches the topics set in the agent configuration file:

```
def data_received(self, peer, sender, bus, topic, headers, message):
    '''Callback function to publish data to remote platform.'''
    def publish_external(agent, topic, headers, message):
        '''On matching captured topic publish message to remote platform.'''
        try:
            _log.debug(
                'Attempting to publish remotely {}, {}, {}'.format(topic,
                                                                    headers,
                                                                    message))

                'Attempting to publish '
                'remotely {}, {}, {}'.format(topic, headers, message))
            # Publish to remote platform by agent (additionally instantiated
            # Agent).
            agent.vip.pubsub.publish(peer='pubsub',
                                     topic=topic,
                                     headers=headers,
                                     message=message).get(timeout=30)
        except:
            _log.debug('Data dropped {}, {}, {}'.format(topic,
                                                        headers,
                                                        message))

    if sender == 'pubsub.compat':
        message = jsonapi.loads(message[0])
        del(headers[headers_mod.CONTENT_TYPE])
        assert isinstance(message, list)
        assert isinstance(message[0], dict)
        assert isinstance(message[1], dict)
        print("MESSAGE VALUES ARE: {}".format(message[0]))
        print("DATA VALUES ARE: {}".format(message[1]))
        for v in message[1].values():
            assert 'tz' in v.keys()
            assert 'units' in v.keys()
            assert 'type' in v.keys()

    # check if topic matches topics set up in configuration file.
    for rex in point_ex:
        if rex.match(topic):
            publish_external(self._target_platform, topic, headers, message)
            break
        else:
            _log.info("Topic does not match: {}".format(topic))
```

The following excerpt from the **received_data** method checks if the captured topic matches any of the topics in the agent configuration file. If the captured topic does not match, the agent creates a log entry with this information. If the topic does match, the agent calls the **publish_external** function.

```
# check if topic matches topics set up in configuration file.
for rex in point_ex:
    if rex.match(topic):
        publish_external(self._target_platform, topic, headers, message)
        break
    else:
        _log.info("Topic does not matchNot a matching topic: {}".format(topic))
```

The nested function **publish_external** will call the agent's (the additional Agent instance) **vip.pubsub.publish** method to forward (publish) the message to the remote platform.

The **shutdown** method for the SimpleForwarder agent unsubscribes from any currently held subscriptions (This agent created the subscriptions in the startup method):

```
@Core.receiver("onstop")
def stopping(self, sender, **kwargs):
    '''
    Release subscription to the message bus on shutdown.
    '''
    try:
        # unsubscribes to all topics that we are subscribed to.
        self.vip.pubsub.unsubscribe(peer='pubsub',
                                    prefix=None,
                                    callback=None)
    except KeyError:
        # means that the agent didn't start up properly so the pubsub
        # subscriptions never got finished.
        pass
```

The following is the SimpleForwarder agent's code in its entirety:

```
from __future__ import absolute_import, print_function
import logging
import re
import sys

import gevent
from gevent.core import callback
from __builtin__ import list

from volttron.platform.vip.agent import Core, Agent
```

```python
from volttron.platform.agent import utils
from volttron.platform.messaging import topics, headers as headers_mod


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


def simpleforwarder(config_path, **kwargs):
    config = utils.load_config(config_path)
    destination_vip = config.get('destination-vip')
    identity = config.get('identity', kwargs.pop('identity', None))
    forward_identity = config.get('forward_identity', None)
    forward_points = config.get('forward_points', [])
    point_ex = [re.compile(v) for v in forward_points]
    pub_list = \
        [item.split('/')[0] if '/' in item else item for item in forward_points]

    assert destination_vip
    assert point_ex

    class SimpleForwarder(Agent):
        '''This is a simple example of a historian agent that writes stuff
        to a SQLite database. It is designed to test some of the functionality
        of the BaseHistorian agent.
        '''

        @Core.receiver("onstart")
        def starting(self, sender, **kwargs):
            '''
            Subscribes to the platform message bus on the actuator, record,
            datalogger, and device topics to capture data.
            '''
            _log.info('Starting forwarder to {}'.format(destination_vip))

            agent = Agent(identity=forward_identity, address=destination_vip)
            event = gevent.event.Event()

            # agent.core.run set the event flag to true when agent is running
            gevent.spawn(agent.core.run, event)

            # Wait until the agent is fully initialized and ready to
            # send and receive messages.
            event.wait()
            self._target_platform = agent

            # subscribe to configured topics
            for item in pub_list:
                self.vip.pubsub.subscribe(peer='pubsub', prefix=item,
                                          callback=self.data_received)

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

            def publish_external(agent, topic, headers, message):
                '''Callback function to publish data to remote platform.'''
                try:
                    _log.debug(
                        'Attempting to publish '
                        'remotely {}, {}, {}'.format(topic, headers, message))
                    agent.vip.pubsub.publish(peer='pubsub',
                                             topic=topic,
                                             headers=headers,
                                             message=message).get(timeout=30)
                except:
                    _log.debug('Data dropped {}, {}, {}'.format(topic,
                                                                headers,
                                                                message))

            if sender == 'pubsub.compat':
                message = utils.jsonapi.loads(message[0])
                del(headers[headers_mod.CONTENT_TYPE])
                assert isinstance(message, list)
                assert isinstance(message[0], dict)
                assert isinstance(message[1], dict)
                print("MESSAGE VALUES ARE: {}".format(message[0]))
                print("DATA VALUES ARE: {}".format(message[1]))
                for v in message[1].values():
                    assert 'tz' in v.keys()
                    assert 'units' in v.keys()
                    assert 'type' in v.keys()

            for rex in point_ex:
                if rex.match(topic):
                    publish_external(self._target_platform, topic, headers, message)
            else:
                publish_external(self._target_platform, topic, headers, message)

        @Core.receiver("onstop")
        def stopping(self, sender, **kwargs):
            '''
            Release subscription to the message bus on shutdown
            '''
            try:
                # unsubscribes to all topics that we are subscribed to.
                self.vip.pubsub.unsubscribe(peer='pubsub',
                                            prefix=None,
                                                callback=None)
            except KeyError:
                # means that the agent didn't start up properly so the pubsub
                # subscriptions never got finished.
                pass
    SimpleForwarder.__name__ = 'SimpleForwarder'
    return SimpleForwarder(**kwargs)


def main(argv=sys.argv):
    '''Main method called by the eggsecutable.'''
    try:
        utils.vip_main(simpleforwarder)
    except Exception as e:
        print(e)
        _log.exception('unhandled exception')


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

5.4 Remote Procedure Protocol: How to and Examples

Remote procedure calls (RPC) is a new feature added with VOLTTRON 3.0. VIP introduced the ability to create new point-to-point protocols, called subsystems, enabling the implementation of JSON-RPC 2.0[11]. 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.

5.4.1 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(method, name=None)
```

```
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,)

    @RPC.export('say_bye')
    def bye(self, name):
        '''Build and return a goodbye string'''
        return 'Goodbye, %s.' % (name,)
```

[11] http://www.jsonrpc.org/specification

```
    @Core.receiver('setup')
    def onsetup(self, sender, **kwargs):
        self.vip.rpc.export('add')
```

5.4.2 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 the peer with the given arguments. Returns a gevent AsyncResult object.

```
RPC.notify(peer, method, *args, **kwargs)
```

Send a one-way notification message to peer by calling method without returning a result:

```
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')
```

5.4.3 Inspecting Exported Methods

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
```

Additional examples at:

https://github.com/VOLTTRON/VOLTTRON3.0-docs/wiki/RPC-by-example

5.5 Agent Development in Eclipse

The Eclipse IDE (integrated development environment) is not required for agent development, but it can be a powerful developmental tool. For those wishing to use it, download the IDE from:

For 32-bit machines:
http://www.eclipse.org/downloads/download.php?file=/technology/epp/downloads/release/mars/R/eclipse-java-mars-R-linux-gtk.tar.gz

For 64-bit machines:
http://www.eclipse.org/downloads/download.php?file=/technology/epp/downloads/release/mars/R/eclipse-java-mars-R-linux-gtk-x86_64.tar.gz

The previous links will take you to the Eclipse download page. Choose a download mirror closest to your location.

This link will take you to the main Eclipse webpage:

http://www.eclipse.org/

5.5.1 Installing Eclipse

To install Eclipse, enter the following commands in a terminal:

1. Install Eclipse dependency:

```
# apt-get install openjdk-7-jdk
```

2. After downloading the eclipse archive file, move the package to the opt directory (enter this command from a terminal in the directory where eclipse was downloaded):

```
$ tar -xvf eclipse-java-mars-R-linux-gtk-x86_64.tar.gz
# mv eclipse /opt/
```

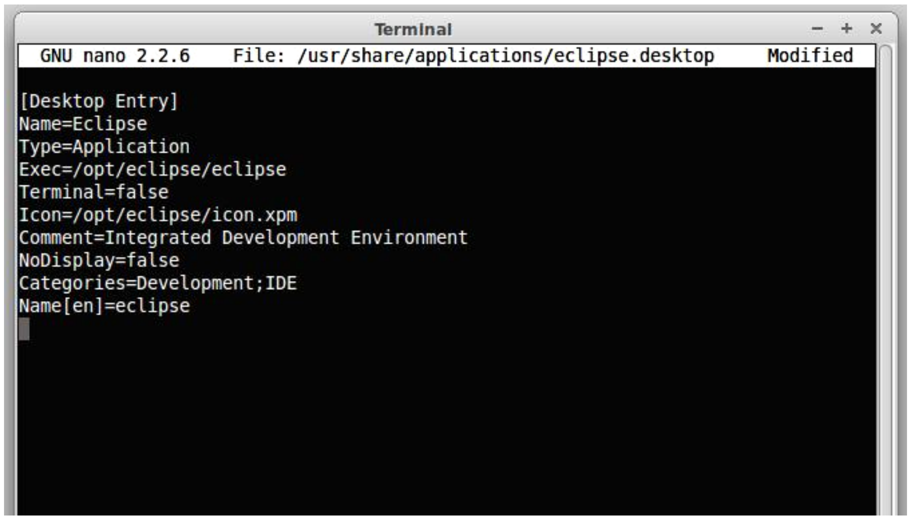
   - For 32-bit machines, replace “gtk-x86_64” with “linux-gtk” in the previous command.

3. Create desktop shortcut:

```
# touch /usr/share/applications/eclipse.desktop
# nano /usr/share/applications/eclipse.desktop
```

Enter the following text, as shown in Figure 56, and save the file. To avoid typos, copy and paste the following:

```
[Desktop Entry]
Name=Eclipse
Type=Application
Exec=/opt/eclipse/eclipse
Terminal=false
Icon=/opt/eclipse/icon.xpm
Comment=Integrated Development Environment
NoDisplay=false
Categories=Development;IDE
Name[en]=eclipse
```



```
  GNU nano 2.2.6     File: /usr/share/applications/eclipse.desktop       Modified

[Desktop Entry]
Name=Eclipse
Type=Application
Exec=/opt/eclipse/eclipse
Terminal=false
Icon=/opt/eclipse/icon.xpm
Comment=Integrated Development Environment
NoDisplay=false
Categories=Development;IDE
Name[en]=eclipse
```

Figure 56: Eclipse Desktop File

4. Copy the shortcut to the desktop:

```
$ cp /usr/share/applications/eclipse.desktop  ~/Desktop/
```

Eclipse is now installed and ready to use.

5.5.2 Installing Pydev and EGit Eclipse Plug-ins

The transactional network code is stored in a Git repository. There is a plug-in available for Eclipse that makes development more convenient (note: you must have Git installed on the system and have built the project).

- As shown in Figure 57:
    - Select: Help -> Install New Software

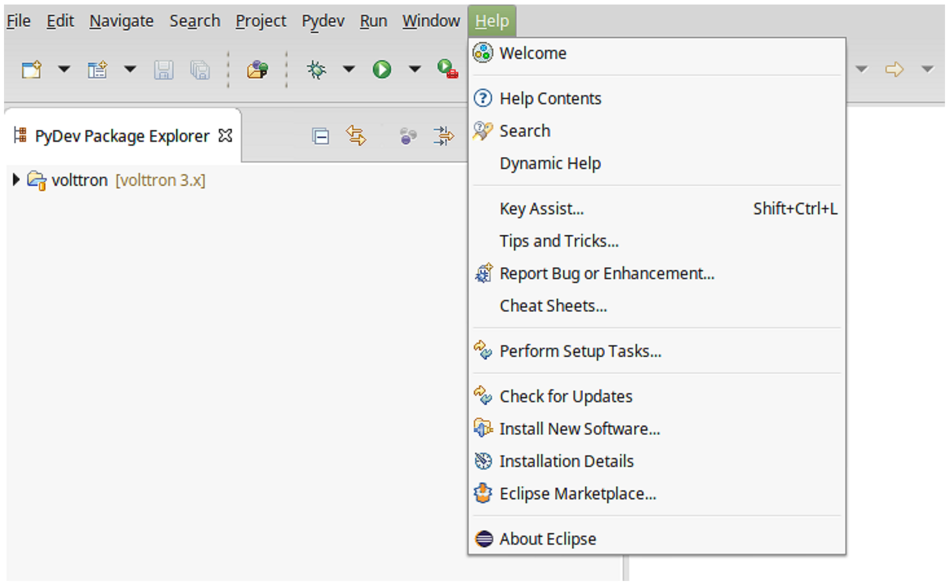


Figure 57: Installing Eclipse EGit Plug-in

- Click on the "Add" button, as shown in Figure 58.

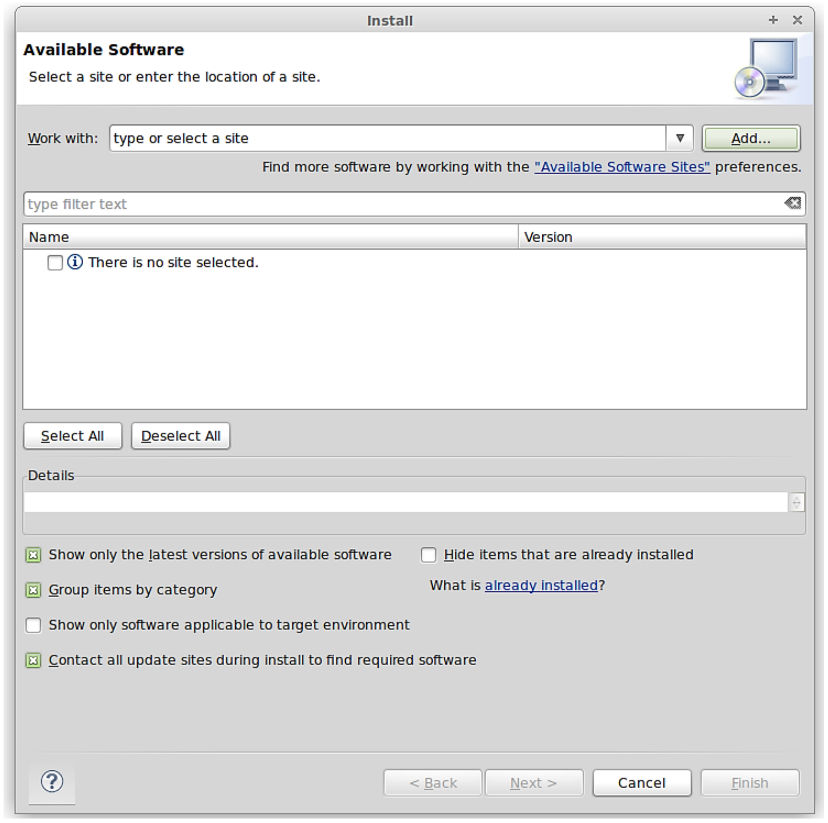
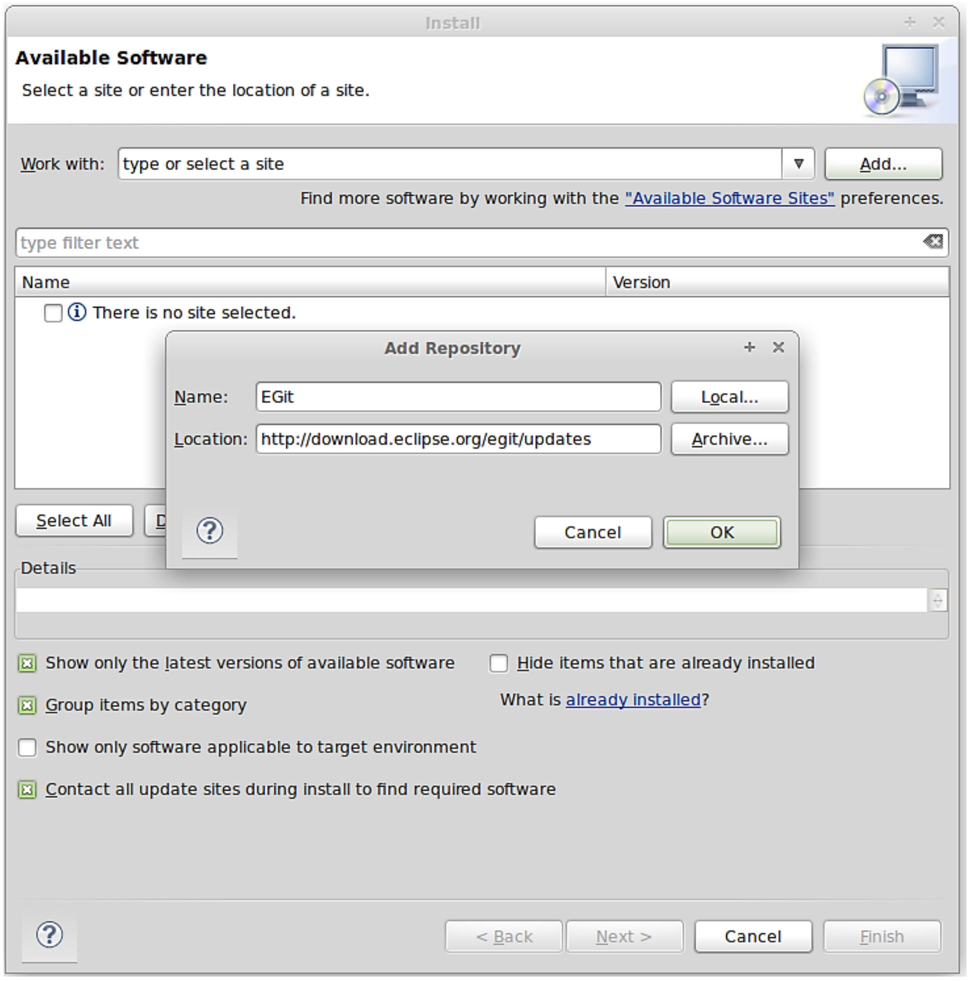

Install
Available Software
Select a site or enter the location of a site.
Work with:
type or select a site
Add...
Find more software by working with the "Available Software Sites" preferences.
type filter text
Name
Version
There is no site selected.
Select All
Deselect All
Details
Show only the latest versions of available software
Hide items that are already installed
Group items by category
What is already installed?
Show only software applicable to target environment
Contact all update sites during install to find required software
< Back
Next >
Cancel
Finish

Figure 58: Installing Eclipse Egit Plug-in (continued)

- As shown in Figure 59, enter the following:
    - For name use: EGit
    - For location: http://download.eclipse.org/egit/updates


Install

Available Software

Select a site or enter the location of a site.

Work with: type or select a site

Find more software by working with the "Available Software Sites" preferences.

type filter text

Name	Version
There is no site selected.	

Add Repository

Name: EGit

Location: http://download.eclipse.org/egit/updates

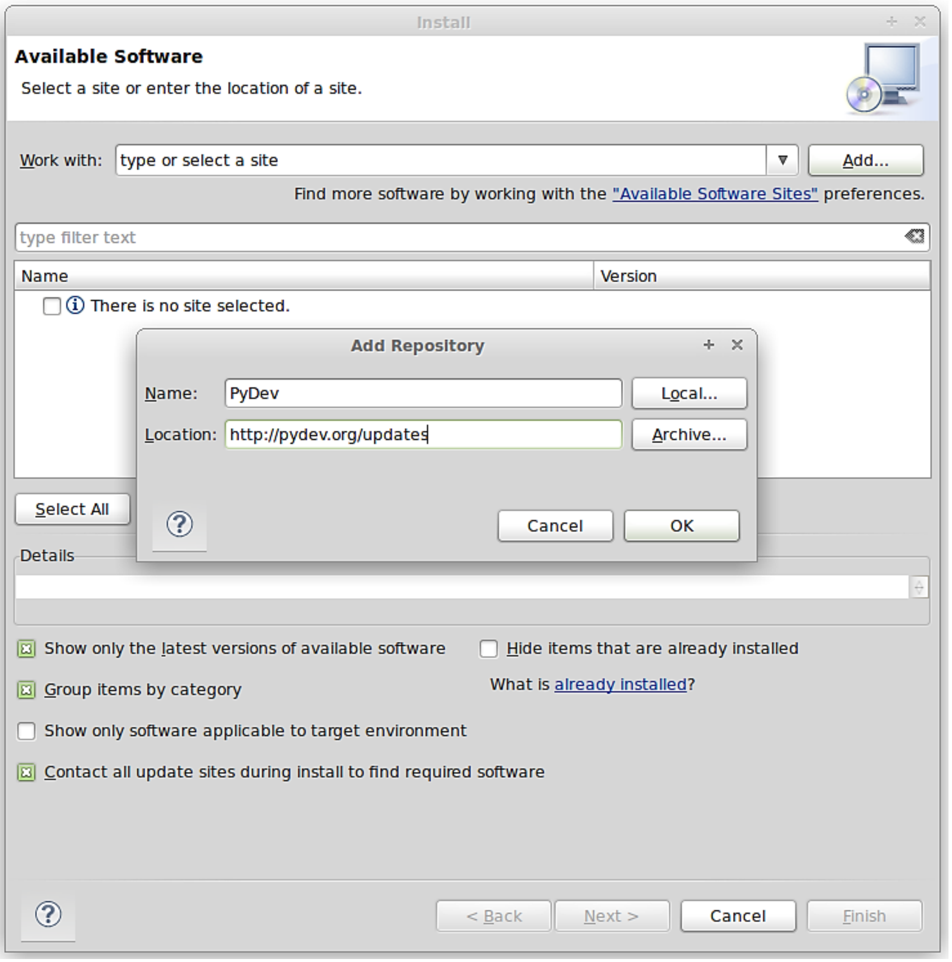
[✓] Show only the latest versions of available software
[ ] Hide items that are already installed
[✓] Group items by category
What is already installed?
[ ] Show only software applicable to target environment
[✓] Contact all update sites during install to find required software

Figure 59: Installing Eclipse Egit Plug-in (continued)

- After hitting OK, check the Select All button
- Click through Next ->Agree to Terms ->Finish
- Allow Eclipse to restart

After installing Eclipse, you must add the PyDev plug-in to the environment. In Eclipse:

- Help -> Install New Software
- Click on the Add button
- As shown in Figure 60, enter the following:
    - For name use: PyDev
    - For location: http://pydev.org/updates
    - Click OK


Install

Available Software

Select a site or enter the location of a site.

Work with: type or select a site

Find more software by working with the "Available Software Sites" preferences.

type filter text

Name	Version
☐ There is no site selected.	

Add Repository

Name: PyDev

Location: http://pydev.org/updates

☒ Show only the latest versions of available software

☐ Hide items that are already installed

☒ Group items by category

What is already installed?

☐ Show only software applicable to target environment

☒ Contact all update sites during install to find required software

Figure 60: Installing Eclipse PyDev Plug-in

1. Check the box for PyDev
2. Click through Next, Agree to Terms, Finish
3. Allow Eclipse to restart

5.5.3 Checkout VOLTTRON Project

VOLTTRON can be imported into Eclipse from an existing VOLTTRON project (VOLTTRON was previously checked out from GitHub) or a new download from GitHub.

5.5.3.1 Import VOLTTRON into Eclipse from an Existing Local Repository (Previously Downloaded VOLTTRON Project)

To import an existing VOLTTRON project into Eclipse, the following steps should be followed:

1. Select File, then import, as shown in Figure 61

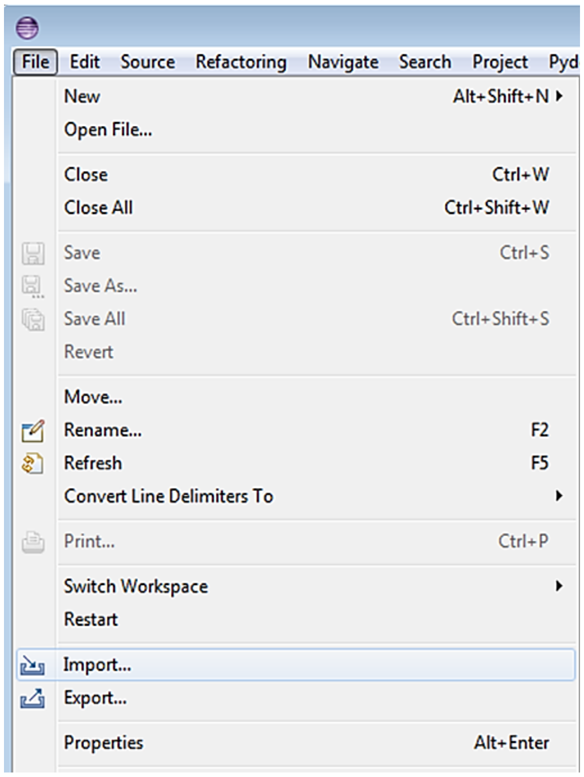


Figure 61: Checking VOLTTRON with Eclipse from Local Source

2. Select Git -> Projects from Git, then click the Next button (Figure 62)

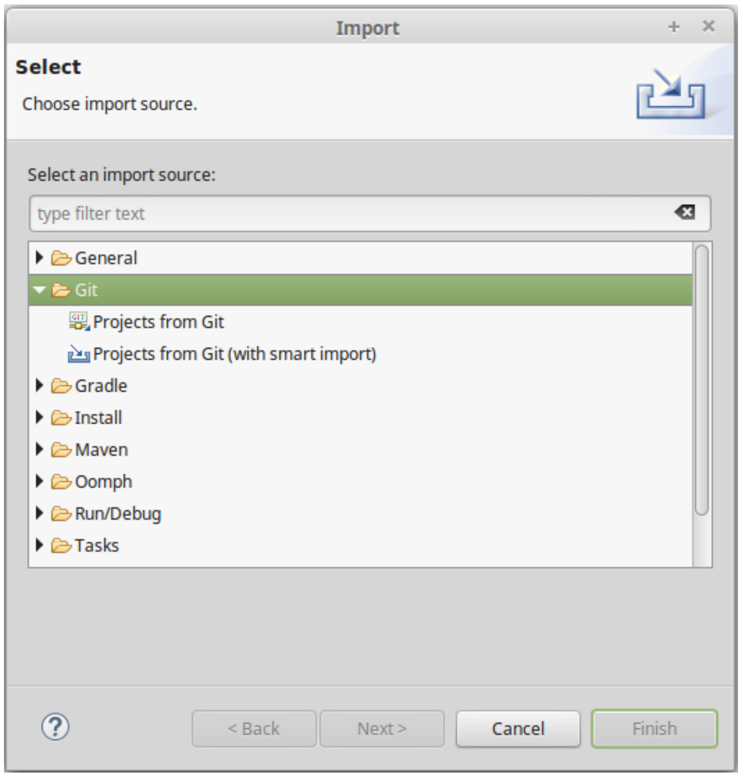


Figure 62: Checking VOLTTRON with Eclipse from Local Source (continued)

3. As shown in Figure 63:
    - Select Existing local repository -> Next >

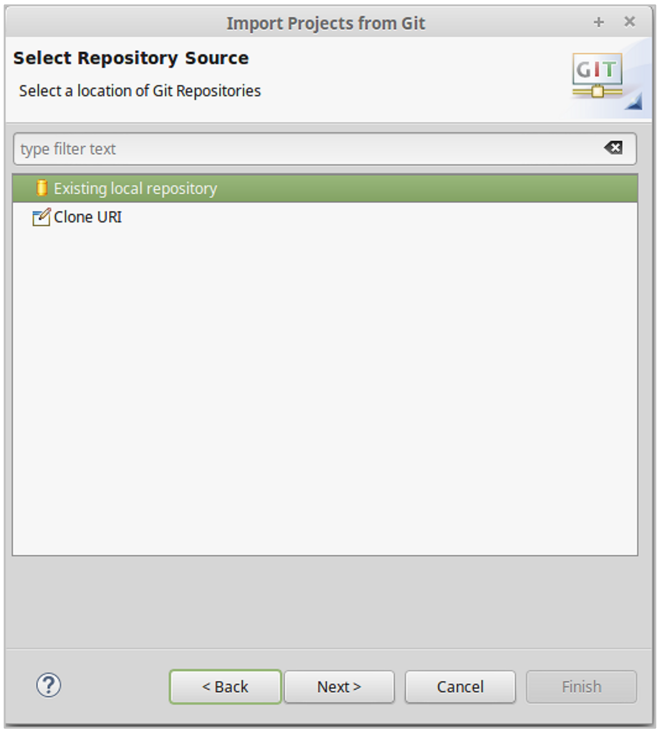


Figure 63: Checking VOLTTRON with Eclipse from Local Source (continued)

4. Select Add (Figure 64)

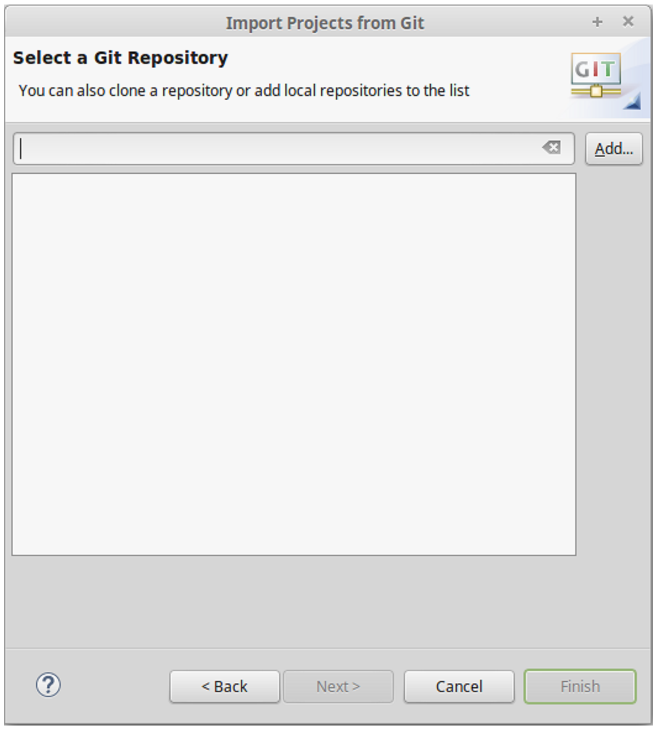
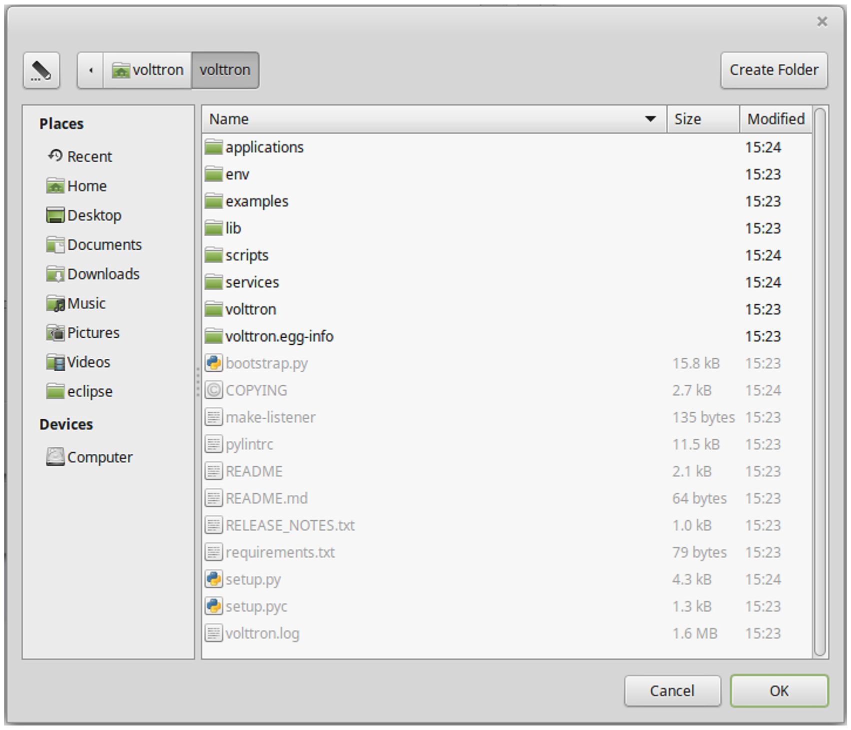


Figure 64: Checking VOLTTRON with Eclipse from Local Source (continued)

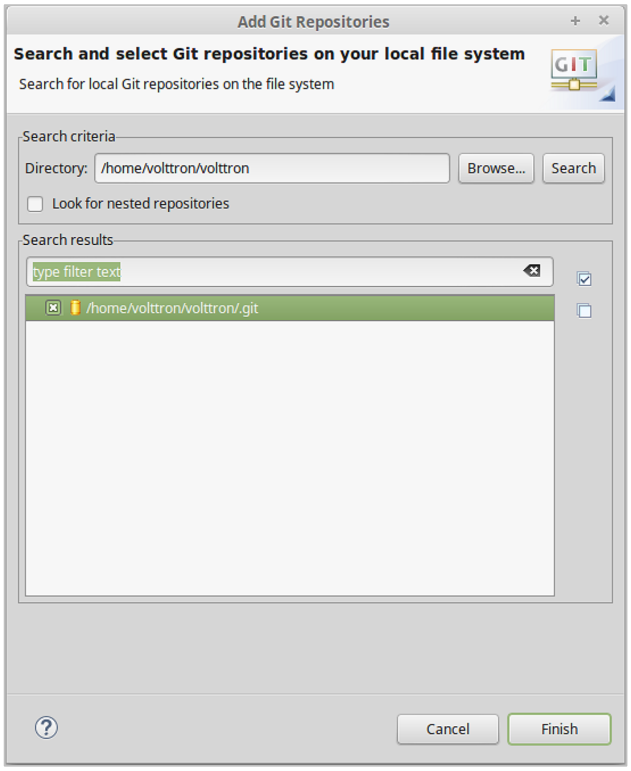
5. Select Browse -> navigate to the top-level base VOLTTRON directory and select OK (Figure 65)



Name	Size	Modified
applications		15:24
env		15:23
examples		15:23
lib		15:23
scripts		15:24
services		15:24
volttron		15:23
volttron.egg-info		15:23
bootstrap.py	15.8 kB	15:23
COPYING	2.7 kB	15:24
make-listener	135 bytes	15:23
pylintrc	11.5 kB	15:23
README	2.1 kB	15:23
README.md	64 bytes	15:23
RELEASE_NOTES.txt	1.0 kB	15:23
requirements.txt	79 bytes	15:23
setup.py	4.3 kB	15:24
setup.pyc	1.3 kB	15:23
volttron.log	1.6 MB	15:23

Figure 65: Checking Out VOLTTRON with Eclipse from Local Source (continued)

6. Select Finish (Figure 66)


Add Git Repositories

Search and select Git repositories on your local file system

Search for local Git repositories on the file system

Search criteria

Directory: /home/volttron/volttron

Look for nested repositories

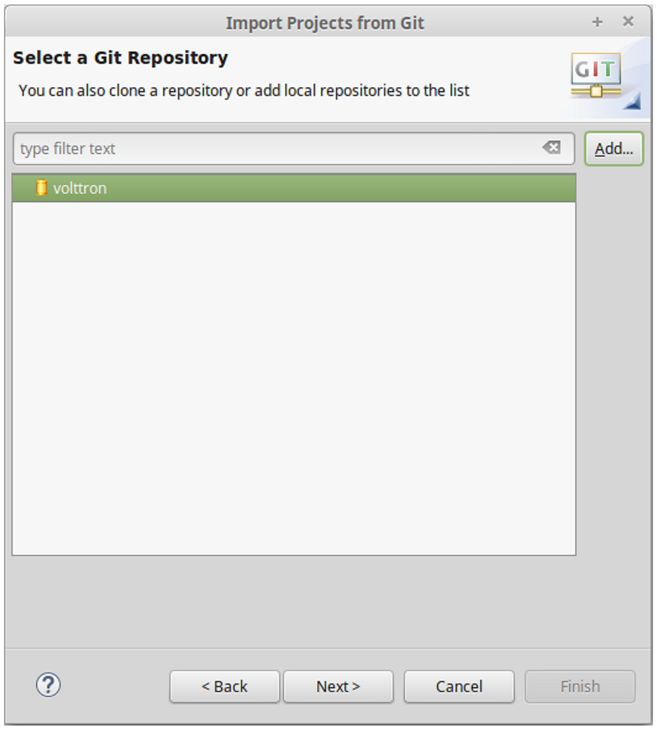
Search results

type filter text

/home/volttron/volttron/.git

Figure 66: Checking Out VOLTTRON with Eclipse from Local Source (continued)

7. Choose Next > (Figure 67).


Import Projects from Git

Select a Git Repository

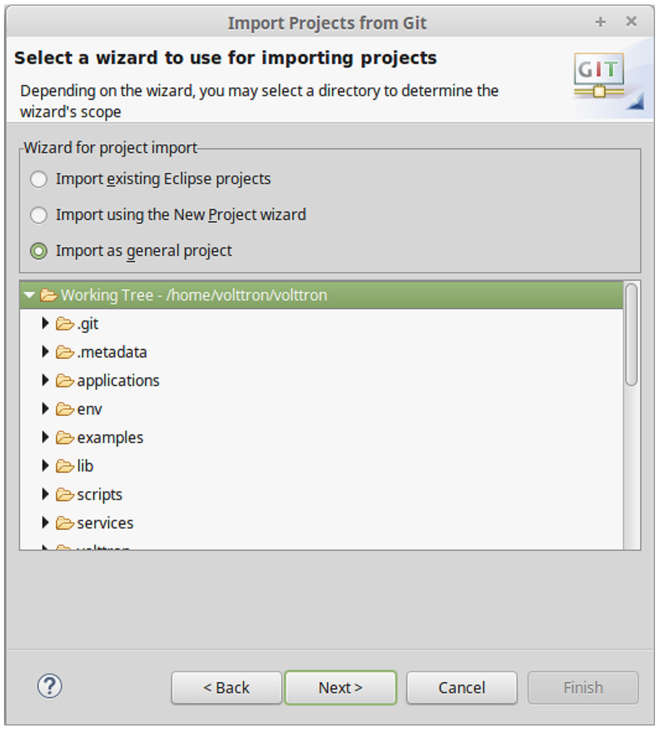
You can also clone a repository or add local repositories to the list

type filter text

volttron

Figure 67: Checking Out VOLTTRON with Eclipse from Local Source (continued)

8. Choose Import as general project and click Next -> Finish, the project will be imported into the workspace (Figure 68)


Import Projects from Git

Select a wizard to use for importing projects

Depending on the wizard, you may select a directory to determine the wizard's scope

Wizard for project import

- Import existing Eclipse projects
- Import using the New Project wizard
- [✓ Import as general project]

```
Working Tree - /home/volttron/volttron
  .git
  .metadata
  applications
  env
  examples
  lib
  scripts
  services
```

Figure 68: Checking Out VOLTTRON with Eclipse from Local Source (continued)

5.5.3.2 Import New VOLTTRON Project from GitHub

To import a new VOLTTRON project directly from GitHub into Eclipse, the following steps should be followed.

1. Select File, then Import (Figure 69)

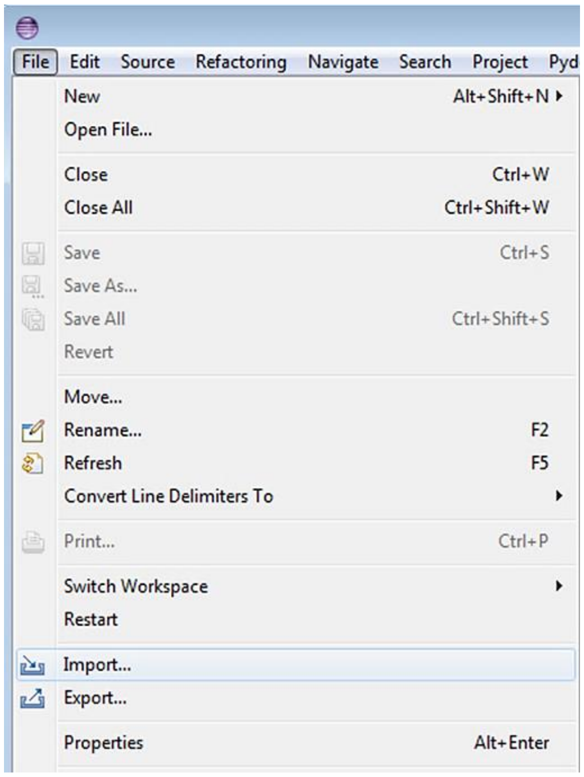


Figure 69: Checking Out VOLTTRON with Eclipse from GitHub

2. Select Git -> Projects from Git, then click the Next button (Figure 69)

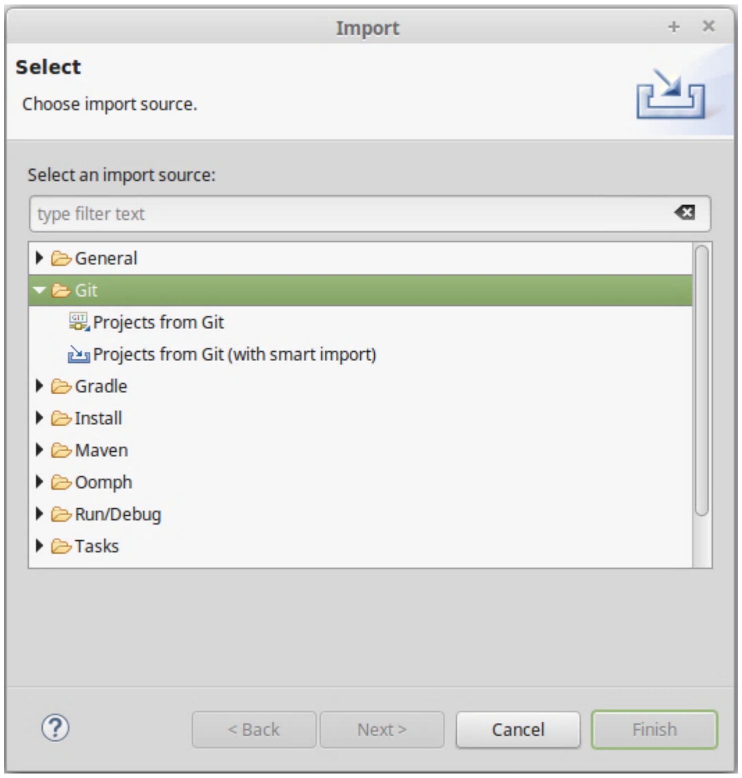
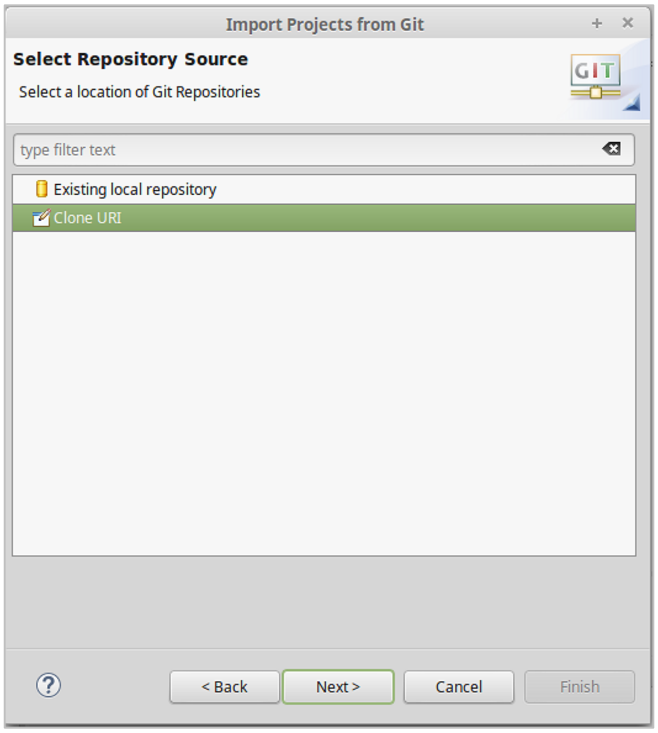


Figure 70: Checking Out VOLTTRON with Eclipse from GitHub (continued)

3. As shown in Figure 71, select Clone URI -> Next >



Select Repository Source

Select a location of Git Repositories

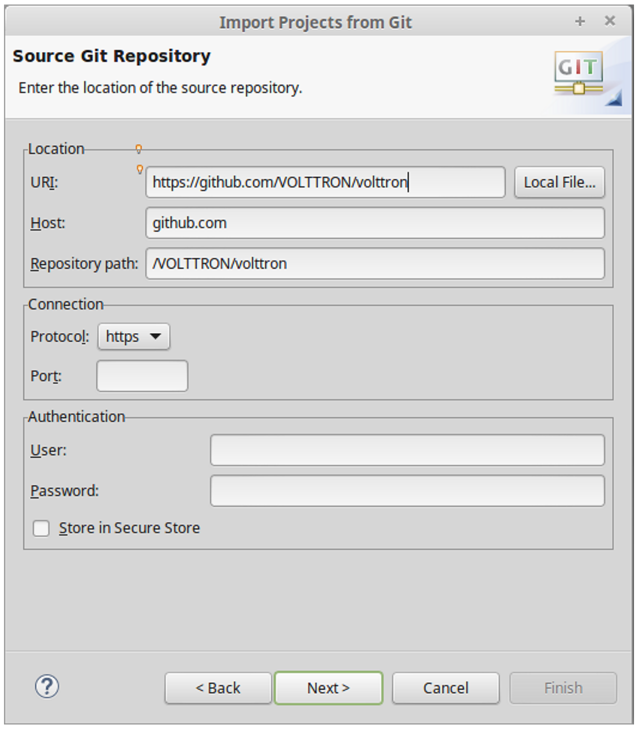
type filter text

Existing local repository

Clone URI

Figure 71: Checking Out VOLTTRON with Eclipse GitHub (continued)

4. Fill in https://github.com/VOLTTRON/volttron.git for the URI. If one has a GitHub account, enter a username and password in the User and Password sections. This is not required, but will allow one to receive notifications from GitHub for VOLTTRON related news (Figure 72):



Source Git Repository

Enter the location of the source repository.

Location

URI: https://github.com/VOLTTRON/volttron

Host: github.com

Repository path: /VOLTTRON/volttron

Connection

Protocol: https

Port:

Authentication

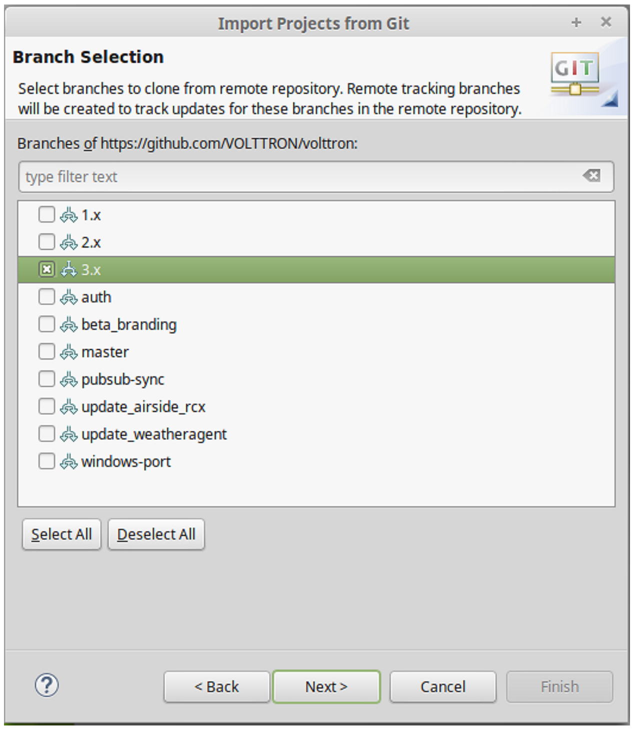
User:

Password:

Store in Secure Store

Figure 72: Checking Out VOLTTRON with Eclipse from GitHub (continued)

5. Select the 3.x branch (Figure 73)


Import Projects from Git

Branch Selection

Select branches to clone from remote repository. Remote tracking branches will be created to track updates for these branches in the remote repository.

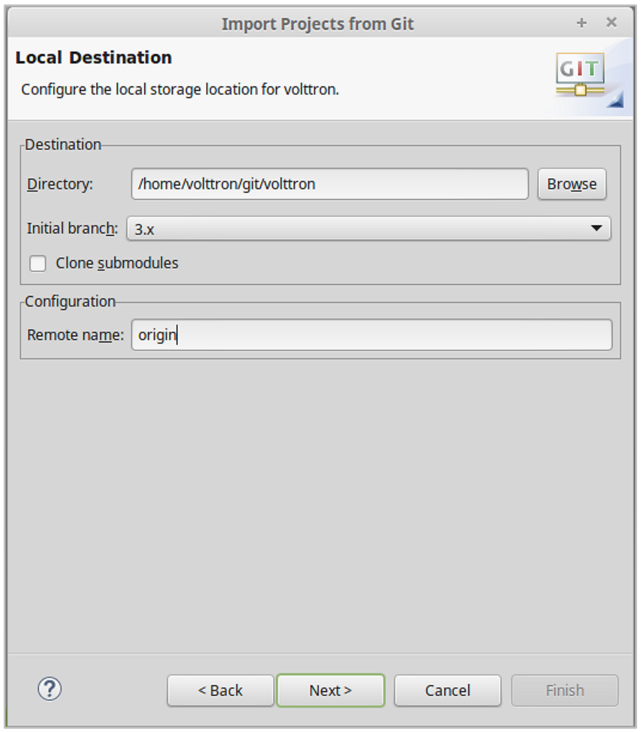
Branches of https://github.com/VOLTTRON/volttron:

type filter text

- [ ] 1.x
- [ ] 2.x
- [x] 3.x
- [ ] auth
- [ ] beta_branding
- [ ] master
- [ ] pubsub-sync
- [ ] update_airside_rcx
- [ ] update_weatheragent
- [ ] windows-port

Figure 73: Checking Out VOLTTRON with Eclipse from GitHub (continued)

6. Select a location to save the local repository (Figure 74)


Import Projects from Git

Local Destination

Configure the local storage location for volttron.

Destination

Directory: /home/volttron/git/volttron

Initial branch: 3.x

- [ ] Clone submodules

Configuration

Remote name: origin

Figure 74: Checking Out VOLTTRON with Eclipse from GitHub (continued)

7. Select Import as general project, select Next, then select Finish (Figure 75), the project will now be imported into the workspace

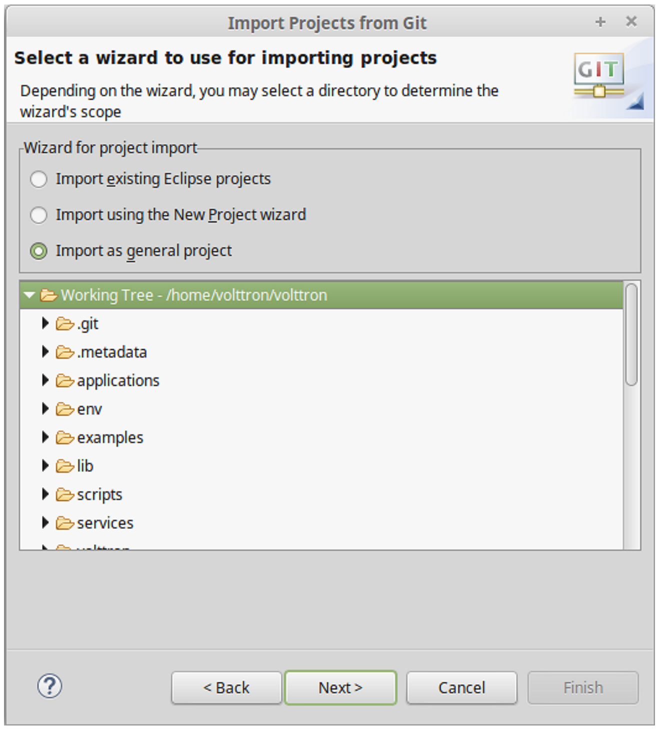

Import Projects from Git
Select a wizard to use for importing projects
Depending on the wizard, you may select a directory to determine the wizard's scope
Wizard for project import
Import existing Eclipse projects
Import using the New Project wizard
Import as general project
Working Tree - /home/volttron/volttron
.git
.metadata
applications
env
examples
lib
scripts
services
< Back
Next >
Cancel
Finish

Figure 75: Checking Out VOLTTRON with Eclipse from GitHub (continued)

If the VOLTTRON project has not been built (`<project directory>/bootstrap.py` file has not been run). Then proceed to Section 2.4 and follow the instruction for running the *bootstrap.py* script before proceeding to the following sections.

5.5.4 Linking Eclipse

PyDev must now be configured to use the Python interpreter packaged with VOLTTRON.

1. Select Window -> Preferences
2. Expand the PyDev tree
3. Select Interpreters -> Python interpreter
4. Select New (Figure 76)

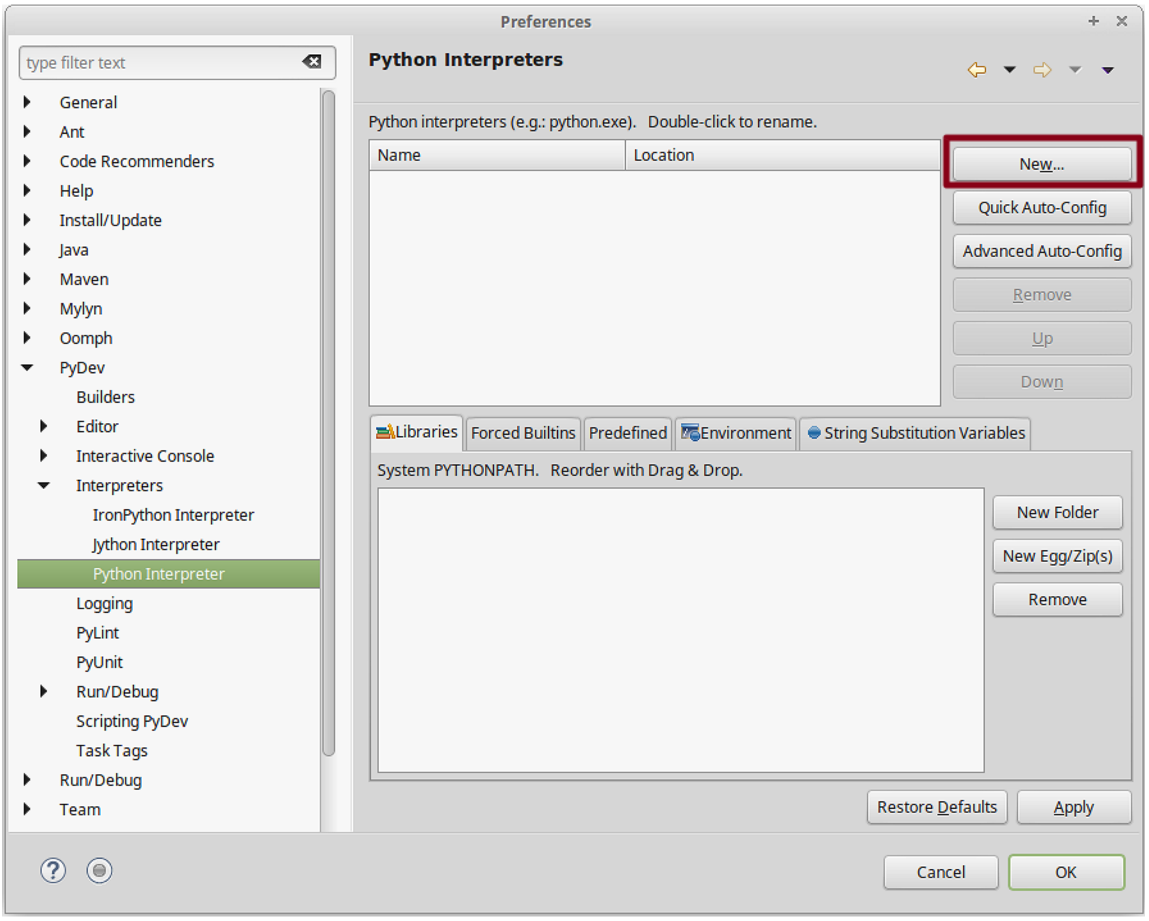
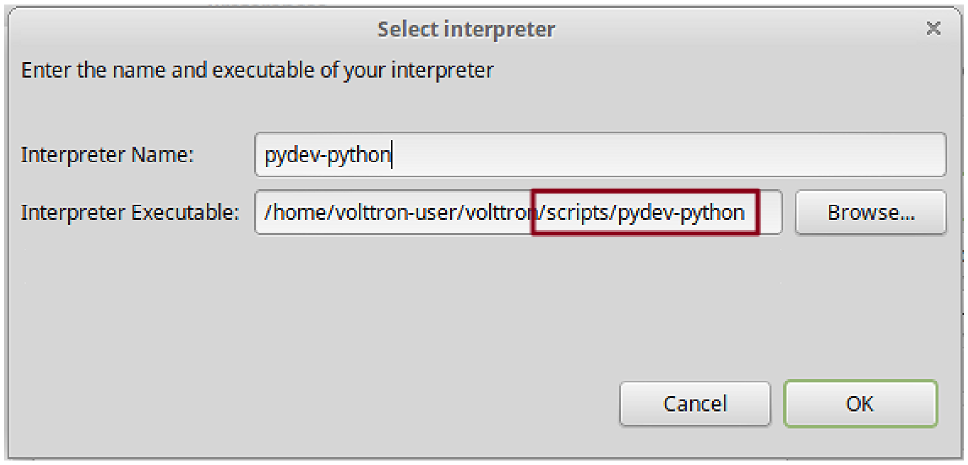


Figure 76: Configuring PyDev

5. Select Browse and navigate to the pydev-python file located at (`<project directory>/scripts/pydev-python`) (Figure 77)
6. Select OK (Figure 78)

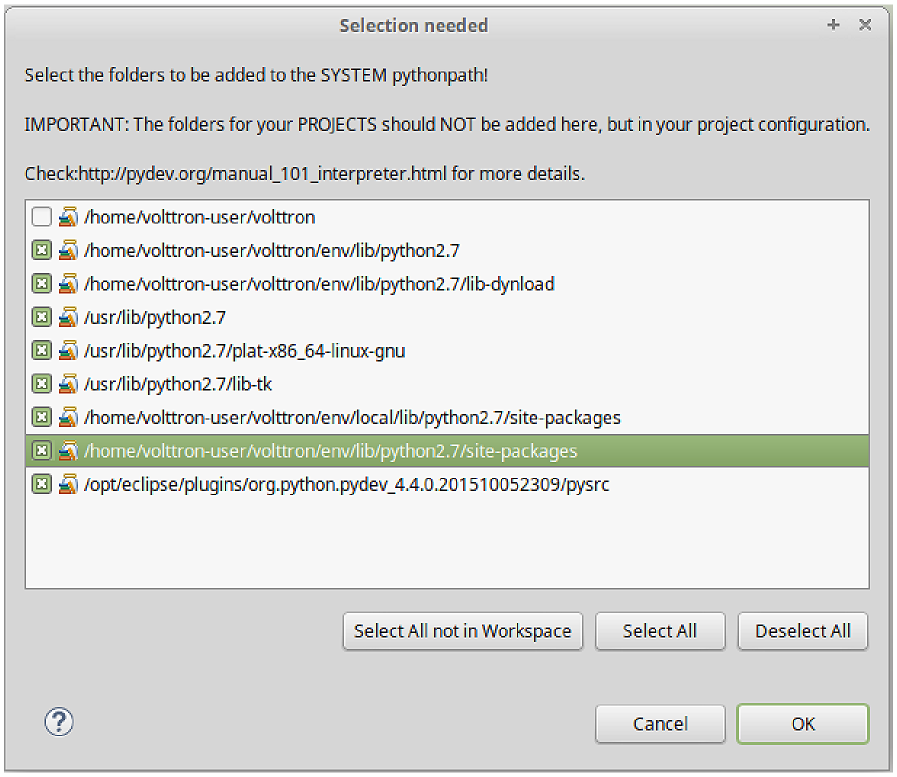


Interpreter Name: pydev-python

Interpreter Executable: /home/volttron-user/volttron/scripts/pydev-python

Figure 77: Configuring PyDev (continued)

7. Select All, then uncheck the VOLTTRON base directory as shown in Figure 78



Selection needed

Select the folders to be added to the SYSTEM pythonpath!

IMPORTANT: The folders for your PROJECTS should NOT be added here, but in your project configuration.

Check:http://pydev.org/manual_101_interpreter.html for more details.

- [ ] /home/volttron-user/volttron
- [x] /home/volttron-user/volttron/env/lib/python2.7
- [x] /home/volttron-user/volttron/env/lib/python2.7/lib-dynload
- [x] /usr/lib/python2.7
- [x] /usr/lib/python2.7/plat-x86_64-linux-gnu
- [x] /usr/lib/python2.7/lib-tk
- [x] /home/volttron-user/volttron/env/local/lib/python2.7/site-packages
- [x] /home/volttron-user/volttron/env/lib/python2.7/site-packages
- [x] /opt/eclipse/plugins/org.python.pydev_4.4.0.201510052309/pysrc

Figure 78: Configuring PyDev (continued)

8. In the Project/PackageExplorer view on the left, right-click on the project, PyDev-> Set as PyDev Project (Figure 80)

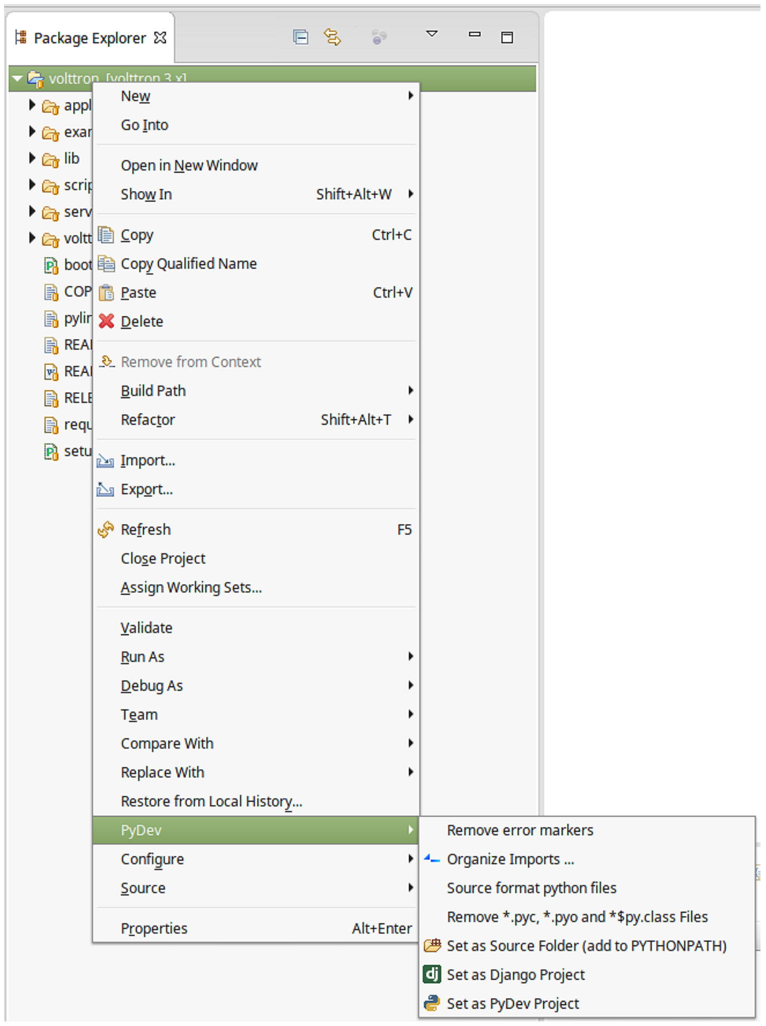


Figure 79: Setting as PyDev Project

9. Switch to the PyDev perspective, select Window -> Perspective -> Open Perspective -> Other -> PyDev (Figure 80)

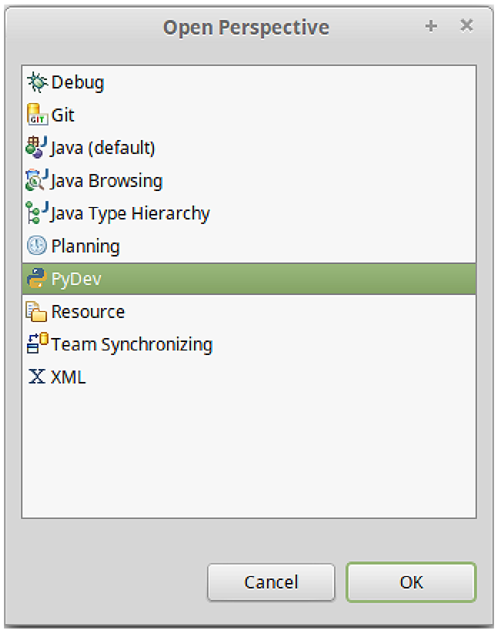


Figure 80: Setting PyDev Perspective in Eclipse

Eclipse should now be configured to use the project's environment.

5.5.5 Running the VOLTTRON Platform and Agents

Now VOLTTRON and agents within VOLTTRON can be run within Eclipse. This section will describe the process to run VOLTTRON and an agent within Eclipse.

5.5.5.1 Setup a Run Configuration for the Platform

The following steps will describe the process for running VOLTTRON within Eclipse:

1. Select Run -> Run Configurations (Figure 81)

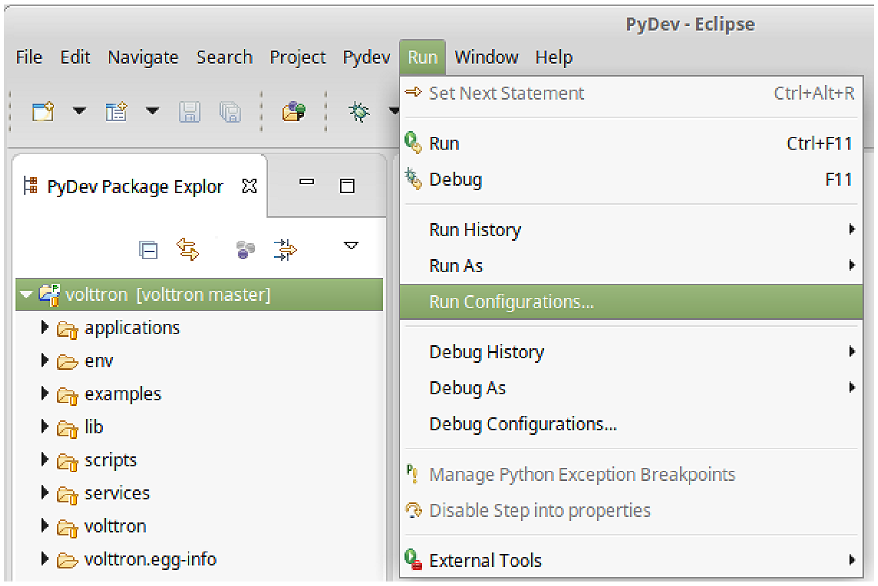
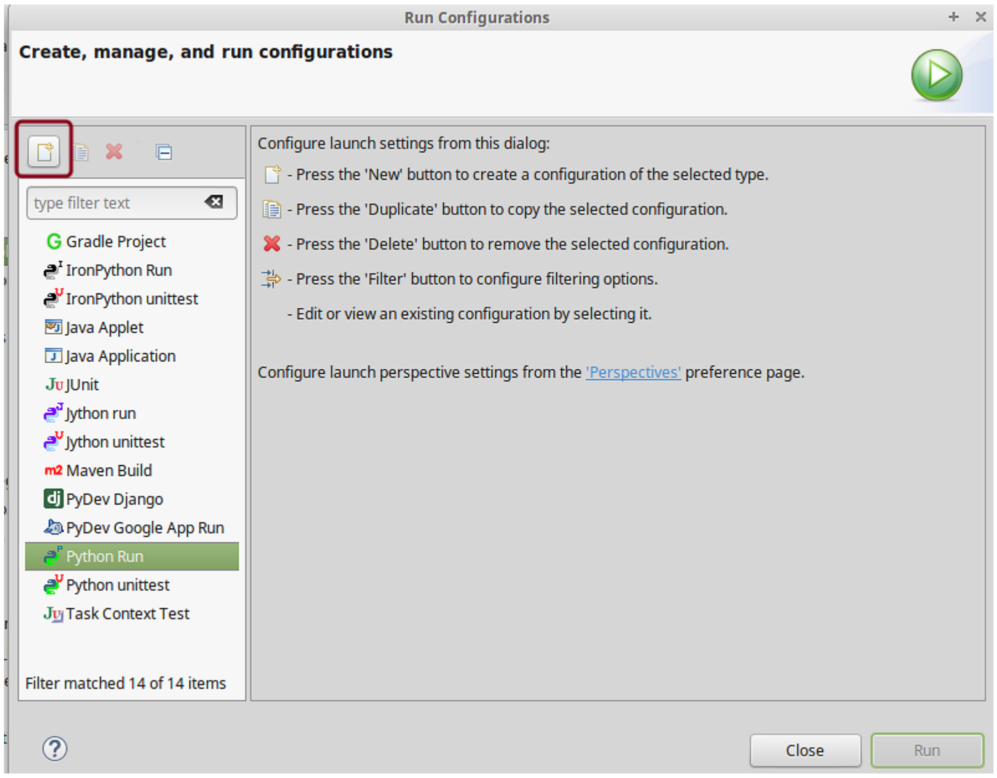


Figure 81: Running VOLTTRON Platform, Setting Up a Run Configuration

2. Select Python Run from the menu on left and Click the New launch configuration button (Figure 82)

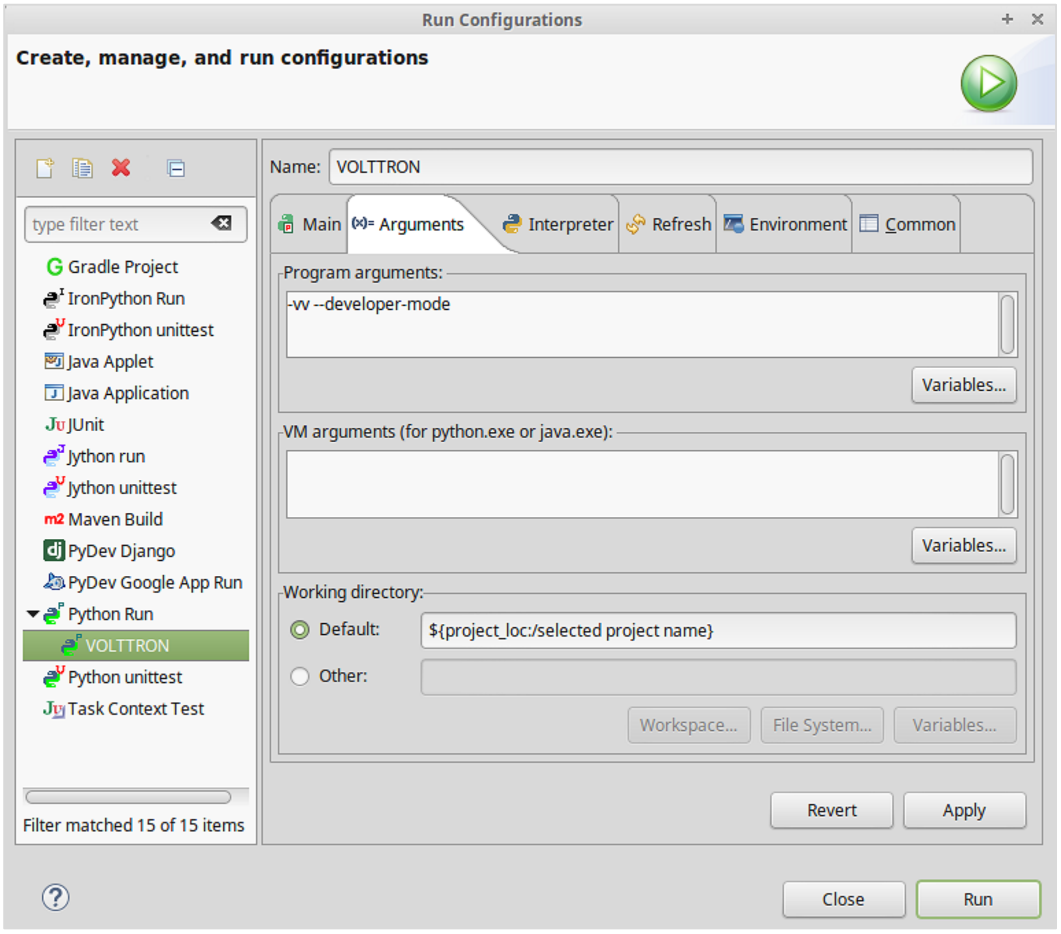


Configure launch settings from this dialog:
- Press the 'New' button to create a configuration of the selected type.
- Press the 'Duplicate' button to copy the selected configuration.
- Press the 'Delete' button to remove the selected configuration.
- Press the 'Filter' button to configure filtering options.
- Edit or view an existing configuration by selecting it.

Configure launch perspective settings from the 'Perspectives' preference page.

Figure 82: Running VOLTTRON Platform, Setting Up a Run Configuration (continued)

3. Change the name (any name may be used but for this example the name VOLTTRON was chosen) and select the main module (`<project directory>/volttron/platform/main.py`)
4. Select the Arguments tab and enter '-vv --developer-mode' in the Program arguments field (Figure 83) then select the Run button



Name: VOLTTRON

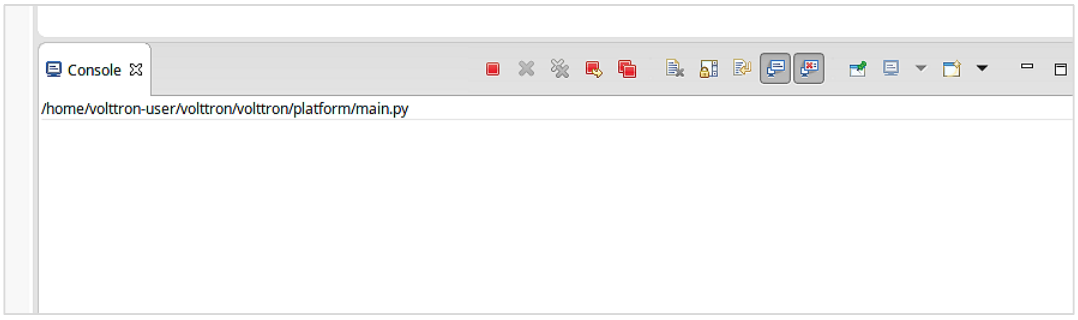
Program arguments: -vv --developer-mode

VM arguments (for python.exe or java.exe):

Working directory:
[✓ Default:] ${project_loc:/selected project name}
Other:

Figure 83: Running VOLTTRON Platform, Setting Up a Run Configuration (continued)

5. If the run is successful, the console should appear similar to Figure 84. If the run does not succeed (red text describing why the run failed will populate the console), click the all stop icon (two red boxes overlaid) on the console and then retry.



```
/home/volttron-user/volttron/volttron/platform/main.py
```

Figure 84: Running VOLTTRON Platform, Console View on Successful Run

5.5.5.2 Configure a Run Configuration for the Listener Agent

The following steps will describe the process for configuring an agent within Eclipse:

1. Select Run -> Run Configurations (Figure 89)

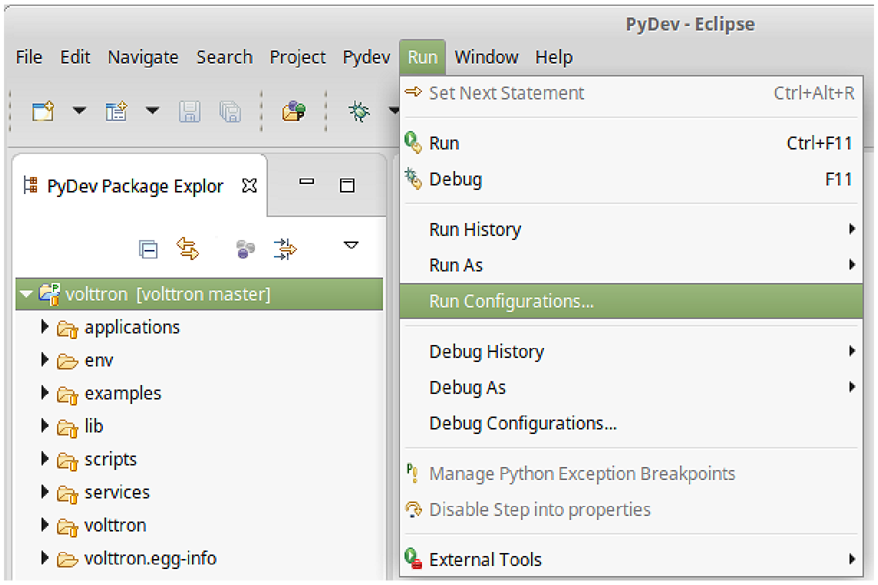
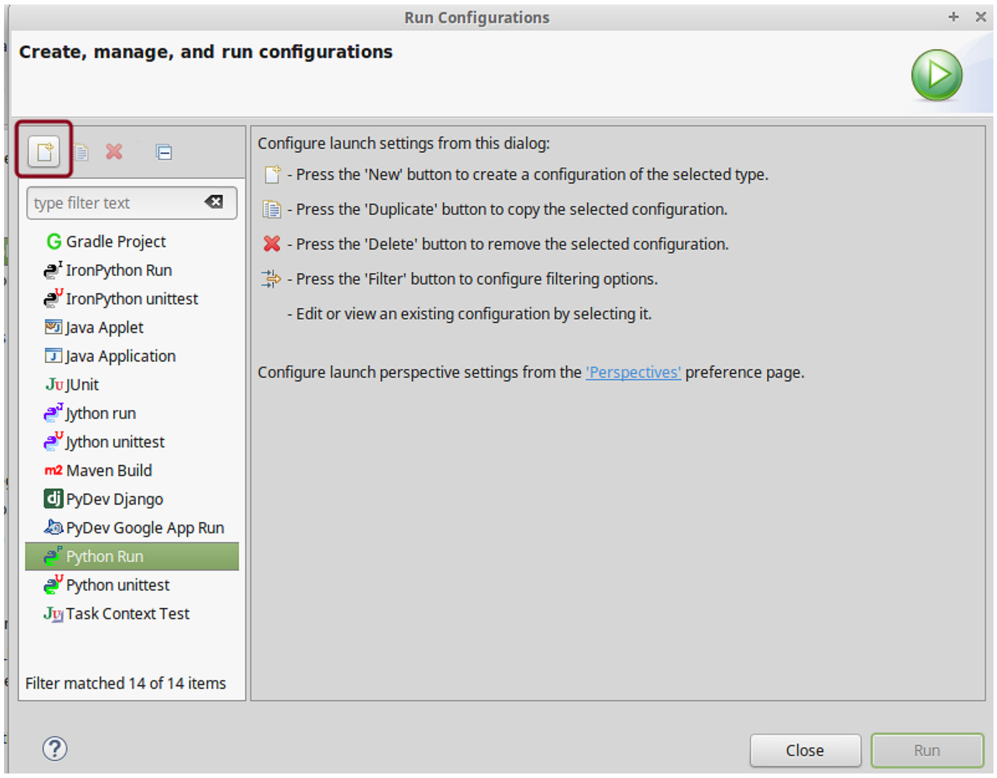


Figure 85: Running the Listener Agent, Setting Up a Run Configuration

2. Select Python Run from the menu on left and Click the New launch configuration button (Figure 90)

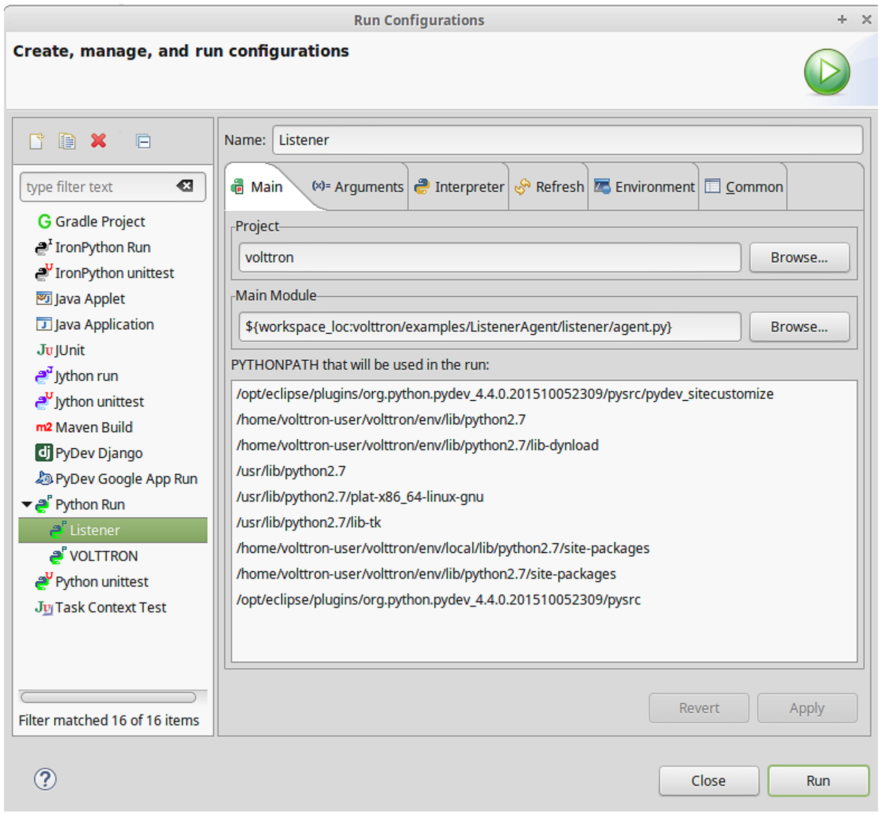


Configure launch settings from this dialog:
- Press the 'New' button to create a configuration of the selected type.
- Press the 'Duplicate' button to copy the selected configuration.
- Press the 'Delete' button to remove the selected configuration.
- Press the 'Filter' button to configure filtering options.
- Edit or view an existing configuration by selecting it.

Configure launch perspective settings from the 'Perspectives' preference page.

Figure 86: Running the Listener Agent, Setting Up a Run Configuration (continued)

3. Change the name (for this example Listener is used) and select the main module (`<project directory>/examples/ListenerAgent/listener/agent.py`) (Figure 87)



Name: Listener

Project: volttron

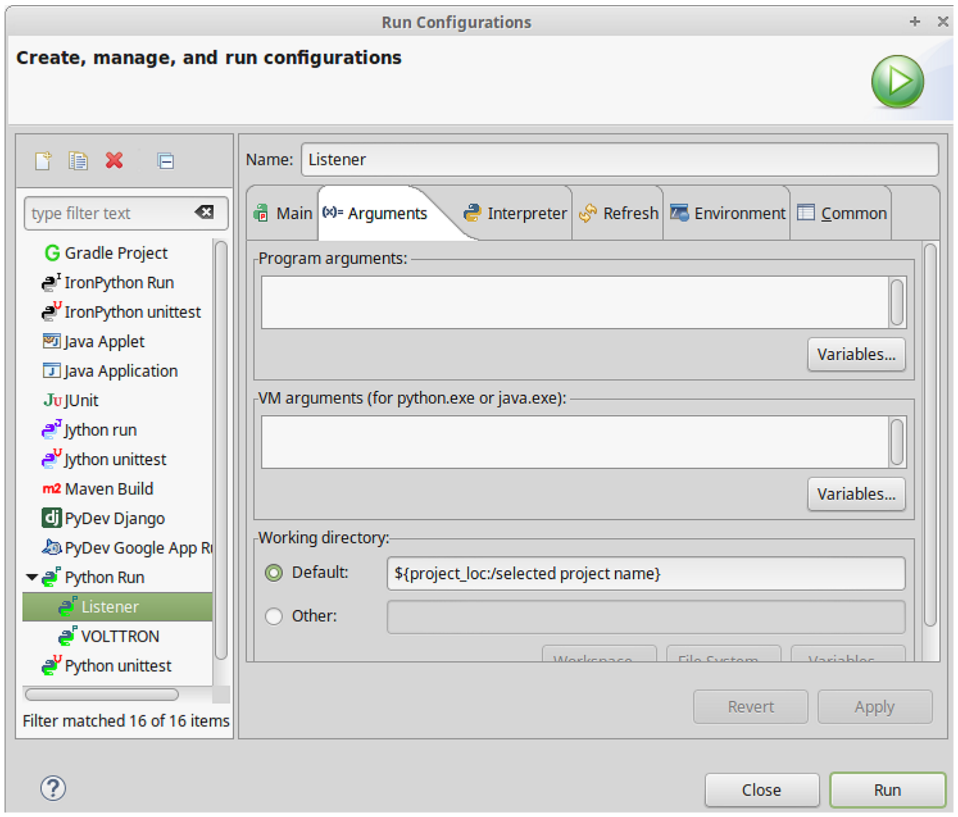
Main Module: ${workspace_loc:volttron/examples/ListenerAgent/listener/agent.py}

PYTHONPATH that will be used in the run:

```
/opt/eclipse/plugins/org.python.pydev_4.4.0.201510052309/pysrc/pydev_sitecustomize
/home/volttron-user/volttron/env/lib/python2.7
/home/volttron-user/volttron/env/lib/python2.7/lib-dynload
/usr/lib/python2.7
/usr/lib/python2.7/plat-x86_64-linux-gnu
/usr/lib/python2.7/lib-tk
/home/volttron-user/volttron/env/local/lib/python2.7/site-packages
/home/volttron-user/volttron/env/lib/python2.7/site-packages
/opt/eclipse/plugins/org.python.pydev_4.4.0.201510052309/pysrc
```

Figure 87: Running the Listener Agent, Setting Up a Run Configuration (continued)

4. Click on the Arguments tab and change Working directory to Default (Figure 88)



Name: Listener

Program arguments:

VM arguments (for python.exe or java.exe):

Working directory:

[✓ Default:] ${project_loc:/selected project name}

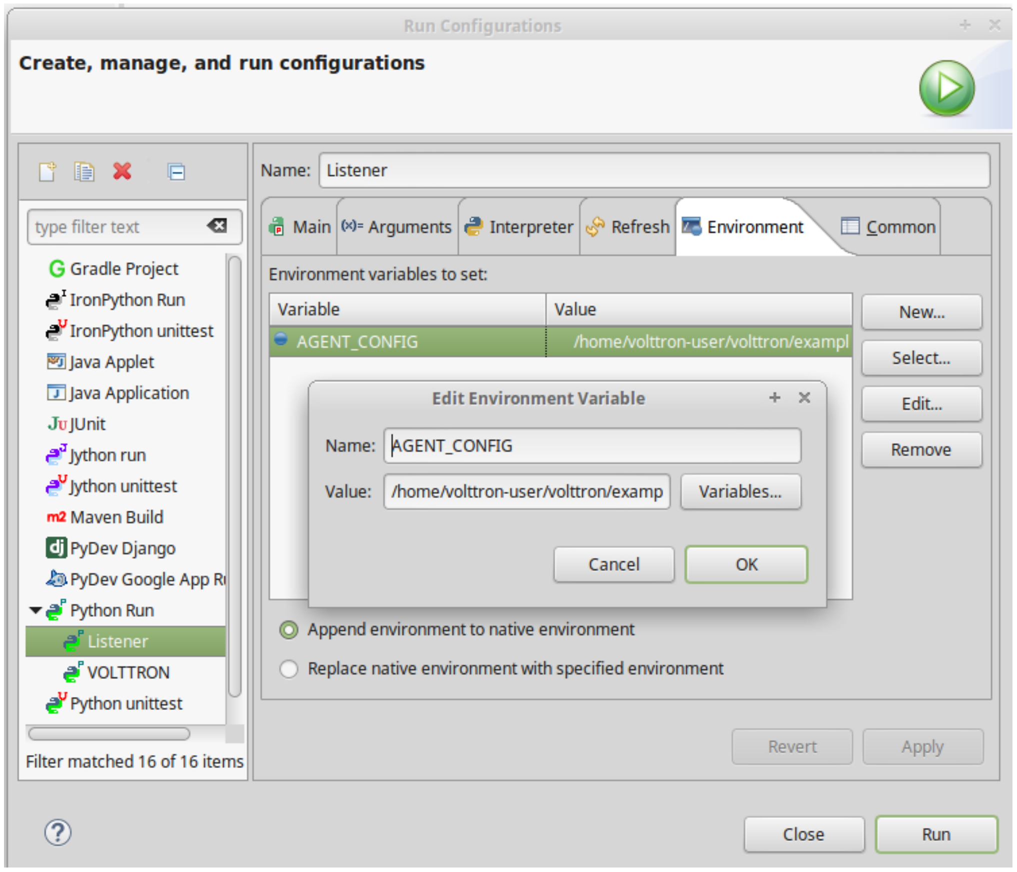
Other:

Figure 88: Running the Listener Agent, Setting Up a Run Configuration (continued)

5. In the Environment tab, select New -> add the following environment variables (bulleted list below), as shown in Figure 89:

    - AGENT_CONFIG = /home/<USER>/examples /ListenerAgent/config

    AGENT_CONFIG is the absolute path the agent's configuration file. To access a remote message bus, use the VIP address as described in Section 3.5



Name: Listener

Environment variables to set:

Variable	Value
AGENT_CONFIG	/home/volttron-user/volttron/exampl

Name: AGENT_CONFIG

Value: /home/volttron-user/volttron/examp

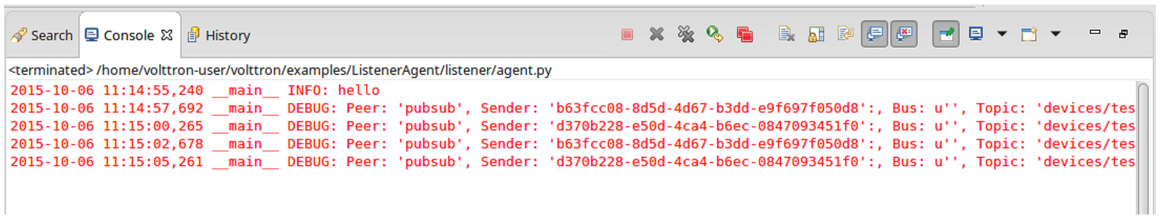
[✓ Append environment to native environment]

Replace native environment with specified environment

Figure 89: Running the Listener Agent, Setting Up a Run Configuration

6. Click Run, this launches the agent (Figure 89)

You should see the agent start to publish and receive its own heartbeat message (Figure 90).



```
<terminated> /home/volttron-user/volttron/examples/ListenerAgent/listener/agent.py
2015-10-06 11:14:55,240 __main__ INFO: hello
2015-10-06 11:14:57,692 __main__ DEBUG: Peer: 'pubsub', Sender: 'b63fcc08-8d5d-4d67-b3dd-e9f697f050d8':, Bus: u'', Topic: 'devices/tes
2015-10-06 11:15:00,265 __main__ DEBUG: Peer: 'pubsub', Sender: 'd370b228-e50d-4ca4-b6ec-0847093451f0':, Bus: u'', Topic: 'devices/tes
2015-10-06 11:15:02,678 __main__ DEBUG: Peer: 'pubsub', Sender: 'b63fcc08-8d5d-4d67-b3dd-e9f697f050d8':, Bus: u'', Topic: 'devices/tes
2015-10-06 11:15:05,261 __main__ DEBUG: Peer: 'pubsub', Sender: 'd370b228-e50d-4ca4-b6ec-0847093451f0':, Bus: u'', Topic: 'devices/tes
```

Figure 90: Listener Agent Output on Eclipse Console

The process for running other agents in Eclipse is identical to that of the Listener agent. There are many useful development tools available within Eclipse and PyDev that make development, debugging, and testing of agents much simpler.

5.5.6 Agent Creation Walkthrough

It is recommended that developers look at the Listener agent before developing their own agent. The Listener agent illustrates the basic functionality of an agent. The following example will demonstrate the steps for creating an agent.

5.5.6.1 Agent Folder Setup

Creating a folder within the workspace will help consolidate the code your agent will utilize.

1. In the VOLTTRON base directory, create a new folder TestAgent
2. In TestAgent, create a new folder tester; this is the package where the Python code will be created (Figure 91).

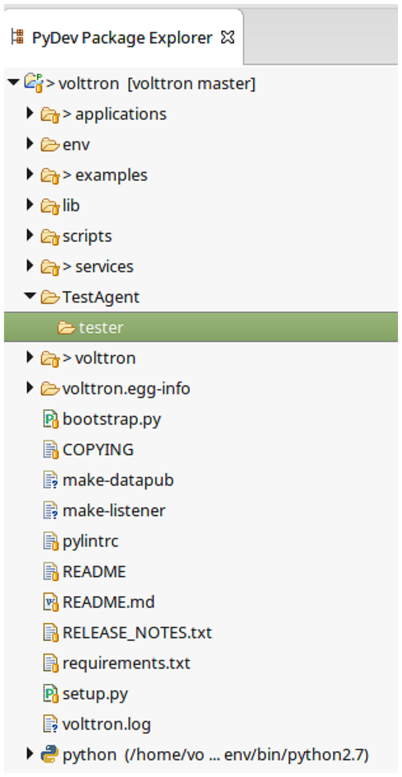


Figure 91: Creating an Agent Test Folder

5.5.6.2 *Create Agent Code*

The following steps describe the necessary agent files and modules.

- In tester, create a file called *__init__.py*, which tells Python to treat this folder as a package
- In the tester package folder, create the file *testagent.py*
- Create a class called TestAgent
- Import the packages and classes needed:

```
from __future__ import absolute_import

from datetime import datetime
import logging
import sys

from volttron.platform.vip.agent import Agent, Core
from volttron.platform.agent import utils
```

Next, set up a logger. The `utils` module from `volttron.platform.agent` builds on Python's already robust logging module and is easy to use. Add the following lines after the import statements:

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

This agent will inherit features from the Agent class (base class) extending the agent's default functionality. The class definition for the TestAgent will be configured as shown below (with `__init__`):

```
class TestAgent(Agent):
    def __init__(self, config_path, **kwargs):
        super(TestAgent, self).__init__(**kwargs)
```

5.5.6.3 *Setting up a Subscription*

First, create a startup method. This method is tagged with the decorator `@Core.receiver("onstart")`. The startup method will run after the agent is initialized. The TestAgent's startup method will contain a subscription to the Listener agent's heartbeat (heartbeat/listeneragent). The TestAgent will detect when a message with this topic is published on the message bus and will run the method specified with the callback keyword argument passed to `self.vip.pubsub.subscribe`.

```
@Core.receiver("onstart")
def starting(self, sender, **kwargs):
    '''
    Subscribes to the platform message bus on
    the heatbeat/listeneragent topic
    '''
    print('TestAgent example agent start-up function')
    self.vip.pubsub.subscribe('pubsub', 'heartbeat/listeneragent',
                              callback=self.on_heartbeat)
```

Next, we need to create the callback method. Typically, the callback is the response to a message (or event). In this simple example, the TestAgent will do a print statement and publish a message to the bus:

```
def on_heartbeat(self, peer, sender, bus, topic, headers, message):
    '''TestAgent callback method'''
    print('Matched topic: {}, for bus: {}'.format(topic, bus))
    self.vip.pubsub.publish('pubsub',
                            'testagent/publish',
                            headers=headers,
                            message='test publishing').get(timeout=30)
```

5.5.6.4 *Argument Parsing and Main Method*

The test agent will need to be able to parse arguments being passed on the command line by the agent launcher. Use the **`utils.default_main`** method to handle argument parsing and other default behavior.

1. Create a main method that can be called by the launcher:

```
def main(argv=sys.argv):
    '''Main method called by the eggsecutable.'''
    try:
        utils.vip_main(TestAgent)
    except Exception as e:
        _log.exception(e)


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

5.5.6.5 *Create Support Files for Test Agent*

VOLTTRON agents need configuration files for packaging, configuration, and launching. The “setup.py” file details the naming and Python package information. The launch configuration file is a JSON formatted text file used by the platform to launch instances of the agent.

5.5.6.6 *Packaging Configuration*

In the TestAgent folder, create a file called "setup.py". This file sets up the name, version, required packages, method to execute, etc. for the agent. The packaging process will also use this information to name the resulting file.

```
from setuptools import setup, find_packages

#get environ for agent name/identifier
packages = find_packages('.')
package = packages[0]

setup(
    name = package + 'agent',
    version = "0.1",
    install_requires = ['volttron'],
    packages = packages,
    entry_points = {
        'setuptools.installation': [
            'eggsecutable = ' + package + '.testagent:main',
        ]
    }
)
```

5.5.6.7 *Launch Configuration*

In TestAgent, create a file called "testagent.launch.json". This is the file the platform will use to launch the agent. It can also contain configuration parameters for the agent:

```
{
    "agentid": "Test1"
}
```

5.5.6.8 *Testing the Agent*

From a terminal, in the base VOLTTRON directory, enter the following commands (with the platform activated and VOLTTRON running):

1. Run pack_install script on TestAgent:

```
$ ./scripts/core/pack_install.sh TestAgent TestAgent/config test-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/testeragent-0.1-py2-none-any.whl as d4ca557a-496c-4f02-8ad9-42f5d435868a testeragent-0.1`

2. Start the agent:

```
$ volttron-ctl start --tag test-agent
```

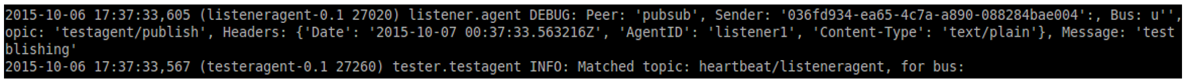
3. Verify that agent is running:

```
$ volttron-ctl 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:

```
$ volttron-ctl stop --tag test-agent
$ volttron-ctl remove --tag test-agent
```

Then re-build and start the updated agent**.**



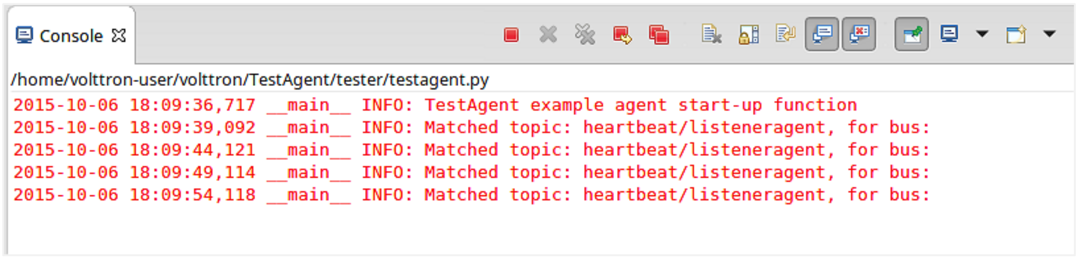
```
2015-10-06 17:37:33,605 (listeneragent-0.1 27020) listener.agent DEBUG: Peer: 'pubsub', Sender: '036fd934-ea65-4c7a-a890-088284bae004':, Bus: u'',
opic: 'testagent/publish', Headers: {'Date': '2015-10-07 00:37:33.563216Z', 'AgentID': 'listener1', 'Content-Type': 'text/plain'}, Message: 'test
blishing'
2015-10-06 17:37:33,567 (testeragent-0.1 27260) tester.testagent INFO: Matched topic: heartbeat/listeneragent, for bus:
```

Figure 92: TestAgent Output In VOLTTRON Log

5.5.6.9 *Running the TestAgent in Eclipse*

If you are working in Eclipse, create a run configuration for the TestAgent based on the Listener agent configuration in the Eclipse development environment (Section 5.5.5).

- Launch the platform (Section 5.5.5.1)
- Launch the TestAgent:
    - (follow steps outlined in Section 5.5.5.2 for launching the Listener agent)
- Launch the Listener agent
- TestAgent should start receiving the heartbeats from Listener agent and the following should be displayed in the console (Figure 93)



```
/home/volttron-user/volttron/TestAgent/tester/testagent.py
2015-10-06 18:09:36,717 __main__ INFO: TestAgent example agent start-up function
2015-10-06 18:09:39,092 __main__ INFO: Matched topic: heartbeat/listeneragent, for bus:
2015-10-06 18:09:44,121 __main__ INFO: Matched topic: heartbeat/listeneragent, for bus:
2015-10-06 18:09:49,114 __main__ INFO: Matched topic: heartbeat/listeneragent, for bus:
2015-10-06 18:09:54,118 __main__ INFO: Matched topic: heartbeat/listeneragent, for bus:
```

Figure 93: Console Output for TestAgent

5.5.7 Adding Additional Features to the TestAgent

Additional code can be added to the TestAgent to utilize additional services in the platform. The following sections will show how to use the weather and device scheduling service within the TestAgent.

5.5.7.1 Subscribing to Weather Data

This agent can be modified to listen to weather data from the Weather agent by adding the following line at the end of the TestAgent startup method. This will subscribe the agent to the temperature subtopic. For the full list of topics available, please see:

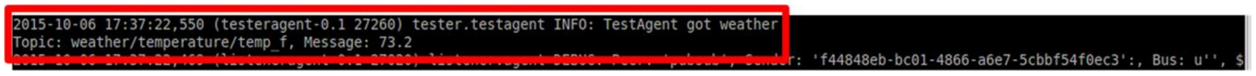
https://github.com/VOLTTRON/volttron/wiki/WeatherAgentTopics

```
self.vip.pubsub.subscribe('pubsub', 'weather/temperature/temp_f',
                          callback=self.on_weather)
```

Then add the callback method `on_weather`:

```
def on_weather(self, peer, sender, bus, topic, headers, message):
    print("TestAgent got weather\nTopic: {}, Message: {}".format(topic, message))
```

The platform log file should appear similar to Figure 94.



```
2015-10-06 17:37:22,550 (testeragent-0.1 27260) tester.testagent INFO: TestAgent got weather
Topic: weather/temperature/temp_f, Message: 73.2
[illegible]r: 'f44848eb-bc01-4866-a6e7-5cbbf54f0ec3':, Bus: u'', $
```

Figure 94: TestAgent Output when Subscribing to Weather Topic

5.5.7.2 Utilizing the Scheduler Agent

The TestAgent can be modified to publish a schedule to the Actuator agent by reserving time on virtual devices. Modify the following code to include current time ranges and include a call to the publish schedule method in setup. The following example will post a simple schedule. For more detailed information on device scheduling, please see:

https://github.com/VOLTTRON/volttron/wiki/ActuatorAgent

Ensure the Actuator agent is running as per Section 3.3. Add the following line to the TestAgent's import statements:

```
from volttron.platform.messaging import topics
```

Add the following lines to the TestAgent's **`starting`** method. This sets up a subscription to the ACTUATOR_RESPONSE topic and calls the **`publish_schedule`** method.

```
self.vip.pubsub.subscribe('pubsub', topics.ACTUATOR_RESPONSE,
                          callback=self.on_schedule_result)
self.publish_schedule()
```

The **`publish_schedule`** method sends a schedule request message to the Actuator agent (Update the schedule with appropriate times):

```
def publish_schedule(self):
    headers = {
                'AgentID': self._agent_id,
                'type': 'NEW_SCHEDULE',
                'requesterID': self._agent_id, # Name of requesting agent
                'taskID': self._agent_id + "-TASK", # Unique task ID
                'priority': 'LOW'             # Task Priority (HIGH, LOW, LOW_PREEMPT)
    }
    msg = [
                ["campus/building/device1", # First time slot.
                 "2014-1-31 12:27:00",      # Start of time slot.
```

```
             "2014-1-31 12:29:00"],     # End of time slot.
            ["campus/building/device1", # Second time slot.
             "2014-1-31 12:26:00",      # Start of time slot.
             "2014-1-31 12:30:00"],     # End of time slot.
            ["campus/building/device2", # Third time slot.
             "2014-1-31 12:30:00",      # Start of time slot.
             "2014-1-31 12:32:00"],     # End of time slot.
            #etc...
        ]
    self.vip.rpc.call('platform.actuator',       # Target agent
                      'request_new_schedule',    # Method to call
                       agent_id,                 # Requestor
                      "some task",               # TaskID
                      "LOW",                     # Priority
                       msg).get(timeout=10)     # Request message
```

Add the call back method for the schedule request:

```
def on_schedule_result(self, topic, headers, message, match):
    print (("TestAgent schedule result \nTopic: {topic}, "
            "{headers}, Message: {message}")
            .format(topic=topic, headers=headers, message=message))
```

5.5.7.3 Full TestAgent Code

The following is the full TestAgent code built in the previous steps:

```
from __future__ import absolute_import

from datetime import datetime
import logging
import sys

from volttron.platform.vip.agent import Agent, Core
from volttron.platform.agent import utils
from volttron.platform.messaging import headers as headers_mod

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

class TestAgent(Agent):
    def __init__(self, config_path, **kwargs):
        super(TestAgent, self).__init__(**kwargs)

    @Core.receiver("onstart")
    def starting(self, sender, **kwargs):
        '''
        Subscribes to the platform message bus on
        the heatbeat/listeneragent topic
        '''
        _log.info('TestAgent example agent start-up function')
        self.vip.pubsub.subscribe(peer='pubsub', topic='heartbeat/listeneragent',
                                  callback=self.on_heartbeat)
        self.vip.pubsub.subscribe('pubsub', topics.ACTUATOR_RESPONSE,
                                    callback=self.on_schedule_result)
        self.vip.pubsub.subscribe('pubsub', 'weather/temperature/temp_f',
                                    callback=self.on_weather)


        self.publish_schedule()

    def on_heartbeat(self, peer, sender, bus, topic, headers, message):
        '''TestAgent callback method'''
        _log.info('Matched topic: {}, for bus: {}'.format(topic, bus))
        self.vip.pubsub.publish(peer='pubsub',
                                topic='testagent/publish',
                                headers=headers,
                                message='test publishing').get(timeout=30)

    def on_weather(self, peer, sender, bus, topic, headers, message):
        _log.info(
            "TestAgent got weather\nTopic: {}, Message: {}".format(topic, message))

    def on_schedule_result(self, topic, headers, message, match):
        print (("TestAgent schedule result \nTopic: {topic}, "
                "{headers}, Message: {message}")
                .format(topic=topic, headers=headers, message=message))

def main(argv=sys.argv):
    '''Main method called by the eggsecutable.'''
    try:
        utils.vip_main(TestAgent)
    except Exception as e:
        _log.info(e)


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

6 Additional VOLTTRON Features (AKA VOLTTRON Restricted)

VOLTTRON Restricted adds a broader security layer on top of the VOLTTRON platform. If you are interested in this package, please contact the VOLTTRON team at volttron@pnnl.gov.

- NOTE: Once the package is installed, all aspects of the package will be enforced. To override VOLTTRON Restricted and disable the package, see Section 6.2.4.

The VOLTTRON Restricted package contains the following security enhancements:

- The creation and use of platform-specific Certificate Authority (CA) certificates.
- Multi-level signing of agent packages.
- Multi-level verification of signed packages during agent execution.
- Command line and agent-based mobility.
- Allows developer to customize an execution contract for required resources on the current and move requested platform.

The following features are enabled by the VOLTTRON Restricted package:

- Signing and verification of agent packages
- Resource monitor
- Example PingPong agent

6.1 Installation of VOLTTRON Restricted

The VOLTTRON Restricted software requires the installation of SWIG. SWIG is a software development tool that connects programs written in C and C++ with a variety of high-level programming languages. To install the VOLTTRON Restricted software, enter the following command from in a terminal from the base VOLTTRON directory:

- Install additional VOLTTRON Restricted dependency:

  ```
  # apt-get install swig
  ```

- Activate VOLTTRON platform:

  ```
  $ . env/bin/activate
  ```

  Note the space after the period.

- Install VOLTTRON Restricted:

  ```
  $ pip install -e <path to volttron restricted>
  ```

6.2 Enabling and Configuring VOLTTRON Restricted Software

The creation of a signed agent package requires four certificates. The developer (creator) certificate is used to sign the agent code and allows the platform to verify that the agent code has not been modified since being distributed. The admin (SOI) certificate is used for allowing the agent into a scope of influence. The initiator certificate is used when the agent is ready to be deployed into a specific platform. The platform certificate is used to sign the possibly modified data that an agent would like to carry with it during moving from platform to platform. All of these certificates must be signed by a "known" Certificate Authority (CA). Figure 95 shows the structure of the agent signing feature:

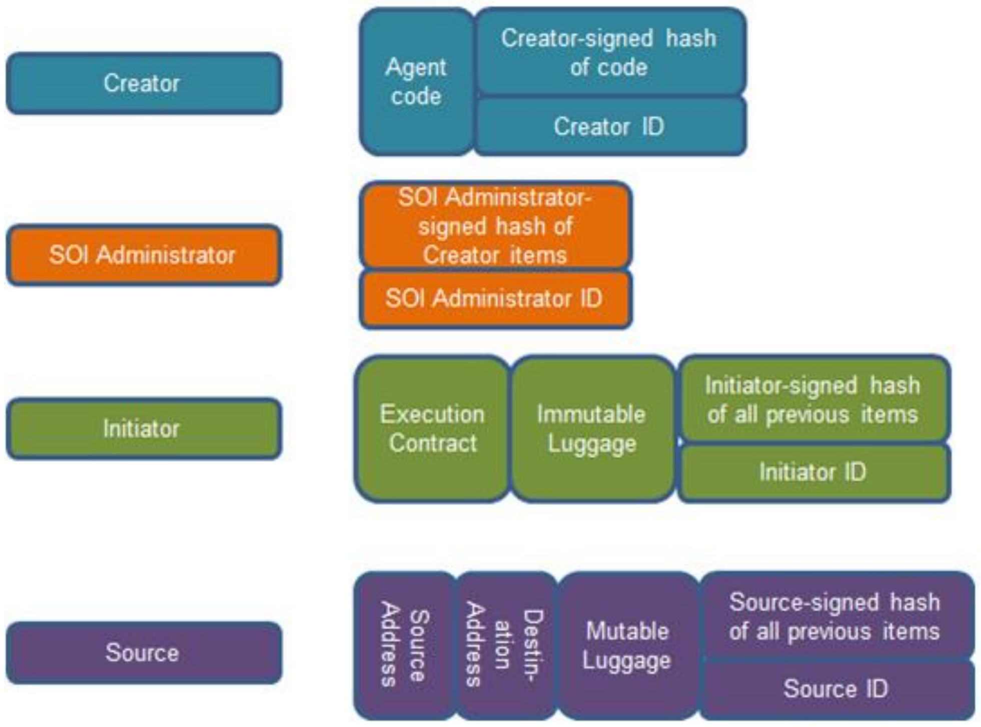


Figure 95: Structure for the Agent Signing Security Feature in VOLTTRON Restricted

To facilitate the development of agents, VOLTTRON Restricted includes packaging commands for creating the platform CA as well as the CA signed certificates for use in the agent signing process.

When the VOLTTRON Restricted package is installed on a platform, the volttron-pkg command will be expanded to

usage: volttron-pkg [-h] [-l FILE] [-L FILE] [-q] [-v] [--verboseness LEVEL] {package,repackage,configure,create_ca,create_cert,sign,verify}

The additional sub-commands:

- **create_ca** - Creates a platform specific root CA. When this command is executed, the user will be required to respond to prompts to fill out the certificate's data.
- **create_cert** - Allows the creation of a CA signed certificate. A type of certificate must be specified as (--creator | --soi | --initiator | --platform) and the name(--name) of the

certificate may be specified. The name will be used as the filename for the certificate on the platform.

- **sign** - Signs the agent package at the specified level.
  - Agent package to be signed (ALWAYS REQUIRED).
  - Signing level must be specified as one of (--creator | --soi | --initiator | --platform) and must be presented in the correct order. In other words, a soi cannot sign the package until the creator has signed it (ALWAYS REQUIRED).
  - **--contract** - (resource contract) a file containing the definition of the necessary agent resources needed to execute properly. This option is only available to the creator.
  - **--config-file** - a file used to define custom configuration for the starting of agent on the platform. This option is available to the initiator.
  - **--certs_dir** - allows the specification of where the certificate store is located. If this is not specified, the default certificate store will be used.
- **verify** - allows the user to verify a package is valid.
  - package - The agent package to validate against.

6.2.1 Creating Required Security Certificates

The following steps describe how to create the required security certificates to run the VOLTTRON Restricted code. From a terminal, in the base VOLTTON directory, enter the following commands:

1. Activate the VOLTTRON platform:

```
$ . env/bin/activate
```

2. Create the root security certificate:

```
$ volttron-pkg create_ca
```

   - Enter information when prompted: Country (default=US), State (default=Washington), Location (default=Richland), Organization (default=PNNL), Organizational Unit (default=Volttron Team), Common Name (default=hostname volttron-ca)

3. Create creator security certificates:

```
$ volttron-pkg create_cert --creator
```

   - Enter information when prompted: Country (default=US), State (default=Washington), Location (default=Richland), Organization (default=PNNL), Organizational Unit (default=Volttron Team), Common Name (default=creator)

4. Create the initiator security certificates:

```
$ volttron-pkg create_cert --initiator
```

- Enter information when prompted: Country (default=US), State (default=Washington), Location (default=Richland), Organization (default=PNNL), Organizational Unit (default=Volttron Team), Common Name (default=initiator)

5. Create the SOI security certificates:

```
$ volttron-pkg create_cert --soi
```

- Enter information when prompted: Country (default=US), State (default=Washington), Location (default=Richland), Organization (default=PNNL), Organizational Unit (default=Volttron Team), Common Name (default=soi)

6. Create the platform security certificates:

```
$ volttron-pkg create_cert --platform
```

- Enter information when prompted: Country (default=US), State (default=Washington), Location (default=Richland), Organization (default=PNNL), Organizational Unit (default=Volttron Team), Common Name (default=platform)

6.2.2 Enabling Agent Mobility Feature

To create the required keys (minimum requirement to run VOLTTRON with Restricted module installed), enter the following commands in a command terminal:

1. Create ssh directory in VOLTTRON_HOME (see Section 2.5 for platform configuration details):

```
$ mkdir -p ~/.volttron/ssh
```

2. Generate ssh key and add to id_rsa file:

```
$ ssh-keygen -t rsa -N '' -f ~/.volttron/ssh/id_rsa
```

3. Create empty file for authorized keys and know hosts:

```
$ touch ~/.volttron/ssh/{authorized_keys,known_hosts}
```

Then, for each host you wish to authorize, its public key must be added to the authorized_keys file on the host to which it needs to connect. The public key has a .pub extension. The added hosts must have VOLTTRON instances installed, with the Restricted code installed and enabled:

4. Copy host information securely:

```
$ scp otherhost.example.com:~/.volttron/ssh/id_rsa.pub ./otherhost.pub
```

5. Append host key(s) to authorized_keys file in $VOLTTRON_HOME/ssh:

```
$ cat otherhost.pub >> ~/.volttron/ssh/authorized_keys
```

6.2.3 Enabling Resource Monitoring

The following steps will enable resource monitoring feature within the VOLTTRON Restricted software. From a terminal, in the base VOLTTRON directory, enter the following commands:

1. Run cgroup setup script:

```
# volttron/scripts/cgroup_setup.sh
```

2. Create cgroups:

```
# env/bin/volttron-ctl create-cgroups -u $USER
```

6.2.4 Configuring Resource Monitoring

The VOLTTRON Restricted module provides additional protection against an agent consuming too many resources to the point of the host system becoming unresponsive or unstable. The resource monitor uses Linux control groups (or cgroups) to limit the CPU cycles and memory an individual agent may consume, preventing its possible overconsumption from adversely affecting other agents and services on the system. When a request is made to move an agent to a new platform, part of the validation of the agent includes checking its execution requirements against resources currently available on the system. If the resources are available and the agent has passed all other validation, the agent will be executed and retain those resource guarantees throughout its lifetime on that platform. If the agent, however, requests memory or CPU cycles that are not available, its move request is denied (move refers to the use of the agent mobility feature, see Section 6.2.2. For agent mobility use-case documentation and an example agent that utilizes the mobility feature, visit the VOLTTRON Wiki at https://github.com/VOLTTRON/volttron/wiki/Ping-Pong-Agent) and it will not execute on the requested platform.

Once an agent has been assigned resources, it is the responsibility of that agent to manage use of its resources. While an agent may exceed its resource guarantees when system utilization is low, when resources given to other agents are required, an agent exceeding the use in its contract may be terminated.

6.2.4.1 Execution Requirements

The execution requirements are specified as a JSON formatted document embedded in the agent during initial provisioning and takes the following form:

```
{
  "requirements": {
    "cpu.bogomips": 100,
    "memory.soft_limit_in_bytes": 2000000
  }
}
```

The contract *must* contain the `requirements` object, specifying the soft requirements, and might optionally specify a hard_requirements object.

6.2.4.1.1 Soft Requirements

Requirements are considered *soft* on the platform because they change depending on the number of agents and other services that are running on the system. They may also be negotiated dynamically in a future release. A list of the current resources that may be reserved are as follows:

- **cpu.bogomips -** The CPU requirements of an agent indicated as either an exact integer (N >= 1) in MIPS (millions of instructions per second) or a floating-point percentage (0.0 < N < 1.0) of the total available BogoMips on a system. BogoMips is a rough calculation performed at system boot indicating the likely number of calculations a system may perform each second.

- **memory.soft_limit_in_bytes -** The maximum amount of random access memory (RAM) an agent requires to perform its tasks, measured in bytes and given as an integer. Additional resources may be added in a future release.

6.2.4.1.2 Hard Requirements

Hard requirements are based on system attributes that are very unlikely to change except after a system reboot. It is rare that an agent would need to set hard requirements and is usually only necessary for architecture-specific code. Each hard requirement is tested for a match.

- kernel.name **-** Kernel name as given by uname.
- kernel.release **-** Kernel release as given by uname.
- kernel.version **-** Kernel version as given by uname.
- architecture **-** Kernel architecture as given by uname.
- os **-** Always 'GNU/Linux'
- platform.version **-** Version of VOLTTRON in use.
- memory.total **-** Total amount of memory on the system in bytes.
- bogomips.total **-** Total of all BogoMips reported for all processors on the system.

6.2.4.1.3 Signing and Launching Agents with VOLTTRON Restricted Enabled

If VOLTTRON Restricted is installed and the security features are enabled (Section 6.2.4.1.4), all agents must be signed prior to launching them. The following steps will describe how to sign an agent and will use the Listener agent as an example (launching the Listener agent without VOLTTRON Restricted enabled is documented in Section 2.6 of this document). From a terminal, in the base VOLTTRON directory, enter the following commands:

1. Package the agent:

```
$ volttron-pkg package examples/ListenerAgent
```

2. Sign the agent as creator (resource_contract is a text file containing the hardware and software requirements for the agent, see Section 6.2.4):

```
$ volttron-pkg sign --creator --contract resource_contract
~/.volttron/packaged/listeneragent-3.0-py2-none-any.whl
```

3. Sign the agent as SOI:

```
$ volttron-pkg sign -soi ~/.volttron/packaged/listeneragent-3.0-py2-none-
any.whl
```

4. Sign the agent as initiator:

```
$ volttron-pkg sign --initiator --config-file examples/ListenerAgent/config
~/.volttron/packaged/listeneragent-3.0-py2-none-any.whl
```

5. Set the configuration file:

```
$ volttron-pkg configure ~/.volttron/packaged/listeneragent-3.0-py2-none-
any.whl examples/ListenerAgent/config
```

6. Install agent into platform (with the platform running):

```
$ volttron-ctl install ~/.volttron/packaged /volttron_wheels/listeneragent-
3.0 -py2-none-any.whl
```

- 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 /tmp/volttron_wheels/weatheragent-0.1-py2-none-any.whl as a9d67c55-7f58-4591-80af-3c1ff8a81740 listeneragent-3.0`

7. Start the agent:

```
$ volttron-ctl start --name listeneragent-3.0
```

8. Verify that agent is running:

```
$ volttron-ctl status
```

6.2.4.1.4 Disable VOLTTRON Restricted After Installation

If one wants to disable all or specific components within the Restricted module (security, resource monitoring, and agent mobility), simply add the following lines (or create the text file and add the lines) to the platform configuration file in the $VOLTTRON_HOME directory (Section 2.5):

- Create or edit (`~/.volttron/config`) and add any or all of the following lines:

```
# Disable the VOLTTRON agent verification feature.
no-verify
# Disable the VOLTTRON resource monitoring features.
no-resource-monitor
# Disable the VOLTTRON mobility features.
no-mobility
# Disable all VOLTTRON restricted features.
no-restricted
```

This can be very useful when developing new applications or agents. In addition, managing of VOLTTRON Restricted features is not recommended within the Eclipse IDE, so disabling Restricted features while developing within Eclipse is recommended. To re-enable features, delete config file or add '#' without quotes at the beginning of the line for the feature you want to re-enable. For example, to disable resource monitoring but re-enabling security and mobility, the config file would contain the following:

```
#no-verify
no-resource-monitor
#no-mobility
```

7 Legacy Features

Many components of VOLTTRON were changed for version 3.0 making this a much more significant upgrade than from 1.0 to 2.0. This had the side-effect of making some previously integral components either optional or redundant. They are included in this section for deployments which wish to continue using them.

7.1 Multi-Building (Multi-Node) Communication

Reason for deprecation: The service this agent provides has been largely superseded by the capabilities of the VIP router. Agents can now directly publish and subscribe to remote message buses.

Multi-building (or multi-node) messaging is implemented as a service-style agent. Its use is optional and it can be enabled/disabled by simply enabling/disabling the MultiBuilding service agent. It is easily configured using the service configuration file and provides several new topics for use in the local agent exchange bus.

7.1.1 Configuration for Multi-Node Communication

The service configuration file may contain the declarations below:

- **building-publish-address:** A ØMQ address on which to listen for messages published by other nodes. Defaults to 'tcp://0.0.0.0:9161'.
- **building-subscribe-address:** A ØMQ address on which to listen for messages subscribed to by other nodes. Defaults to 'tcp://0.0.0.0:9160'.
- **public-key, secret-key:** Curve keypair (create with zmq.curve_keypair()) to use for authentication and encryption. If not provided, all communications will be unauthenticated and unencrypted.
- **hosts:** A mapping (dictionary) of building names to publish/subscribe addresses. Each entry is of the form:
  - `"CAMPUS/BUILDING": {"pub": "PUB_ADDRESS", "sub": "SUB_ADDRESS", "public-key": "PUBKEY", "allow": "PUB_OR_SUB"}`
  - CAMPUS/BUILDING: building for which the given parameters apply
  - PUB_ADDRESS: ØMQ address used to connect to the building for publishing
  - SUB_ADDRESS: ØMQ address used to connect to the building subscriptions
  - PUBKEY: curve public key of the host used to authenticate incoming connections
  - PUB_OR_SUB: the string "pub" to allow publishing only or "sub" to allow both publish and subscribe
- **cleanup-period**: Frequency, in seconds, to check for and close stale connections. Defaults to 600 seconds (10 minutes).
- **uuid**: A UUID to use in the Cookie header. If not given, one will be automatically generated.

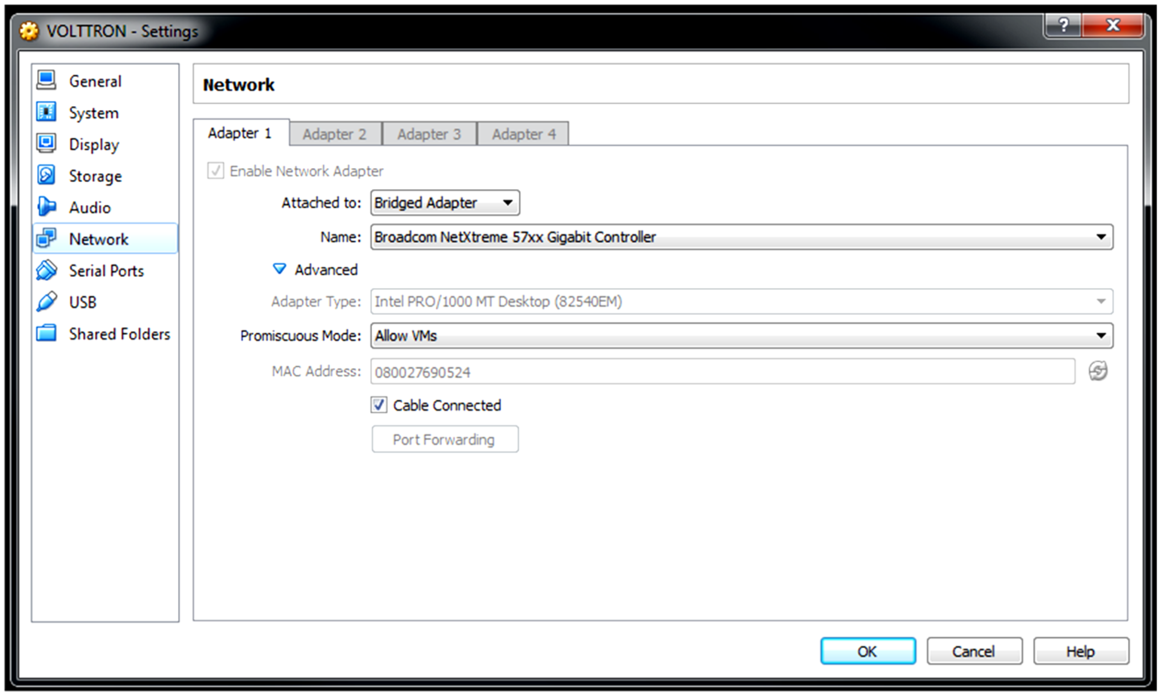
When using a VM to run Linux and VOLTTRON, the VM must be configured to use a bridged adapter. This will allow the VM to receive a unique network IP. From the VirtualBox Settings window on the Network tab, configure the VM as follows (Figure 96):

- For Attached to use: Bridged Adapter
- For Name use: default value (VirtualBox will typically auto-detect your network controller)
- For Promiscuous Mode use: Allow VMs
- To obtain a new IP, open a terminal and enter the following commands (if using Wi-Fi, eth0 below should be replaced by wlan0):

```
# ip link set eth0 down
# ip link set eth0 up
```

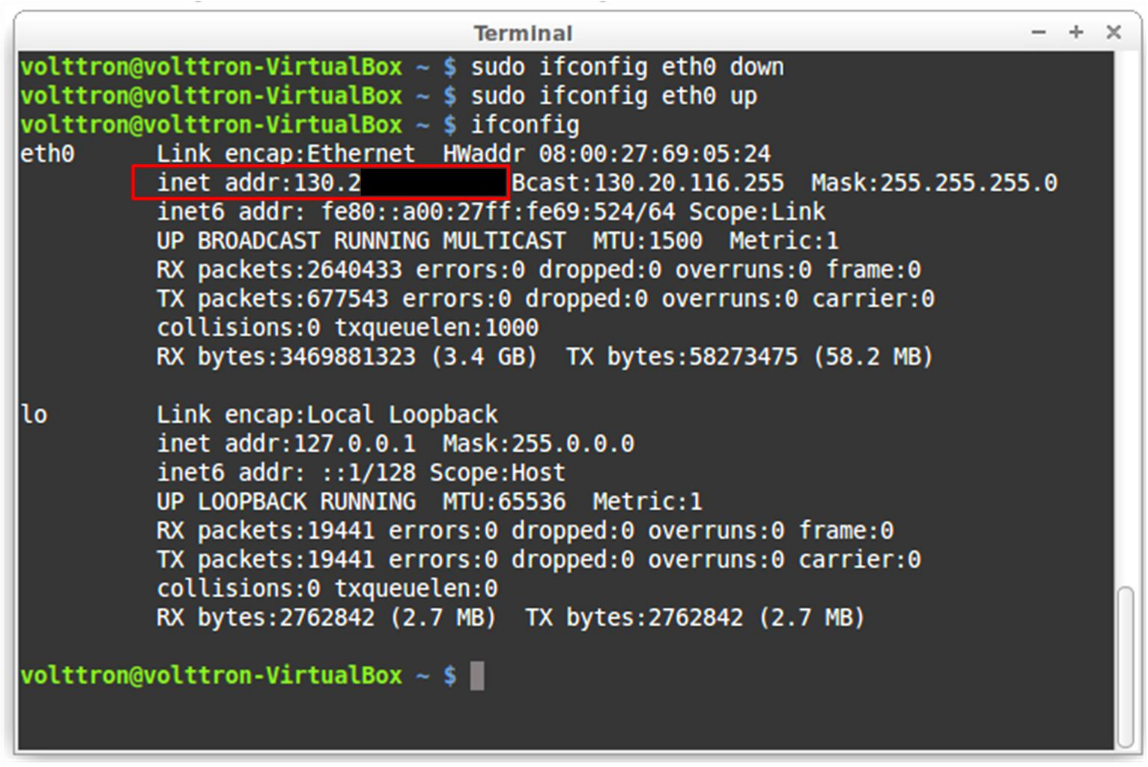
- To view the IP, enter the following command (the IP will be listed under eth0 or wlan0, depending on whether the network connection is wireless or wired, as shown in Figure 97 boxed in red):

```
# ip addr
```



☑ Enable Network Adapter
Attached to: Bridged Adapter
Name: Broadcom NetXtreme 57xx Gigabit Controller
Adapter Type: Intel PRO/1000 MT Desktop (82540EM)
Promiscuous Mode: Allow VMs
MAC Address: 080027690524
☑ Cable Connected

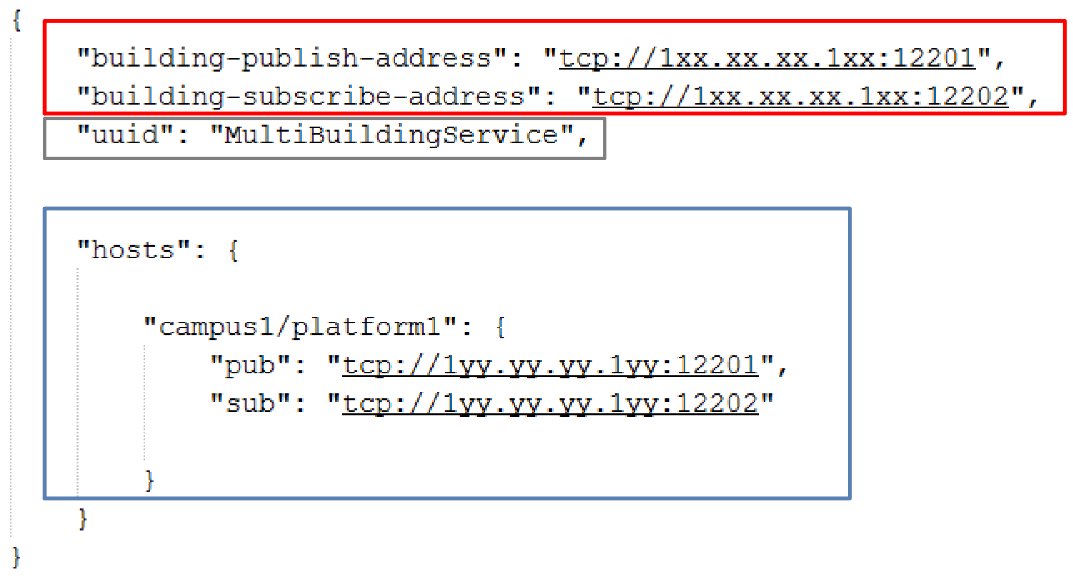
Figure 96: Configuration of VirtualBox VM for Multi-Node Communication



```
volttron@volttron-VirtualBox ~ $ sudo ifconfig eth0 down
volttron@volttron-VirtualBox ~ $ sudo ifconfig eth0 up
volttron@volttron-VirtualBox ~ $ ifconfig
eth0      Link encap:Ethernet  HWaddr 08:00:27:69:05:24
          inet addr:130.2[redacted]Bcast:130.20.116.255  Mask:255.255.255.0
          inet6 addr: fe80::a00:27ff:fe69:524/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:2640433 errors:0 dropped:0 overruns:0 frame:0
          TX packets:677543 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:3469881323 (3.4 GB)  TX bytes:58273475 (58.2 MB)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:19441 errors:0 dropped:0 overruns:0 frame:0
          TX packets:19441 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:2762842 (2.7 MB)  TX bytes:2762842 (2.7 MB)

volttron@volttron-VirtualBox ~ $
```

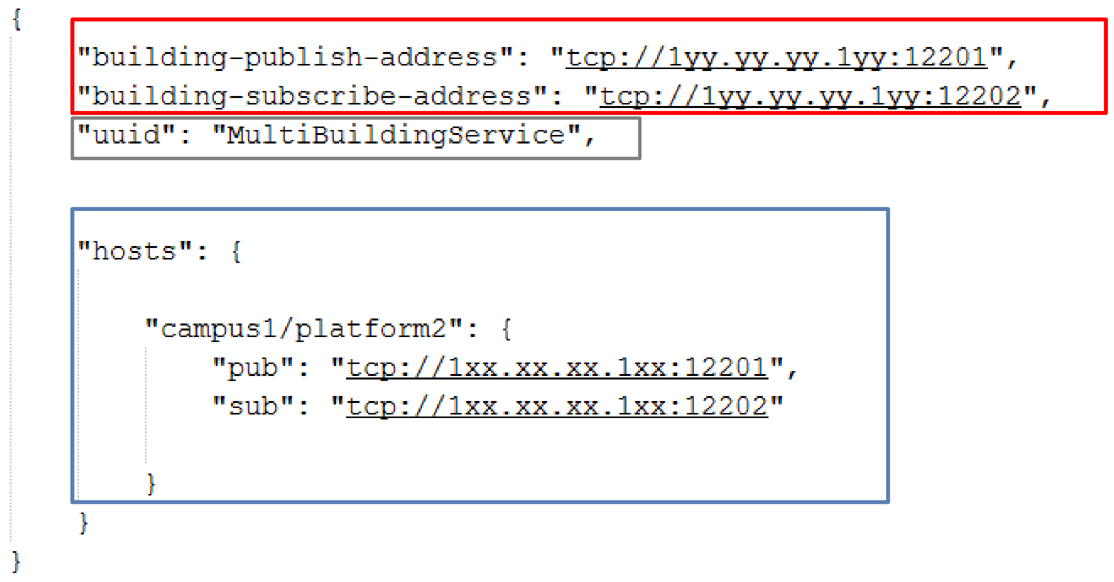
Figure 97: Identifying IP Address on VirtualBox

Figure 98 and Figure 99 shows the configuration file for two instances of the MultiBuilding agent, each running on a separate instance of VOLTTRON for a different campus and different building (the agents could run on the same campus and building and only run on separate instances of VOLTTRON). The top red boxes in Figure 98 and Figure 99 give information about the agent's native message bus, IP, and preferred port. The bottom red box in Figure 98 and Figure 99 provides identifying information for the companion VOLTTRON instance to allow the agent to publish on its message bus. The blue box gives the MultiBuilding agent information about any other platforms that it will allow publishing on its message bus. These configurations will enable two-way communication between the two VOLTTRON instances.



```
{
    "building-publish-address": "tcp://1xx.xx.xx.1xx:12201",
    "building-subscribe-address": "tcp://1xx.xx.xx.1xx:12202",
    "uuid": "MultiBuildingService",

    "hosts": {

        "campus1/platform1": {
            "pub": "tcp://1yy.yy.yy.1yy:12201",
            "sub": "tcp://1yy.yy.yy.1yy:12202"

        }
    }
}
```

Figure 98: Example of MultiBuilding Agent Configuration File for "campus1/platform1"



```
{
    "building-publish-address": "tcp://1yy.yy.yy.1yy:12201",
    "building-subscribe-address": "tcp://1yy.yy.yy.1yy:12202",
    "uuid": "MultiBuildingService",

    "hosts": {

        "campus1/platform2": {
            "pub": "tcp://1xx.xx.xx.1xx:12201",
            "sub": "tcp://1xx.xx.xx.1xx:12202"

        }
    }
}
```

Figure 99: Example of MultiBuilding Agent Configuration File for "campus1/platform2"

7.1.2 Using Data Published From Another VOLTTRON

The MultiBuilding agent enables the communication between separate VOLTTRON instances but does not actually facilitate that communication. The following section will illustrate a simple agent that will publish read data from its message bus and publish that data to any number of other platform message bus (the appropriate configuration of the MultiBuilding agent is required, as detailed in the previous section).

```
@matching.match_exact(topics.DEVICES_VALUE(point='all', **device_path))
def publish_signal(self, topic, headers, message, match):
    '''
    Publish Data on other Bus
    '''
    print "publishing self data to other platform"
    msg = jsonapi.loads(message[0])
        for receiver in self.receiving_platforms:
            print ('Sending data to: ' + receiver)
            self.publish_json(
                topics.BUILDING_SEND(campus=config.get('campus'),
                                     building=receiver,
                                     topic=self.topic),
                {COOKIE: self.uuid}, msg)
```

The above block of code collects all the data for a configured device and publishes the data to its companion VOLTTRON platform message bus on the BUILDING_SEND topic. The following shows the configuration file for this agent (data for “campus1/platform1/device1” must be published to the message bus for platform1 via the sMAP driver or some other mechanism in order for this example to work):

```
{
    "agentid": "MultiNodePublisher",
    "receiving_platforms": ["platform2"],
    "campus": "campus1",
    "building": "platform1",
    "device": "device1"
}
```

The receiving platform(s) must match the platform(s) configured in the MultiBuilding agent configuration file in the “hosts” section (e.g., “platform1” will publish to “platform2” in this example using the MultiNodePublisher). The following is a complete example agent that publishes to another VOLTTRON message bus:

```
import logging
import sys
from zmq.utils import jsonapi
from volttron.platform.agent import BaseAgent, PublishMixin, periodic
from volttron.platform.agent import utils, matching
from volttron.platform.messaging import headers as headers_mod
from volttron.platform.messaging import topics
from volttron.platform.messaging.headers import COOKIE

def cookie_headers(request, **headers):
    if request:
        try:
            headers[COOKIE] = request[COOKIE]
        except KeyError:
            pass
    return headers

def MultiNodePublisher(config_path, **kwargs):
    '''
    Publish Device Information from Multi-Node
    communication
    '''
    config = utils.load_config(config_path)
    device_path = rtu_path = dict((key, config[key])
                                  for key in ['campus',
                                              'building',
                                              'unit'])

    class Agent(PublishMixin, BaseAgent):
        def __init__(self,**kwargs):
            super(Agent, self).__init__(**kwargs)
            self.topic = config.get('topic', 0)
            self.receiving_platforms = config.get('receiving_platforms',0)
            self.uuid = config.get('agentid')

        def setup(self):

            super(Agent, self).setup()

        @matching.match_exact(topics.DEVICES_VALUE(point='all', **device_path))
        def publish_signal(self, topic, headers, message, match):
            '''
            Publish Data on other Bus
            '''
            print "publishing self data to other platform"
            msg = jsonapi.loads(message[0])
                for receiver in self.receiving_platforms:
                    print ('Sending data to: ' + receiver)
                    self.publish_json(
                        topics.BUILDING_SEND(campus=config.get('campus'),
                                             building=receiver,
                                             topic=self.topic),
                        {COOKIE: self.uuid}, msg)

    Agent.__name__ = 'MultiNodePublisher'
    return Agent(**kwargs)

def main(argv=sys.argv):
    '''Main method called by the eggsecutable.'''
    utils.default_main(MultiNodePublisher,
                       description='Multi-Node Example Publisher',
                       argv=argv)

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

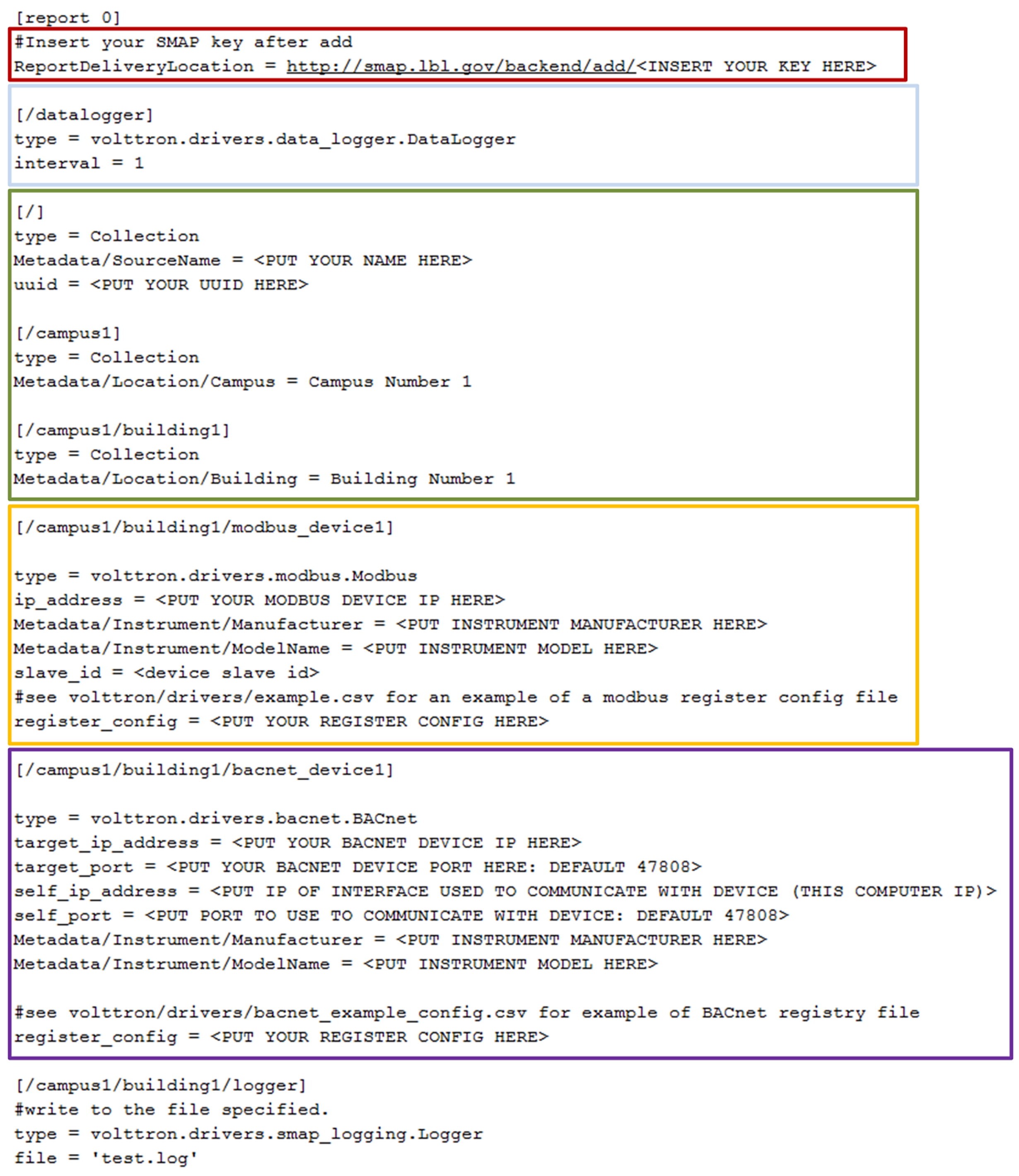
7.2 sMAP Driver: Legacy Device Communication

The legacy sMAP driver is still available for use within VOLTTRON. The sMAP driver is not as modular or extensible as the new BACnet and Modubs drivers, and there will be no further development or bug fixes for the sMAP driver. In the future, if the sMAP driver becomes unusable because of system and

VOLTTRON platform updates, it will be removed from the platform. The following section describes the configuration and use of the sMAP driver.

The legacy sMAP driver requires BACnet and/or Modbus registry files (for communication with BACnet and Modbus devices, respectively). Information on the format for these registry files is found in section configuring the Modbus/BACnet registry file(s) and the sMAP configuration file, the driver can now be launched. The driver can be run as a VOLTTRON agent or by directly calling Twisted. Running the device driver as an agent is the same process as the building and launching of the Listener agent or the Weather agent.

1. Configure the sMAP configuration file, as shown in Figure 100. If configuring the sMAP driver for device communication using only one of the supported protocols (BACnet or Modbus), then comment out (add # to the beginning of each line) the section for the unused communication protocol. For example, if Modbus is not used for device communication, comment out lines in the configuration file that correspond to the yellow boxed parameter in Figure 100.



```
[report 0]
#Insert your SMAP key after add
ReportDeliveryLocation = http://smap.lbl.gov/backend/add/<INSERT YOUR KEY HERE>

[/datalogger]
type = volttron.drivers.data_logger.DataLogger
interval = 1

[/]
type = Collection
Metadata/SourceName = <PUT YOUR NAME HERE>
uuid = <PUT YOUR UUID HERE>

[/campus1]
type = Collection
Metadata/Location/Campus = Campus Number 1

[/campus1/building1]
type = Collection
Metadata/Location/Building = Building Number 1

[/campus1/building1/modbus_device1]

type = volttron.drivers.modbus.Modbus
ip_address = <PUT YOUR MODBUS DEVICE IP HERE>
Metadata/Instrument/Manufacturer = <PUT INSTRUMENT MANUFACTURER HERE>
Metadata/Instrument/ModelName = <PUT INSTRUMENT MODEL HERE>
slave_id = <device slave id>
#see volttron/drivers/example.csv for an example of a modbus register config file
register_config = <PUT YOUR REGISTER CONFIG HERE>

[/campus1/building1/bacnet_device1]

type = volttron.drivers.bacnet.BACnet
target_ip_address = <PUT YOUR BACNET DEVICE IP HERE>
target_port = <PUT YOUR BACNET DEVICE PORT HERE: DEFAULT 47808>
self_ip_address = <PUT IP OF INTERFACE USED TO COMMUNICATE WITH DEVICE (THIS COMPUTER IP)>
self_port = <PUT PORT TO USE TO COMMUNICATE WITH DEVICE: DEFAULT 47808>
Metadata/Instrument/Manufacturer = <PUT INSTRUMENT MANUFACTURER HERE>
Metadata/Instrument/ModelName = <PUT INSTRUMENT MODEL HERE>

#see volttron/drivers/bacnet_example_config.csv for example of BACnet registry file
register_config = <PUT YOUR REGISTER CONFIG HERE>

[/campus1/building1/logger]
#write to the file specified.
type = volttron.drivers.smap_logging.Logger
file = 'test.log'
```

Figure 100: An Example sMAP Configuration File

The required information is shown below:

- The sMAP URL (location) and sMAP key
- sMAP metadata information
- /campus/building - path for publishing and subscribing to device data on the VOLTTRON message bus

- Modbus configurable parameters:
  - Device IP address and slave identification (if applicable)
  - Location of Modbus registry file
  - Desired device metadata
- BACnet configurable parameters:
  - Device IP address
  - Location of BACnet registry file
  - Desired device metadata
- The path to the data logger scripts. This information shows the sMAP driver where to find the logging code. The logging code allows an agent to publish information to the logging topic, where that information will then be pushed to sMAP.

2. Save the file within the transactional network workspace. Subsequent examples from this document will assume the file is saved as (`~/volttron/smap.ini`).
3. Edit the TwistdLauncher configuration file (`<project directory>/services/deprecated/TwistdLauncher/twistd.launcher`). The file should contain the following entry where *smap.ini* is the name of the sMAP configuration file described in Figure 100:

```
{
    "exec": "~/volttron/env/bin/twistd -n smap  ~/volttron/<smap.ini>"
}
```

From the base VOLTTRON directory, enter the following commands in a terminal window:

4. Run pack_install script on TwistdLauncher agent:

   ```
   ./scripts/core/pack_install.sh services/deprecated/TwistdLauncher
   services/deprecated/TwistdLauncher/twistd.launcher twistd
   ```

   - 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/launcheragent-0.1-py2-none-any.whl as cbbcdb21-3a66-474b-984f-0f31a94ad6belauncheragent-0.1`**

5. Start the agent:

   ```
   $ volttron-ctl start --tag twistd
   ```

6. Verify that agent is running:

   ```
   $ volttron-ctl status
   $ tail volttron.log
   ```

If changes are made to the TwistdLauncher 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:

```
$ volttron-ctl stop --tag twistd

$ volttron-ctl remove --tag twistd
```

Then rebuild and start the updated agent.