



U.S. DEPARTMENT OF
ENERGY

PNNL-23879

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

VOLTTRON 2.0: User Guide

RG Lutes
S Katipamula
BA Akyol
ND Tenney

JN Haack
KE Monson
BJ Carpenter

November 2014



Pacific Northwest
NATIONAL LABORATORY

*Proudly Operated by **Battelle** Since 1965*

DISCLAIMER

United States Government. Neither the United States Government nor any agency thereof, nor Battelle Memorial Institute, nor any of their employees, makes **any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights.** Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof, or Battelle Memorial Institute. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

PACIFIC NORTHWEST NATIONAL LABORATORY

operated by

BATTELLE

for the

UNITED STATES DEPARTMENT OF ENERGY

under Contract DE-AC05-76RL01830

Printed in the United States of America

Available to DOE and DOE contractors from the

Office of Scientific and Technical Information,

P.O. Box 62, Oak Ridge, TN 37831-0062;

ph: (865) 576-8401, fax: (865) 576-5728

email: reports@adonis.osti.gov

Available to the public from the National Technical Information Service,
U.S. Department of Commerce, 5285 Port Royal Rd., Springfield, VA 22161

ph: (800) 553-6847, fax: (703) 605-6900

email: orders@ntis.fedworld.gov

online ordering: <http://www.ntis.gov/ordering.htm>



This document was printed on recycled paper.

(8/00)

VOLTTRON: User Guide

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

November 2014

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, Building Technologies Office 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 consists of VOLTTRON™ agent execution software, 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

Summary	iii
1 Introduction	1
1.1 Background	1
1.2 Transactional Network Platform Overview	1
1.3 VOLTTRON Overview.....	3
1.3.1 VOLTTRON	3
1.3.2 VOLTTRON Services.....	3
2 Deployment of VOLTTRON.....	6
2.1 Installing Linux Virtual Machine.....	6
2.2 Running and Configuring Virtual Machine.....	7
2.3 Installing Required Software.....	13
2.4 Installing the sMAP Server (Optional).....	15
2.5 Checking Out Transactional Network from Repository.....	15
2.6 Building the VOLTTRON Platform	15
2.7 VOLTTRON Home Directory and Configuration	17
2.8 Launching the Listener Agent	17
2.9 Launching the Weather Agent.....	18
2.9.1 Obtaining a Developer Key from WeatherUnderground	18
2.9.2 Configuring WeatherAgent with Developer Key and Location.....	20
2.9.3 Launching the Weather Agent.....	21
2.10 Configuring and Launching sMAP Driver.....	22
2.10.1 Configuring sMAP driver.....	23
2.10.2 Launching the Driver	25
2.11 Configuring and Launching the Actuator Agent.....	27
2.11.1 Configuring the Actuator Agent.....	27
2.11.2 Scheduling a Task	28
2.11.3 Canceling a Task	29
2.11.4 Actuator Error Reply	29
2.11.5 Task Preemption and Schedule Failure	30
2.11.6 Actuator Agent Interaction.....	31
2.11.7 Device Schedule State Announcements.....	32
2.11.8 Launching the Actuator Agent	32
2.11.9 Tips for Working with the Actuator Agent	33
2.12 Multi-Building (Multi-Node) Communication	35
2.12.1 Configuration for Multi-Node Communication	35

	2.12.2 Using Data Published From Another VOLTTRON.....	38
3	Sample Applications/Agents	40
	3.1 Automated Fault Detection and Diagnostic Agent.....	40
	3.1.1 Configuring the AFDD Agent.....	40
	3.1.2 Launching the AFDD Agent	43
	3.2 The Demand Response (DR) Agent.....	46
	3.2.1 Configuring DR Agent	47
	3.2.2 OpenADR (Open Automated Demand Response).....	49
	3.2.3 DR Agent Output to sMAP	50
	3.2.4 Launching the Demand Response Agent	50
	3.3 Other VOLTTRON Applications.....	52
	3.3.1 Autonomous Control of Rooftop Units	52
	3.3.2 Supermarket Refrigeration Application	54
	3.3.3 Renewable Energy Integration Application	55
	3.3.4 Lighting Diagnostic Application.....	55
	3.3.5 Baseline Load Shape Application	57
	3.3.6 Measurement and Verification Application	58
	3.3.7 Smart Monitoring and Diagnostic System Application.....	59
	3.3.8 Analytical Hierarchy Process for Load Curtailment Application	61
4	Agent Development in VOLTTRON	63
	4.1 Example Agent Walkthrough.....	63
	4.2 Explanation of Listener Agent	63
	4.3 Agent Development in Eclipse.....	64
	4.3.1 Installing Eclipse.....	64
	4.3.2 Installing Pydev and EGit Eclipse Plug-ins	65
	4.3.3 Checkout VOLTTRON Project.....	68
	4.3.4 Configuring Eclipse.....	76
	4.3.5 Running the VOLTTRON Platform and Agents.....	78
	4.3.6 Agent Creation Walkthrough	82
	4.3.7 Adding Additional Features to the TestAgent.....	86
5	New VOLTTRON Features (AKA VOLTTRON Restricted).....	90
	5.1 Installation of VOLTTRON Restricted.....	90
	5.2 Enabling and Configuring VOLTTRON Restricted Software	90
	5.2.1 Creating Required Security Certificates.....	92
	5.2.2 Enabling Agent Mobility Feature.....	93
	5.2.3 Enabling Resource Monitoring	93
	5.2.4 Configuring Resource Monitoring	93

Figures

Figure 1: Illustration of the various components of the transactional network.....	2
Figure 2: VirtualBox download page.....	6
Figure 3: Linux Mint download page	7
Figure 4: Creating a Virtual Machine	8
Figure 5: Selecting Memory Size.....	8
Figure 6: Selecting Storage Size	9
Figure 7: Creating Virtual Hard Drive.....	9
Figure 8: Selection of type of hard drive	10
Figure 9: Creating Virtual Hard Drive (continued).....	10
Figure 10: Selection of display type	11
Figure 11: Selection of processor parameter.....	11
Figure 12: Enable bidirectional copy and paste (Shared Clipboard) and Drag'n'Drop.....	12
Figure 13: Loading Linux image	12
Figure 14: Installing Linux Mint Operating System.....	13
Figure 15: Linux Mint Terminal Window	14
Figure 16: Linux Mint Terminal Window After Successful Completion of the 'bootstrap' Script.....	16
Figure 17: Linux Mint Terminal After Successfully Activating and Starting the VOLTTRON Platform	16
Figure 18: Sample Output from the Listener Agent.....	18
Figure 19: WeatherUnderground Website	19
Figure 20: Setting up a Developer Account.....	19
Figure 21: Creating a WeatherUnderground API Key.....	20
Figure 22: WeatherUnderground API Key	20
Figure 23: Entering the WeatherUnderground Developer Key	21
Figure 24: Entering Zip Code for the Location.....	21
Figure 25: Example Output from the Weather Agent.....	22
Figure 26: An Example Modbus Registry File	23
Figure 27: An Example BACnet Registry File	23
Figure 28: An Example sMAP Configuration File	24
Figure 29: Setting Path to Message Bus and Environment Variables.....	26
Figure 30: Example Actuator Agent Configuration File.....	27
Figure 31: Configuration of VirtualBox VM for Multi-Node Communication.....	36
Figure 32: Identifying IP Address on VirtualBox.....	36
Figure 33: Example of MultiBuilding Agent Configuration File for "campus1/platform1"	37
Figure 34: Example of MultiBuilding Agent Configuration File for "campus1/platform2"	37
Figure 35: Example AFDD Agent Configuration File.....	41
Figure 36: File Selection Dialog Box when Inputting Data in a csv File	45
Figure 37: Sample of CSV Data for AFDD Agent	46

Figure 38: Example Configuration File for the DR Agent	48
Figure 39: Autonomous Control Application	52
Figure 40: Autonomous Control Applications Deployment Architecture	53
Figure 41: Autonomous Control Application Software Components.....	53
Figure 42: Supermarket Refrigeration Application Deployment Architecture.....	54
Figure 43: Renewable Integration Application Deployment Architecture	55
Figure 44: Lighting Diagnostic Overview	56
Figure 45: Deployment Architecture for the Lighting Diagnostic Application.....	57
Figure 46: Regression Model Developed by the Baseline Load Shape Application	58
Figure 47: Basic Methodology of the SMDS Application.....	60
Figure 48: Deployment Architecture for the SMDS Application	61
Figure 49: Eclipse Desktop File.....	65
Figure 50: Installing Eclipse EGit Plug-in.....	66
Figure 51: Installing Eclipse Egit Plug-in (continued)	66
Figure 52: Installing Eclipse Egit Plug-in (continued)	67
Figure 53: Installing Eclipse PyDev Plug-in.....	68
Figure 54: Checking VOLTTRON with Eclipse From Local Source.....	69
Figure 55: Checking VOLTTRON with Eclipse from Local Source (continued)	69
Figure 56: Checking VOLTTRON with Eclipse from Local Source (continued)	70
Figure 57: Checking VOLTTRON with Eclipse from Local Source (continued)	70
Figure 58: Checking Out VOLTTRON with Eclipse from Local Source (continued)	71
Figure 59: Checking Out VOLTTRON with Eclipse from Local Source (continued)	71
Figure 60: Checking Out VOLTTRON with Eclipse from Local Source (continued)	72
Figure 61: Checking Out VOLTTRON with Eclipse from GitHub.....	73
Figure 62: Checking Out VOLTTRON with Eclipse from GitHub (continued)	73
Figure 63: Checking Out VOLTTRON with Eclipse GitHub (continued)	74
Figure 64: Checking Out VOLTTRON with Eclipse from GitHub (continued)	74
Figure 65: Checking Out VOLTTRON with Eclipse from GitHub (continued)	75
Figure 66: Checking Out VOLTTRON with Eclipse from GitHub (continued)	75
Figure 67: Checking Out VOLTTRON with Eclipse from GitHub (continued)	76
Figure 68: Configuring PyDev.....	77
Figure 69: Configuring PyDev (Continued)	77
Figure 70: Setting as PyDev Project	78
Figure 71: Setting PyDev Perspective in Eclipse.....	78
Figure 72: Running VOLTTRON Platform, Setting Up a Run Configuration.....	79
Figure 73: Running VOLTTRON Platform, Setting Up a Run Configuration (Continued).....	80
Figure 74: Running VOLTTRON Platform, Console View on Successful Run	80
Figure 75: Running the Listener Agent, Setting Up a Run Configuration.....	81
Figure 76: Configuring the Listener Agent, Setting Up a Run Configuration (Continued).....	82

Figure 77: Listener Agent Output on Eclipse Console	82
Figure 78: Creating an Agent Test Folder	82
Figure 79: TestAgent Output in “volttron.log”	85
Figure 80: Console Output for TestAgent.....	86
Figure 81: TestAgent Output when Subscribing to Weather Topic.....	86
Figure 82: Structure for the Agent Signing Security Feature in VOLTTRON Restricted.....	91

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 platform consists of VOLTTRON™ agent execution software, 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.).

To encourage development and growth of the transactional network platform 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 transactional network platform.

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 transactional network platform and VOLTTRON, launch of agents (applications) on the platform, and help with development of new agents within the platform. 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 validating 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 transactional network platform, VOLTTRON connects devices (RTUs, building systems, 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 to ease development and deployment of agents into the environment.

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 both posts data to the sMAP historian and publishes data to the message bus on a topic for each device; it also provides a means to send control commands from various 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.

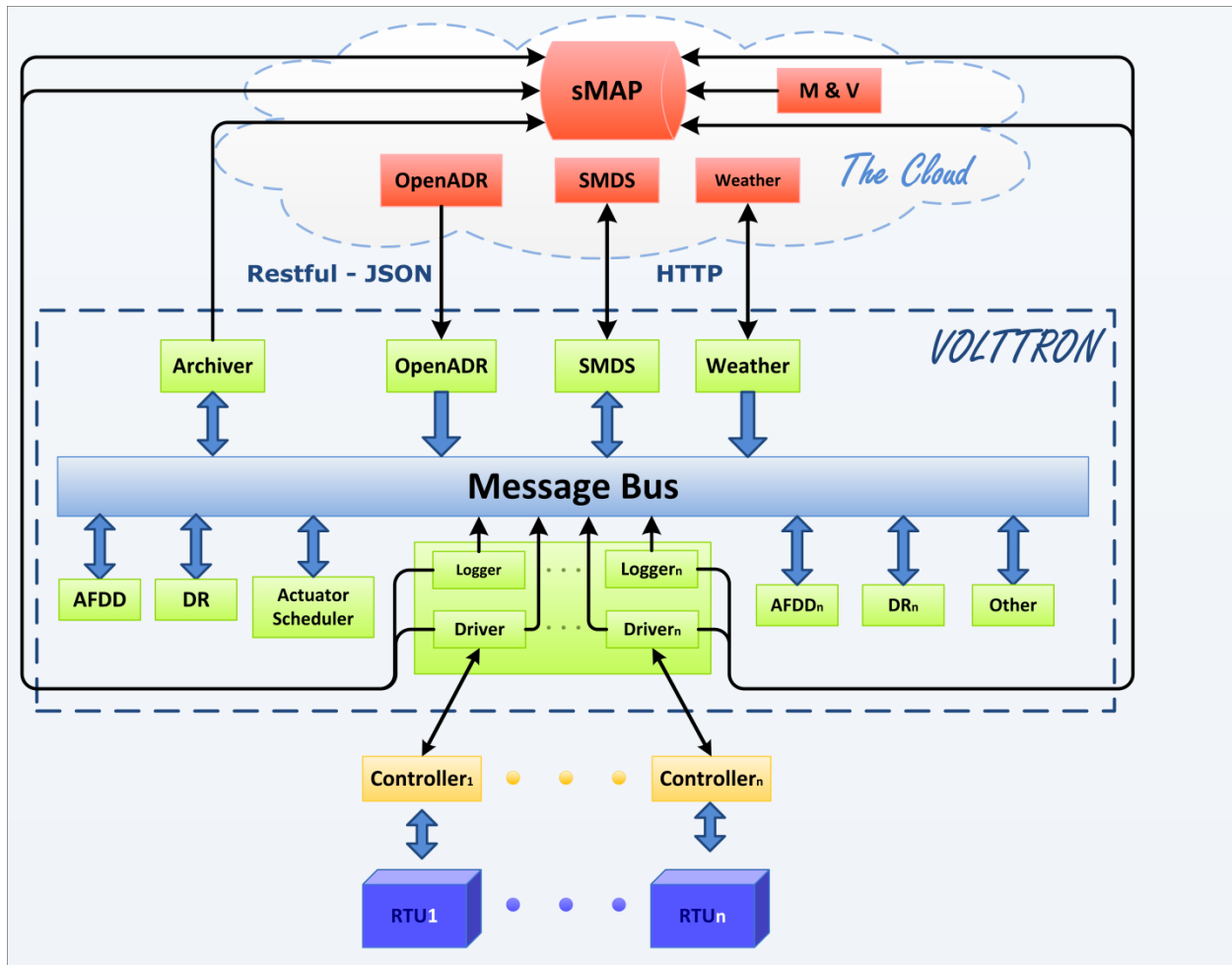


Figure 1: Illustration of the various components of the transactional network

The Archiver, in Figure 1, allows agents to request data from sMAP over the message bus. This isolates agents from the historian and allows the platform the flexibility of using potentially different data storage solutions. For example, because sMAP does not accept string data, a separate database could be used, and the interface to the agents would remain unchanged.

Agents and platform services shown in Figure 1 communicate with each other via the message bus using publish/subscribe 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 get as general or as specific as they want 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

The transactional network platform 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 tools (software) that VOLTTRON utilizes:

- sMAP: VOLTTRON utilizes sMAP¹ for data storage and retrieval. The VOLTTRON Modbus driver publishes data from devices to the platform and also stores the data in the sMAP historian. During development of this driver, the VOLTTRON team contributed error reports and resolved a bug in the sMAP software.
- Drivers for sMAP are written using another open source product called twistd². Twistd is an event-based networking engine.
- 0MQ: The VOLTTRON message bus, which allows agents and services to exchange data, uses Zero MQ³. This free software is used by National Aeronautics and Space Administration (NASA), Cisco, etc. to provide scalable, reliable, and fast communication. The VOLTTRON team have been active members of this open source software community, reporting bugs as well as developing code to fix these software bugs.
- PyModbus: The VOLTTRON Modbus⁴ driver builds on PyModbus⁵, which enables Python code to easily interact with Modbus devices.
- Other open source Python modules being used are:
 - 'avro', 'configobj', 'gevent', 'flexible-jsonrpc', 'numpy', 'posix-clock', 'pyopenssl', 'python-dateutil', 'requests', 'setuptools', 'simplejson', 'zope.interface'

1.3.2 VOLTTRON Services

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

¹ <http://www.cs.berkeley.edu/~stevedh/smap2/index.html>

² <http://twistedmatrix.com/trac/>

³ <http://zeromq.org/>

⁴ <http://www.modbus.org/>

⁵ <http://code.google.com/p/pymodbus/>

- **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 (e.g., RTUs) 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.
- **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 OMQ Curve.
- **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.
- **Logging Service:** Agents can publish integer or double data to arbitrary paths to a logging topic and this service will push them to the sMAP historian for later analysis. The primary use of the logging service is to allow agents to record actions or results from the agent executing its services.
- 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.

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).

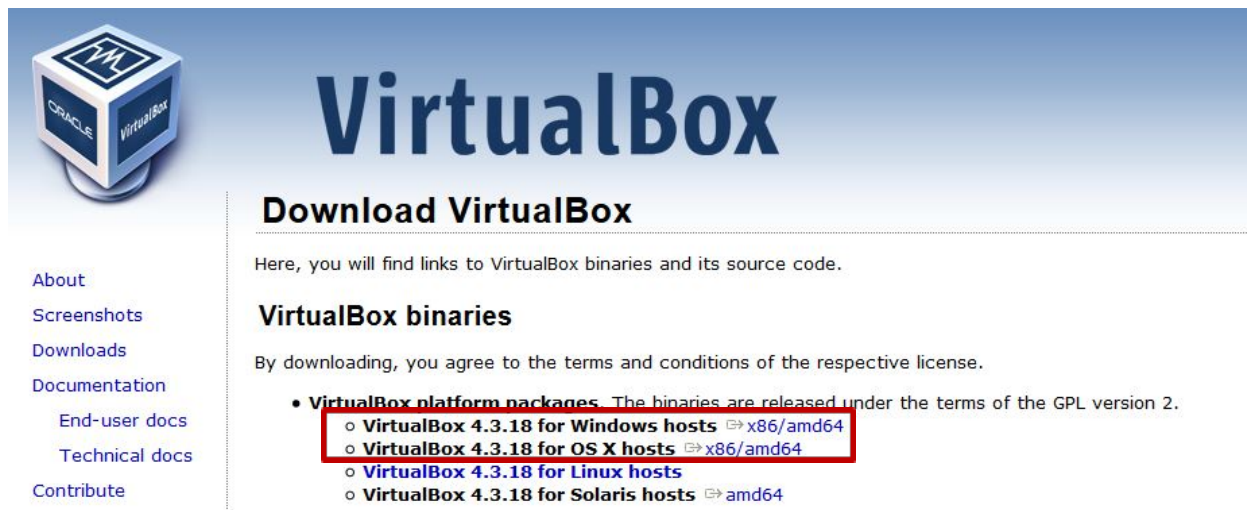


Figure 2: VirtualBox download page

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 4.3.18. 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 Debian based distribution of Linux should work with VOLTTRON (Debian, Ubuntu, Linux Mint, etc.) but this document will describe the development of agents within VOLTTRON where Linux Mint 17 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/download.php>. Set up of the platform in Ubuntu is identical to the setup in Linux Mint⁶ except for changes in the appearance of the desktop. If running VOLTTRON on a system with limited hardware (less than 2 GB of RAM), a 32-bit version of Linux should be used.

Download links				
			EDITION	MULTIMEDIA SUPPORT *
Cinnamon	<u>32-bit</u>	<u>64-bit</u>	An edition featuring the <u>Cinnamon</u> desktop	Yes
Cinnamon No codecs	<u>32-bit</u>	<u>64-bit</u>	A version without multimedia support. For magazines, companies and distributors in the USA, Japan and countries where the legislation allows patents to apply to software and distribution of restricted technologies may require the acquisition of 3rd party licenses*.	No
Cinnamon OEM	<u>64-bit</u>		An installation image for manufacturers to pre-install Linux Mint.	No
MATE	<u>32-bit</u>	<u>64-bit</u>	An edition featuring the <u>MATE</u> desktop	Yes
MATE No codecs	<u>32-bit</u>	<u>64-bit</u>	A version without multimedia support. For magazines, companies and distributors in the USA, Japan and countries where the legislation allows patents to apply to software and distribution of restricted technologies may require the acquisition of 3rd party licenses*.	No
MATE OEM	<u>64-bit</u>		An installation image for manufacturers to pre-install Linux Mint.	No
KDE	<u>32-bit</u>	<u>64-bit</u>	An edition featuring the <u>KDE</u> desktop	Yes
Xfce	<u>32-bit</u>	<u>64-bit</u>	An edition featuring the <u>Xfce</u> desktop	Yes

* Missing codecs and extra applications can be installed with a simple click of the mouse.

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.

⁶ 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.

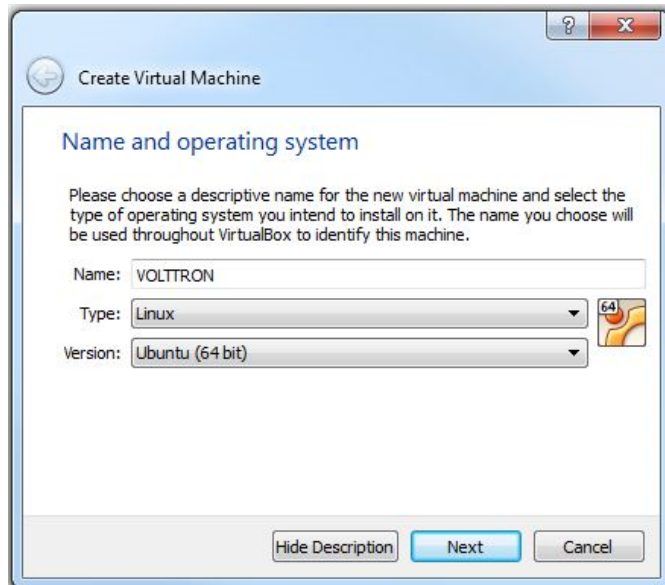


Figure 4: Creating a Virtual Machine

3. Choose the amount of memory to allot 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.

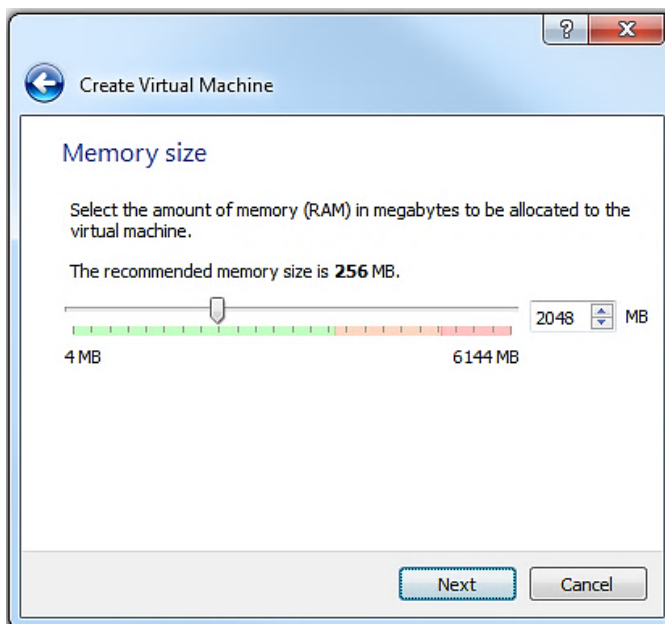


Figure 5: Selecting Memory Size

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

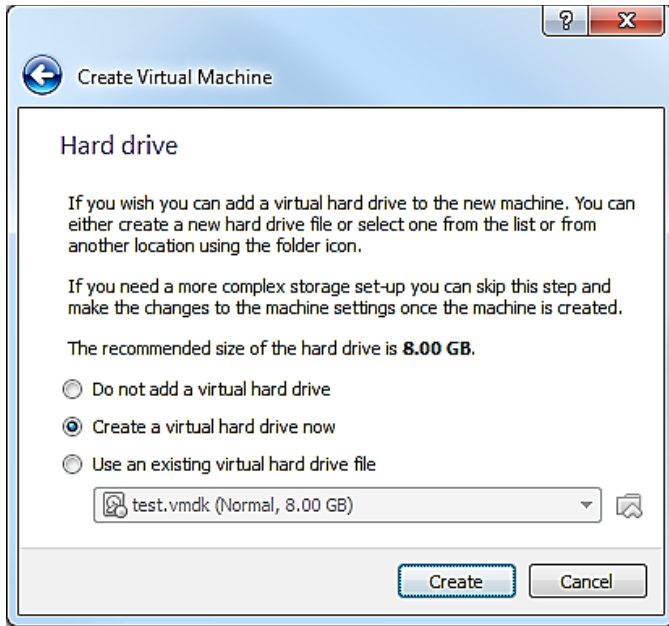


Figure 6: Selecting Storage Size

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

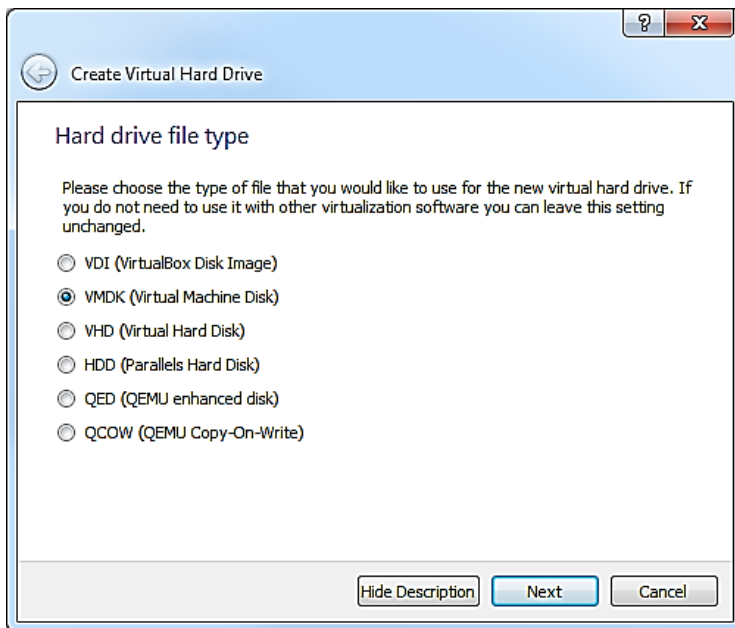


Figure 7: Creating Virtual Hard Drive

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

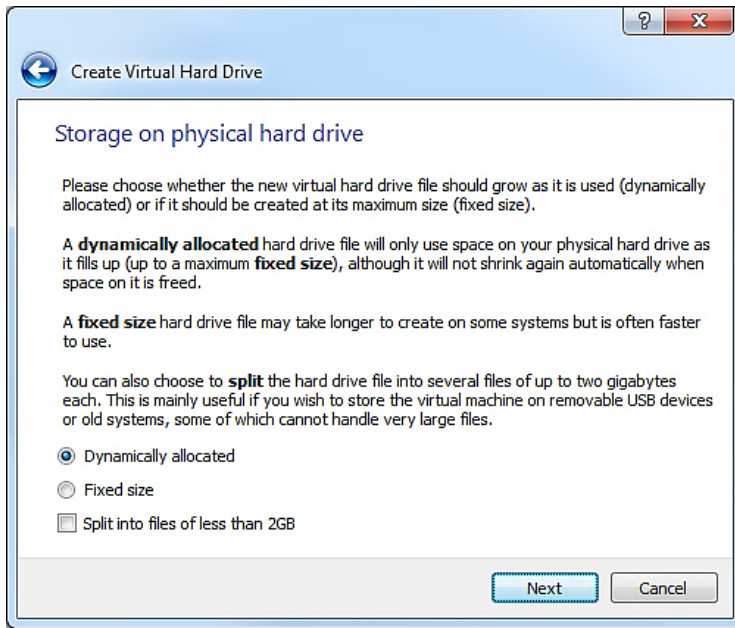


Figure 8: Selection of type of hard drive

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.

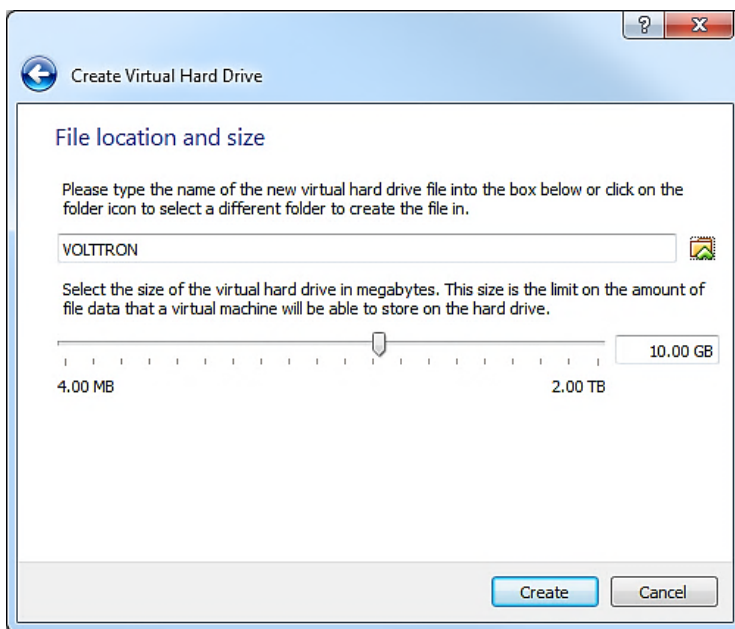


Figure 9: Creating Virtual Hard Drive (continued)

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). Choose OK.

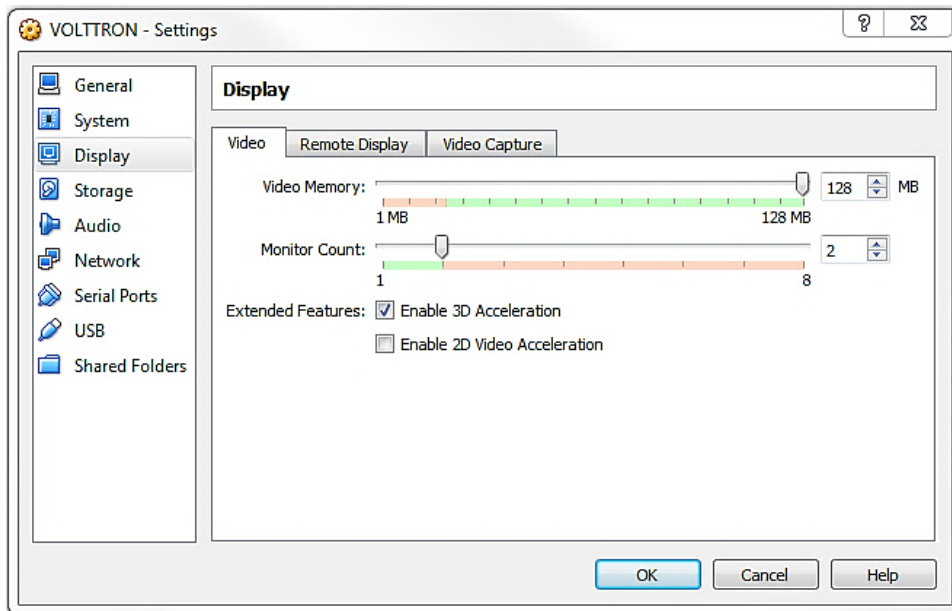


Figure 10: Selection of display type

9. 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 system menu go to the Processor tab and Enable PAE/NX (Figure 11). Choose OK.

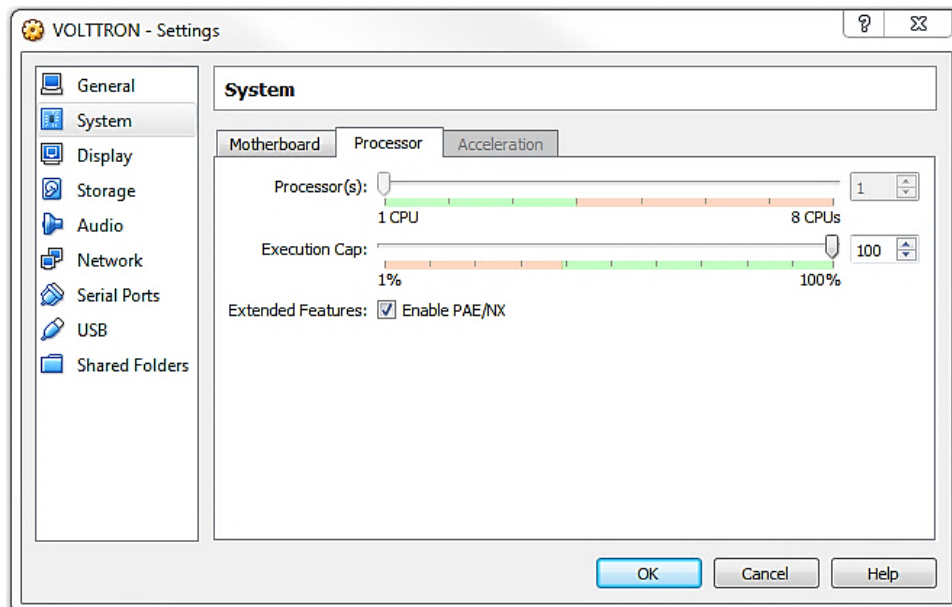


Figure 11: Selection of processor parameter

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.

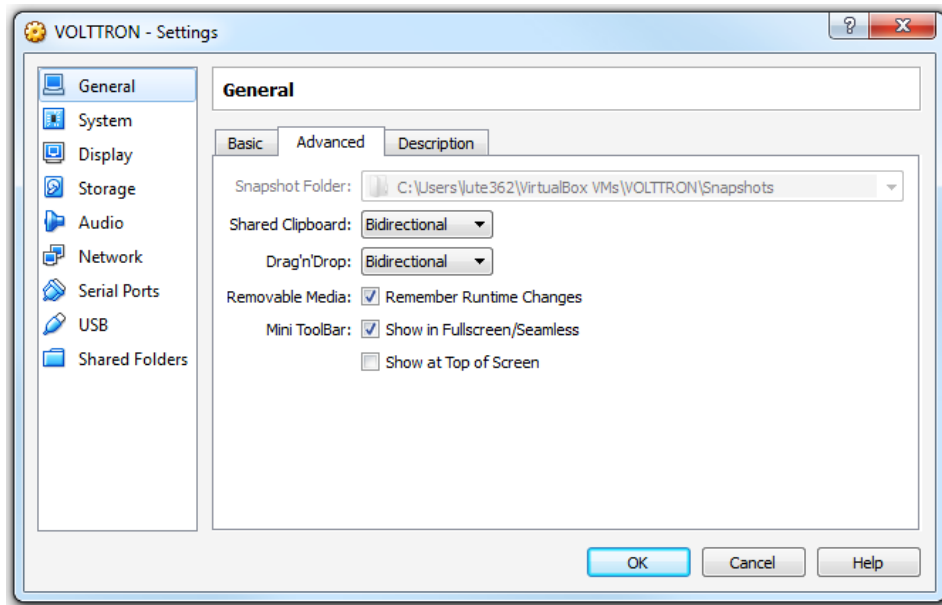


Figure 12: Enable bidirectional copy and paste (Shared Clipboard) and Drag'n'Drop

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.



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 (terminal commands are bold). Figure 15 shows the terminal command to install VOLTTRON software dependencies.



Figure 15: Linux Mint Terminal Window

- Ensures the installer is up to date:
sudo apt-get update
- This installs Git. The transactional network source code including VOLTTRON and other agent code is stored in a Git repository:
sudo apt-get install git
- This installs Python DevTools. This is a Python software development tool necessary for running VOLTTRON:
sudo apt-get install python-dev
- g++ is a C++ compatible runtime library:
sudo apt-get install g++
- build-essential is used to build and install Debian packages:
sudo apt-get install build-essential
- Required development library:
sudo apt-get install libevent-dev
sudo apt-get install libssl-dev
sudo apt-get install openssl
- Optional Python module that allows developers to utilize Python interface library:
sudo apt-get install python-tk
- One line command to grab all dependencies (copy and paste to avoid typos):

**sudo apt-get update && sudo apt-get install build-essential openssl git python-dev g++
libevent-dev libssl-dev python-tk**

2.4 Installing the sMAP Server (Optional)

VOLTTRON uses sMAP as its data repository for storing data from devices and log messages from agents. If you have access to an existing sMAP server, you can configure your VOLTTRON instance to work with that as in 2.10.1.

To install your own sMAP instance, follow the installation instructions from the following URL: <http://pythonhosted.org/Smmap/en/2.0/install.html>

We recommend skipping “Installing from Source” and installing it on the recommended OS.

2.5 Checking Out Transactional Network from Repository

Ensure you have installed the required packages before proceeding. We recommend creating a directory, ~/volttron, the instructions in this guide are written assuming this file structure. Enter the following commands (terminal commands are bold).

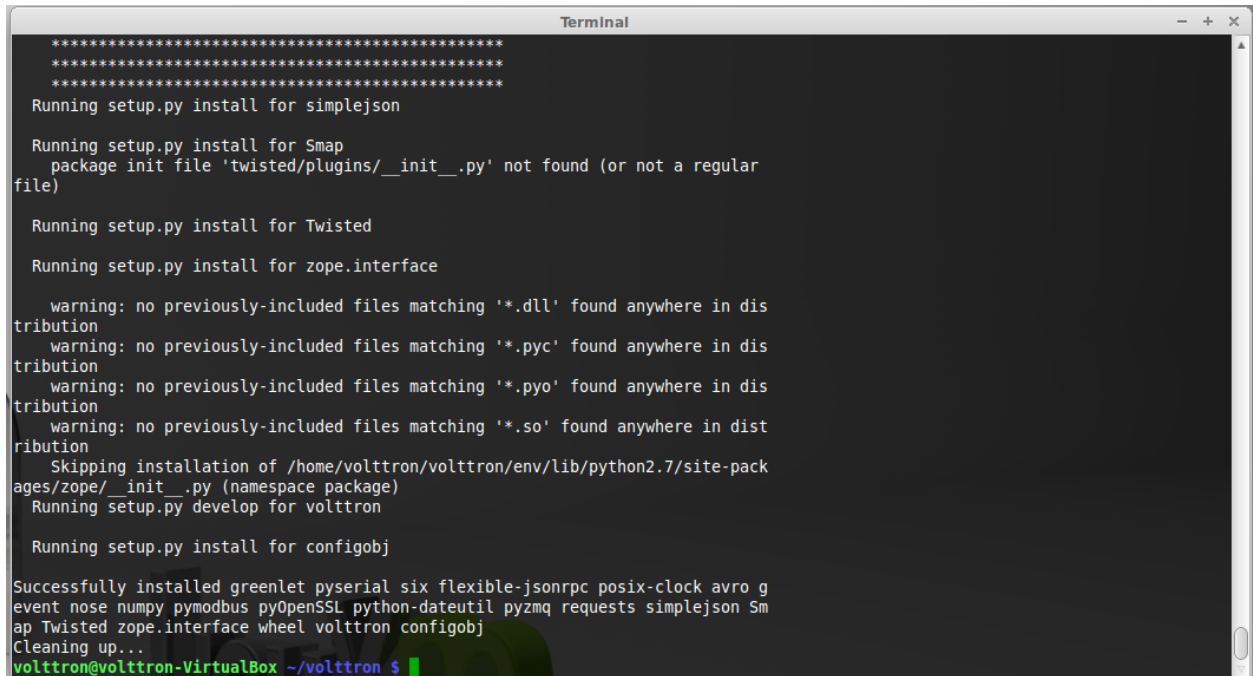
1. Creates the volttron directory, downloads the VOLTTRON source code and creates a local copy on your machine:

git clone <https://github.com/VOLTTRON/volttron> volttron

2.6 Building the VOLTTRON Platform

In the volttron directory, enter the following commands (terminal commands are bold and explanation of the commands are in normal font):

2. Go to volttron directory:
cd volttron
3. VOLTTRON includes scripts that automatically pull down dependencies and build the necessary libraries. The “bootstrap” script has to be run only once. Some of the packages (especially Numpy) can be very verbose when they install. Wait for the wall of text to end. To run the bootstrap script, from the volttron directory, 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 install for simplejson

Running setup.py install for Smap
package init file 'twisted/plugins/__init__.py' not found (or not a regular
file)

Running setup.py install for Twisted

Running setup.py install for zope.interface

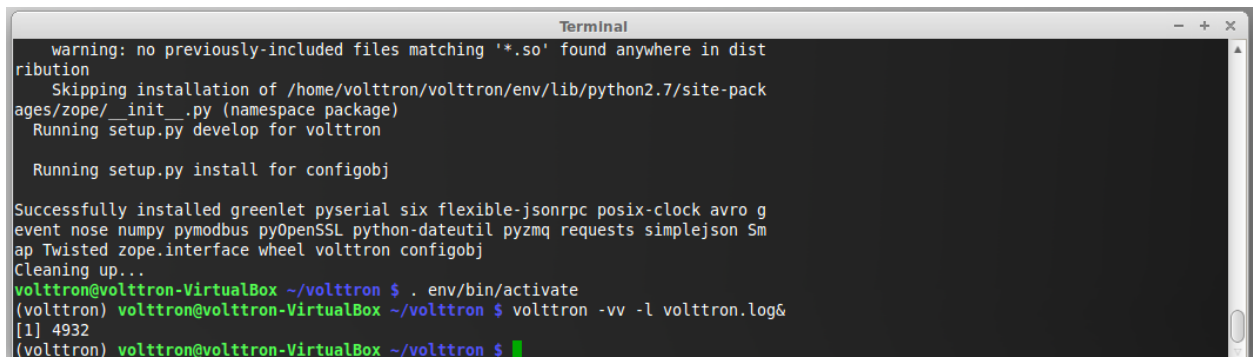
warning: no previously-included files matching '*.dll' found anywhere in dist
tribution
warning: no previously-included files matching '*.pyc' found anywhere in dist
tribution
warning: no previously-included files matching '*.pyo' found anywhere in dist
tribution
warning: no previously-included files matching '*.so' found anywhere in dist
tribution
Skipping installation of /home/volttron/volttron/env/lib/python2.7/site-pack
ages/zope/__init__.py (namespace package)
Running setup.py develop for volttron

Running setup.py install for configobj

Successfully installed greenlet pyserial six flexible-jsonrpc posix-clock avro g
event nose numpy pymodbus pyOpenSSL python-dateutil pyzmq requests simplejson Sm
ap Twisted zope.interface wheel volttron configobj
Cleaning up...
volttron@volttron-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 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:



```
warning: no previously-included files matching '*.so' found anywhere in dist
tribution
Skipping installation of /home/volttron/volttron/env/lib/python2.7/site-pack
ages/zope/__init__.py (namespace package)
Running setup.py develop for volttron

Running setup.py install for configobj

Successfully installed greenlet pyserial six flexible-jsonrpc posix-clock avro g
event nose numpy pymodbus pyOpenSSL python-dateutil pyzmq requests simplejson Sm
ap Twisted zope.interface wheel volttron configobj
Cleaning up...
volttron@volttron-VirtualBox ~/volttron $ . env/bin/activate
(volttron) volttron@volttron-VirtualBox ~/volttron $ volttron -vv -l volttron.log&
[1] 4932
(volttron) volttron@volttron-VirtualBox ~/volttron $
```

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

At this point, all required software has been installed and basic configuration has been completed. Next, the installation has to be tested.

2.7 VOLTTRON Home Directory and Configuration

By default, the VOLTTRON projects bases its files out of VOLTTRON_HOME, which defaults to "~/volttron".

- \$VOLTTRON_HOME/ - agents contains the agents installed on the platform
- \$VOLTTRON_HOME/ - certificates contains the certificates for use with the Licensed VOLTTRON code.
- \$VOLTTRON_HOME/ - run contains files create by the platform during execution. The main ones are the 0MQ files created for publish and subscribe.
- \$VOLTTRON_HOME/ - ssh keys used by agent mobility in the licensed VOLTTRON code
- \$VOLTTRON_HOME/config - Default location to place a config file to override any platform settings.
- /tmp/volttron_wheels - is where agent packages created with `volttron-pkg` package are created

2.8 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 4.3 for information on installing and running agents in the Eclipse IDE. The Listener agent is a VOLTTRON platform 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.

From the volttron directory, enter the following commands in a terminal window:

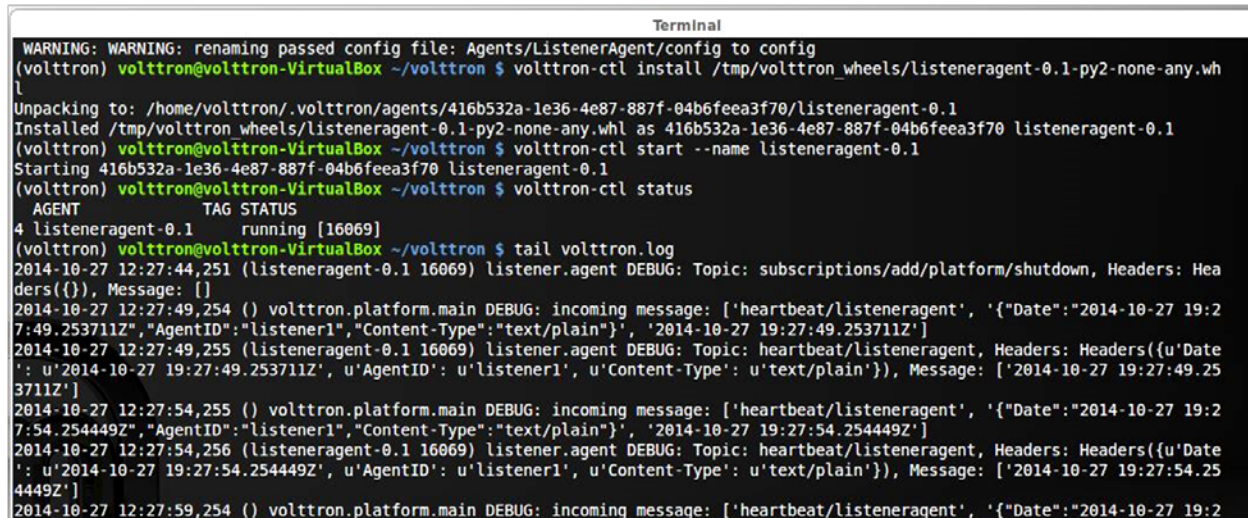
1. Package the agent:
volttron-pkg package Agents/ListenerAgent
2. Set the configuration file:
volttron-pkg configure /tmp/volttron_wheels/listeneragent-0.1-py2-none-any.whl Agents/ListenerAgent/config
3. Install agent into platform (with the platform running):
volttron-ctl install /tmp/volttron_wheels/listeneragent-0.1-py2-none-any.whl
 - Upon successful completion of this command, the terminal output will inform one on 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 416b532a-1e36-4e87-887f-04b6feca3f70 listeneragent-0.1**
4. Start the agent:
volttron-ctl start --name listeneragent-0.1
 - Agent commands can also use the uuid as an identifier (i.e., **volttron-ctl start --uuid 416b532a-1e36-4e87-887f-04b6feca3f70**). This is helpful when managing multiple instances of the same agent.
5. Verify that agent is running:
volttron-ctl status
tail 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-0.1
```

```
volttron-ctl remove --name listeneragent-0.1
```

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



```
Terminal
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.whl
Unpacking to: /home/volttron/.volttron/agents/416b532a-1e36-4e87-887f-04b6f6ea3f70/listeneragent-0.1
Installed /tmp/volttron_wheels/listeneragent-0.1-py2-none-any.whl as 416b532a-1e36-4e87-887f-04b6f6ea3f70 listeneragent-0.1
(volttron) volttron@volttron-VirtualBox ~/volttron $ volttron-ctl start --name listeneragent-0.1
Starting 416b532a-1e36-4e87-887f-04b6f6ea3f70 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: Headers({}), Message: []
2014-10-27 12:27:49,254 () volttron.platform.main DEBUG: incoming message: ['heartbeat/listeneragent', '{"Date": "2014-10-27 19:27:49.253711Z", "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': '2014-10-27 19:27:49.253711Z', 'u'AgentID': 'u'listener1', 'u'Content-Type': 'u'text/plain'}), Message: ['2014-10-27 19:27:49.253711Z']
2014-10-27 12:27:54,255 () volttron.platform.main DEBUG: incoming message: ['heartbeat/listeneragent', '{"Date": "2014-10-27 19:27:54.254449Z", "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': '2014-10-27 19:27:54.254449Z', 'u'AgentID': 'u'listener1', 'u'Content-Type': 'u'text/plain'}), Message: ['2014-10-27 19:27:54.254449Z']
2014-10-27 12:27:59,254 () volttron.platform.main DEBUG: incoming message: ['heartbeat/listeneragent', '{"Date": "2014-10-27 19:27:59.254449Z", "AgentID": "listener1", "Content-Type": "text/plain"}', '2014-10-27 19:27:59.254449Z']
```

Figure 18: Sample Output from the Listener Agent

2.9 Launching the Weather Agent

The Weather agent, another VOLTTRON service agent, retrieves weather information from the WeatherUnderground 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 WeatherUnderground.

2.9.1 Obtaining a Developer Key from WeatherUnderground

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

- Go to WeatherUnderground site (Figure 19) the following URL <http://www.wunderground.com/weather/api/>
- Select, Sign Up for FREE

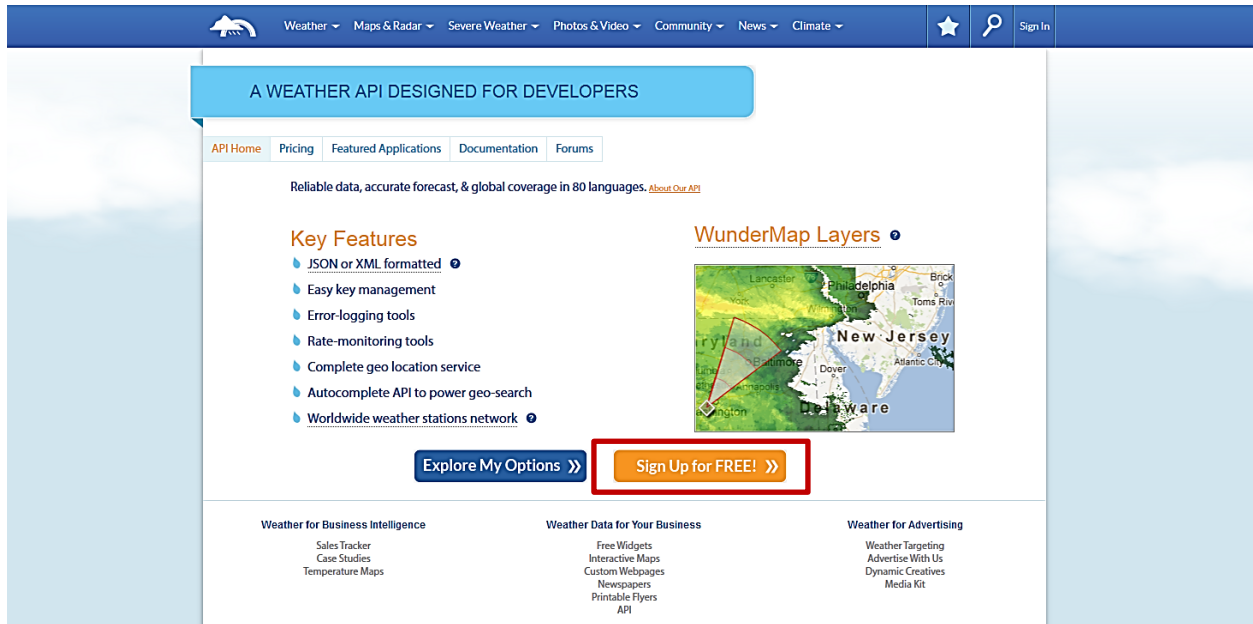


Figure 19: WeatherUnderground Website

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

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:

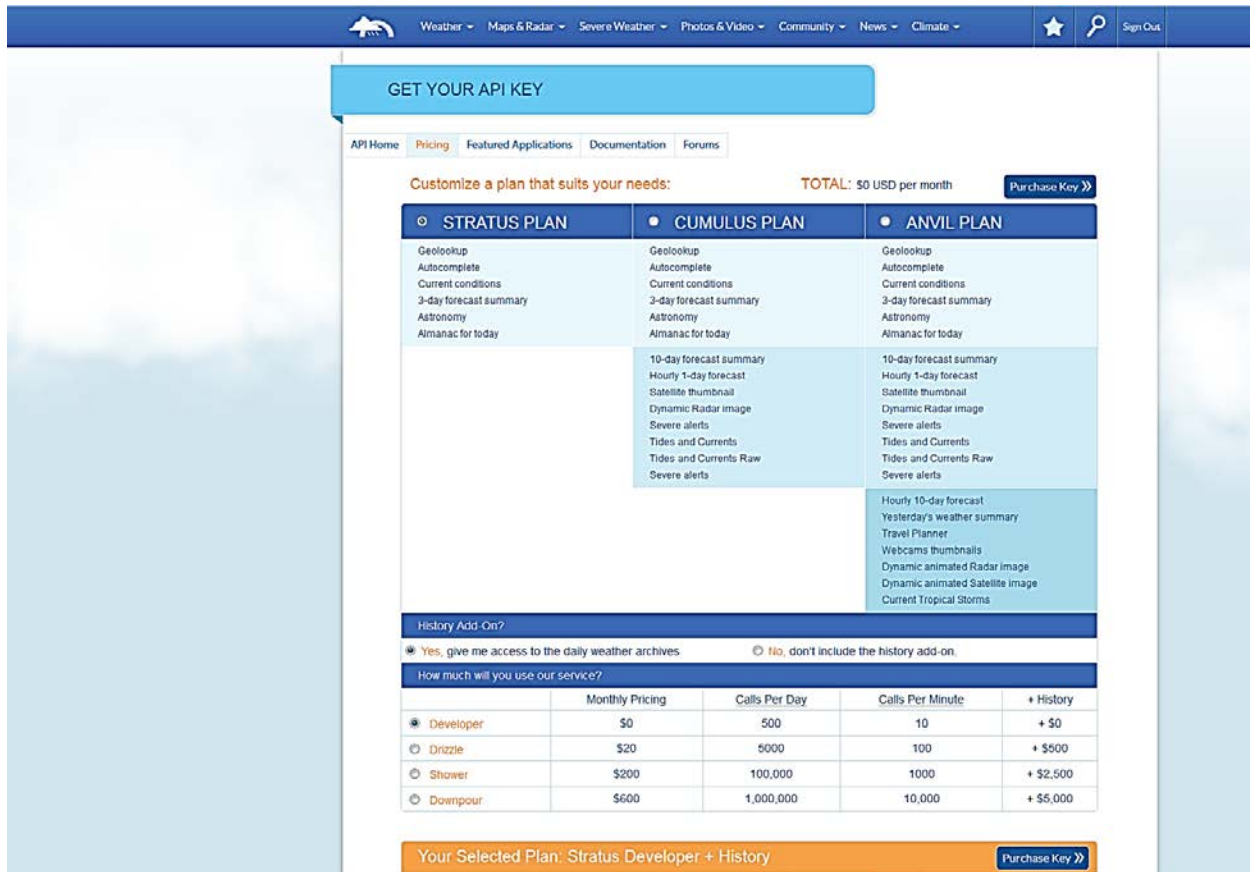


Figure 21: Creating a WeatherUnderground API Key

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

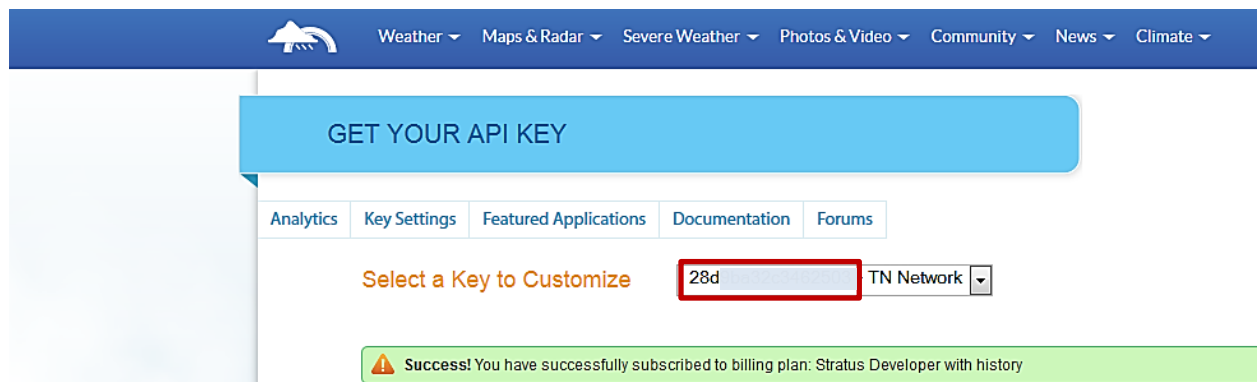


Figure 22: WeatherUnderground API Key

2.9.2 Configuring WeatherAgent with Developer Key and Location

The following steps will show how to configure the Weather agent with the developer key from WeatherUnderground and how to enter a zip code to get weather data from that zip code.

Edit `Agents/WeatherAgent/weather/settings.py` with your WeatherUnderground key. From the `volttron` directory, enter the following terminal commands:

- Go to WeatherAgent directory:

cd Agents/WeatherAgent/weather

2. Open settings.py with a text editor or nano:
nano settings.py
3. Enter the key, as shown in Figure 23:

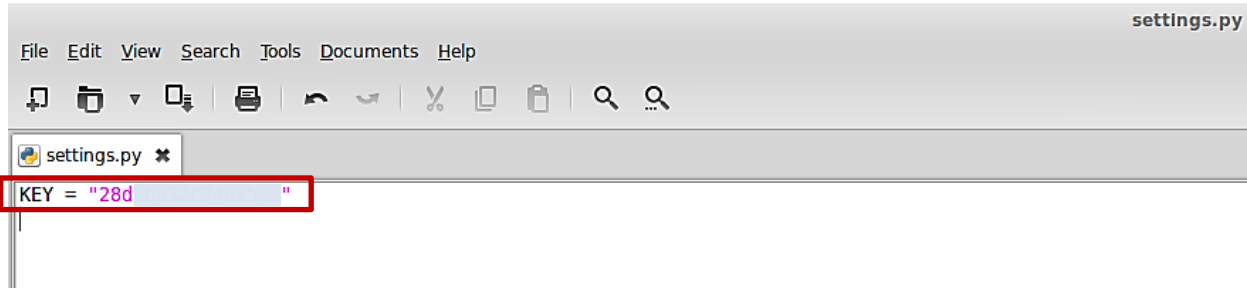


Figure 23: Entering the WeatherUnderground Developer Key

4. Open the Weather agent's configuration file, Agents/WeatherAgent/config, and edit "zip" field, as shown in Figure 24:
nano config

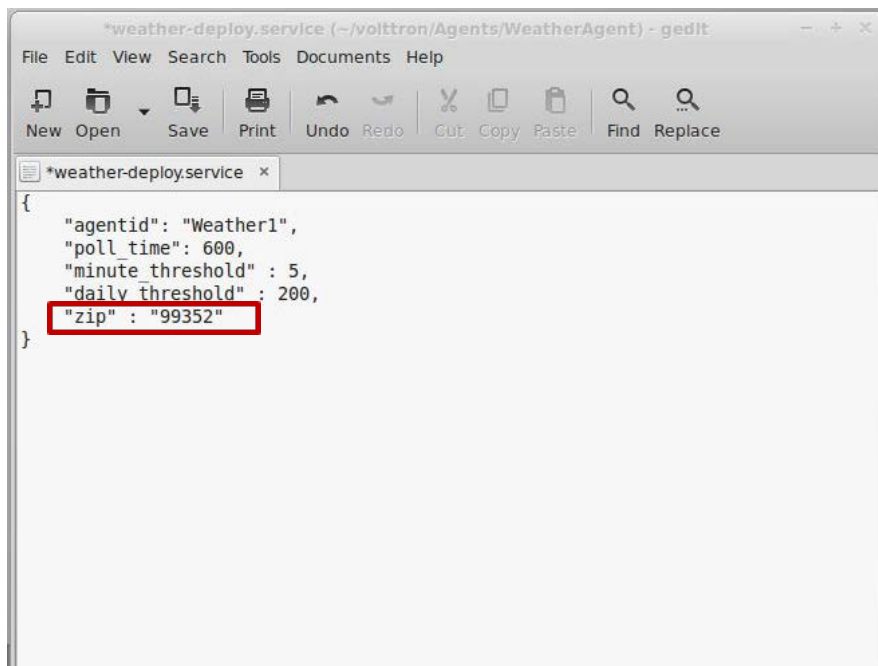


Figure 24: Entering Zip Code for the Location

2.9.3 Launching the Weather Agent

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

1. Package the agent:
volttron-pkg package Agents/WeatherAgent
2. Set the configuration file:

**volttron-pkg configure /tmp/volttron_wheels/weatheragent-0.1-py2-none-any.whl
Agents/WeatherAgent/weather-deploy.service**

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

volttron-ctl install /tmp/volttron_wheels/weatheragent-0.1-py2-none-any.whl

- Upon successful completion of this command, the terminal output will inform one of the install directories, 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
a84c91a7-9b0d-491f-aa12-c6abc676a55b weatheragent-0.1**

4. Start the agent:

volttron-ctl start --name weatheragent-0.1

- Agent commands can also use the uuid as an identifier (i.e., **volttron-ctl start --uuid a84c91a7-9b0d-491f-aa12-c6abc676a55b**). This is helpful when managing multiple instances of the same agent.

5. Verify that agent is running:

**volttron-ctl status
tail 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 --name weatheragent-0.1

volttron-ctl remove --name weatheragent-0.1

Figure 25 shows example output from the Weather Agent.

```
File: volttron.log
Terminal
DEBUG: incoming message: ['weather/temperature/heat_index_c', '{"From": "Weather1", "Content-Type": "text/plain"}', 'NA']
DEBUG: incoming message: ['weather/temperature/windchill_c', '{"From": "Weather1", "Content-Type": "text/plain"}', 'NA']
DEBUG: incoming message: ['weather/temperature/feelslike_string', '{"From": "Weather1", "Content-Type": "text/plain"}', '57.0 F (13.9 C)']
DEBUG: incoming message: ['weather/temperature/temperature_string', '{"From": "Weather1", "Content-Type": "text/plain"}', '57.0 F (13.9 C)']
DEBUG: incoming message: ['weather/cloud_cover/all', '{"Content-Type": ["application/json"], "From": "Weather1"}', '{"visibility_mi\\": \\']
DEBUG: incoming message: ['weather/cloud_cover/visibility_mi', '{"From": "Weather1", "Content-Type": "text/plain"}', '10.0']
DEBUG: incoming message: ['weather/cloud_cover/solarradiation', '{"From": "Weather1", "Content-Type": "text/plain"}', '--']
DEBUG: incoming message: ['weather/cloud_cover/weather', '{"From": "Weather1", "Content-Type": "text/plain"}', 'Mostly Cloudy']
DEBUG: incoming message: ['weather/cloud_cover/visibility_km', '{"From": "Weather1", "Content-Type": "text/plain"}', '16.1']
DEBUG: incoming message: ['weather/cloud_cover/UV', '{"From": "Weather1", "Content-Type": "text/plain"}', '3']
DEBUG: incoming message: ['weather/location/all', '{"Content-Type": ["application/json"], "From": "Weather1"}', '{"display_location\\": {\\']
DEBUG: incoming message: ['weather/location/display_location/all', '{"Content-Type": ["application/json"], "From": "Weather1"}', '{"city\\']
DEBUG: incoming message: ['weather/location/display_location/city', '{"From": "Weather1", "Content-Type": "text/plain"}', 'Richland']
DEBUG: incoming message: ['weather/location/display_location/full', '{"From": "Weather1", "Content-Type": "text/plain"}', 'Richland, WA']
DEBUG: incoming message: ['weather/location/display_location/magic', '{"From": "Weather1", "Content-Type": "text/plain"}', '1']
```

Figure 25: Example Output from the Weather Agent

2.10 Configuring and Launching sMAP Driver

The sMAP driver allows a user to store time series data in the sMAP historian and to communicate with Modbus and BACnet compliant devices. Configuring the driver consists of creating a Modbus and/or a BACnet registry files and creating a sMAP driver configuration file. The Modbus and BACnet registry files tell the driver what the Modbus address is for each register, what data type it can hold, and if the register is writeable or read-only. The sMAP driver configuration file tells the sMAP driver where to find the Modbus and BACnet registry files, the location (url) of the sMAP historian to write data to, and information related to the device you are monitoring or controlling.

2.10.1 Configuring sMAP driver

The basic Modbus commands can instruct a device to change a value in one of its registers, control or read an I/O port, as well as command the device to send back one or more values contained in its registers. 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 or BACnet register), and the point/register address on the device (point address). An example of a Modbus registry file is shown in Figure 26, and an example BACnet registry file is shown in Figure 27.

1. Create the Modbus registry file and/or the BACnet registry file.

PNNL Point Name	Units	Units Details	Modbus Register	Writable	Point Address	Notes
ReturnAirCO2	PPM	0.00-2000.00	>f	FALSE	1001	CO2 Reading 0.00-2000.0 ppm
SupplyFanSpeed	%	0.00 to 100.00	>f	FALSE	1003	Fan speed from drive
CoolSupplyFanSpeed1	%	0.00 to 100.00 (75 default)	>f	TRUE	1005	Fan speed on cool 1 call
CoolSupplyFanSpeed2	%	0.00 to 100.00 (90 default)	>f	TRUE	1007	Fan speed on Cool2 Call
DischargeAirTemperature	F	(-)39.99 to 248.00	>f	FALSE	1009	Discharge air reading
ReturnAirCO2Stpt	PPM	1000.00 (default)	>f	TRUE	1011	Setpoint to enable demand control ventilation
DamperSignal	%	0.00 - 100.00	>f	FALSE	1023	Output to the economizer damper
MixedAirTemperature	F	(-)39.99 to 248.00	>f	FALSE	1025	Mixed Air Temperature from Probe
OutsideAirTemperature	F	(-)39.99 to 248.00	>f	FALSE	1029	Outside Air Temperature
OutdoorAirVolume	%	0.00 to 100.00	>f	FALSE	1031	Outside air volume calculated by multiplying damper and fan speed
ReturnAirTemperature	F	(-)39.99 to 248.00	>f	FALSE	1037	Return air temperature reading
DamperCommand	%	0-100	>f	TRUE	1059	Damper position from Volttron to CATALYST
MinimumDamperPositionStPt	%	0-100	>f	TRUE	1063	Damper Minimum Position Set to CATALYST from Volttron
HeatingTemperatureStPt	F	0-100	>f	TRUE	1065	Heating setpoint sent to the CATALYST from Volttron
CoolingTemperatureStPt	F	0-100	>f	TRUE	1067	Cooling setpoint sent to the CATALYST from Volttron

Figure 26: An Example Modbus Registry File

PNNL Point Name	Units	Unit Details	BACnet Object Type	Property	Writable	Index	Notes
DischargeAirStaticPressure	inchesOfWater	-0.20 to 5.00	analogInput	presentValue	FALSE	3000108	Resolution: 0.001
DischargeAirTemperature	degreesFahrenheit	-50.00 to 250.00	analogInput	presentValue	FALSE	3000109	Resolution: 0.1
MixedAirTemperature	degreesFahrenheit	-50.00 to 250.00	analogInput	presentValue	FALSE	3000116	Resolution: 0.1
OutdoorAirHumidity	percentRelativeHumidity	0.00 to 100.00	analogInput	presentValue	FALSE	3000117	Resolution: 0.1
PreheatTemperature	degreesFahrenheit	-50.00 to 250.00	analogInput	presentValue	FALSE	3000119	Resolution: 0.1
ReturnAirTemperature	degreesFahrenheit	-50.00 to 250.00	analogInput	presentValue	FALSE	3000120	Resolution: 0.1
ReturnAirHumidity	percentRelativeHumidity	0.00 to 100.00	analogInput	presentValue	FALSE	3000124	Resolution: 0.1
CoolingValveOutputCommand	percent	0.00 to 100.00 (default 0.0)	analogOutput	presentValue	TRUE	3000107	Resolution: 0.1
MixedAirDamperOutputCommand	percent	0.00 to 100.00 (default 0.0)	analogOutput	presentValue	TRUE	3000110	Resolution: 0.1
PreheatValveOutputCommand	percent	0.00 to 100.00 (default 0.0)	analogOutput	presentValue	TRUE	3000111	Resolution: 0.1
ReheatValveOutputCommand	percent	0.00 to 100.00 (default 0.0)	analogOutput	presentValue	TRUE	3000112	Resolution: 0.1
SupplyFanSpeedOutputCommand	percent	0.00 to 100.00 (default 0.0)	analogOutput	presentValue	TRUE	3000113	Resolution: 0.1
ReturnFanSpeedOutputCommand	percent	0.00 to 100.00 (default 0.0)	analogOutput	presentValue	TRUE	3000122	Resolution: 0.1

Figure 27: An Example BACnet Registry File

The data fields boxed in red in Figure 26 and Figure 27 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 inside the workspace (i.e., ~/volttron/drivers/modbus.csv or ~/volttron/drivers/bacnet.csv).

For more details on the Modbus registry file or BACnet registry file visit the Transactional Network Wiki:

Modbus - <https://github.com/VOLTTRON/volttron/wiki/ModbusDriver>

BACnet - <https://github.com/VOLTTRON/volttron/wiki/BacnetDriver>

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

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

2. Configure the sMAP configuration file, as shown in Figure 28. 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 28.

```
[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 28: 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 VOLTRON 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.

3. Save the file within the transactional network workspace. Subsequent examples from this document will assume the file is saved as (~/.voltron/sMAP.ini).

2.10.2 Launching the Driver

After 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 VOLTRON agent or by directly calling Twisted. Running the device driver as an agent is the same process as the building and launching the Listener agent or the Weather agent.

From the voltron directory enter the following commands in a terminal window:

4. Package the agent:
voltron-pkg package Agents/TwistdLauncher
5. Set the configuration file:
voltron-pkg configure /tmp/voltron_wheels/launcheragent-0.1-py2-none-any.whl Agents/TwistdLauncher/twistd.launcher
6. Install agent into platform (with the platform running):
voltron-ctl install /tmp/voltron_wheels/launcheragent-0.1-py2-none-any.whl
 - Upon successful completion of this command, the terminal output will inform one on 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/voltron_wheels/launcheragent-0.1-py2-none-any.whl as 4d12141c-8aa8-4e88-9772-d3792710a313 launcheragent-0.1**

7. Start the agent:

volttron-ctl start --name launcheragent-0.1

- Agent commands can also use the uuid as an identifier (i.e., **volttron-ctl start --uuid 416b532a-1e36-4e87-887f-04b6feca3f70**). This is helpful when managing multiple instances of the same agent.

8. Verify that agent is running:

volttron-ctl status

tail volttron.log

The device driver can also be launched as a separate service:

1. Permanently add AGENT_PUB_ADDR and AGENT_SUB_ADDR (environment variable describing the path to the VOLTTRON message bus where “**home/volttron/**” is the path to the project directory). From a terminal in the volttron directory, complete the following steps (step 1 and step 2 are only required the first time the driver is launched):

- Open the file containing the paths to set environment variables and edit as shown in Figure 29:

sudo nano /etc/environment

2. Append the following lines to the file with quotation marks:

AGENT_PUB_ADDR="ipc:///home/volttron/.volttron/run/publish"

AGENT_SUB_ADDR="ipc:///home/volttron/.volttron/run/subscribe"

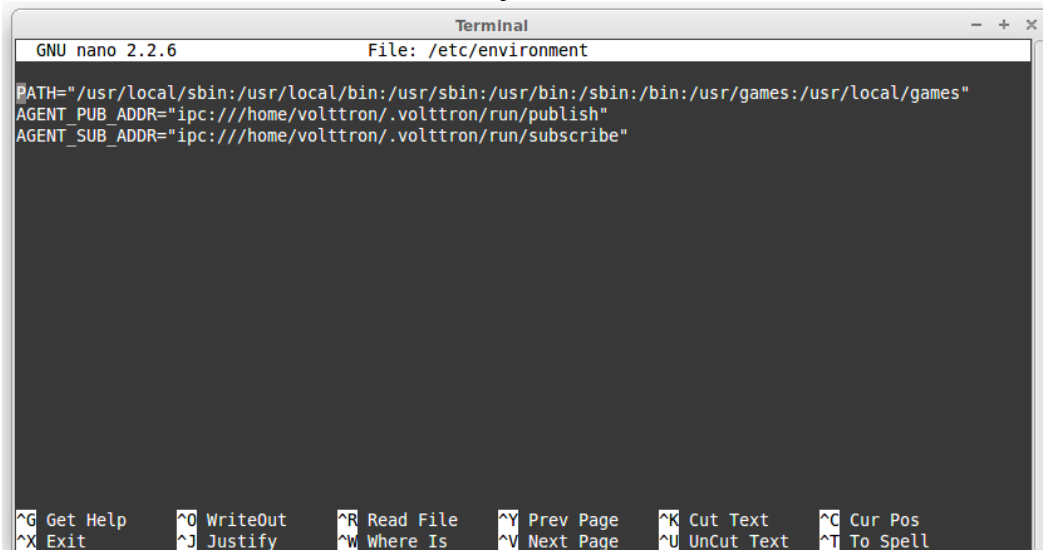


Figure 29: Setting Path to Message Bus and Environment Variables

3. Activate the project:

. env/bin/activate (note the space after the period)

4. Launch the sMAP driver:

twistd -n smap sMAP.ini

Keep this terminal window open. The driver will continue to run and allow you to interact with the device through the Actuator agent.

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 --name launcheragent-0.1  
volttron-ctl remove --name launcheragent-0.1
```

2.11 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 sMAP 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.

2.11.1 Configuring the Actuator Agent

Before launching the Actuator agent, we must create or modify the Actuator agent's configuration file (Figure 30). 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.

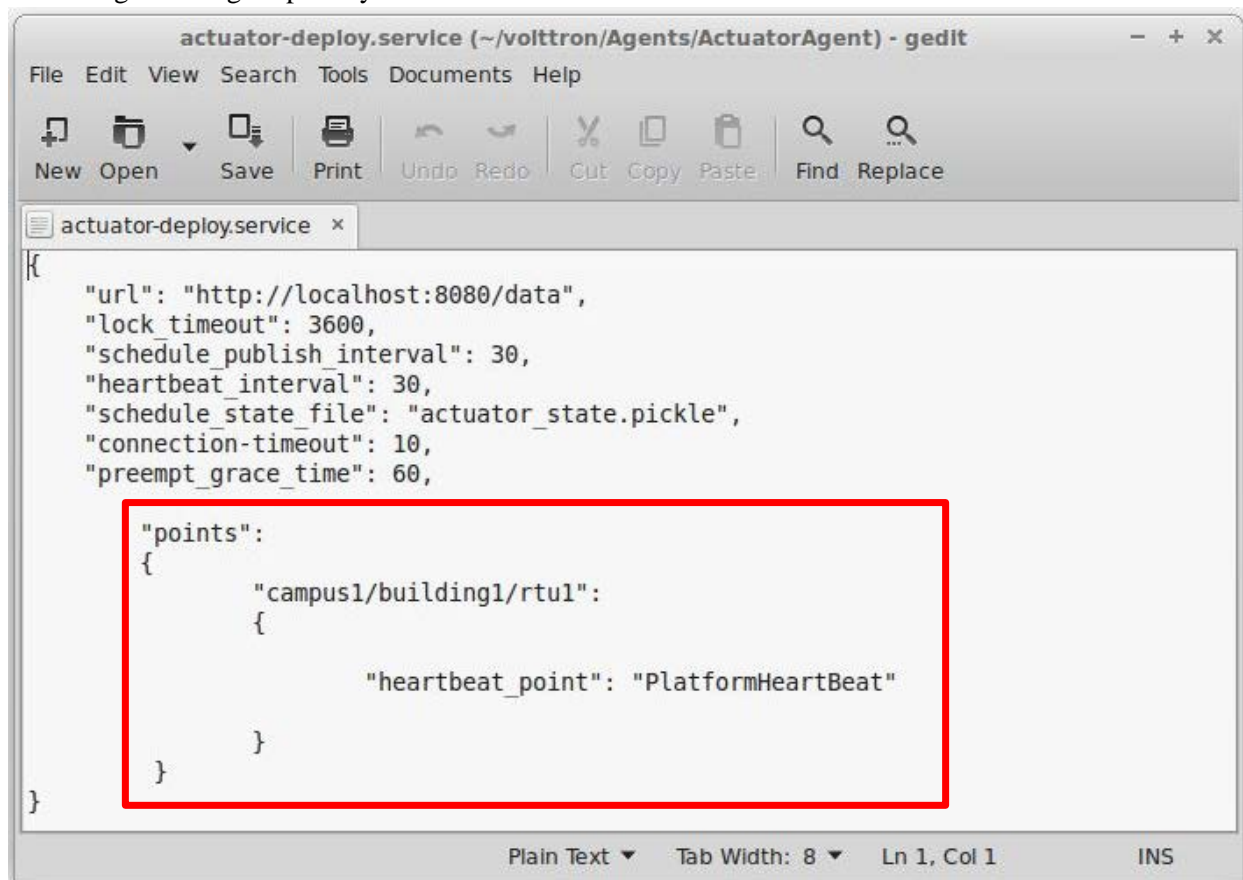


Figure 30: Example Actuator Agent Configuration File

The configuration parameters shown in Figure 30 are defined as follows:

- **schedule_publish_interval** – The interval in seconds between schedule announcements for devices being managed by the Actuator agent. These are the devices configured within the sMAP driver and the published schedule shows agent access information including which agent has scheduled access to the device.
- **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 parameters shown in the red box (in Figure 30) are related to a device heartbeat. This is a register on the control device that the Actuator agent toggles to indicate proper communication with the platform has been established. The fields in red should be left blank unless your device has this register setup.

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

2.11.2 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 “RTU/actuators/schedule/request” with the following header:

```
{
  'type': 'NEW_SCHEDULE',
  'requesterID': <Agent ID>, #The name of the requesting agent.
  'taskID': <unique task ID>, #unique ID for scheduled task.

  'priority': <task priority>, #The desired task priority, must be 'HIGH',
  'LOW', or 'LOW_PREEMPT'
}
```

The schedule request message should be formatted as follows (before converting to json):

```
[
  ["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, as noted below:

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.

2.11.3 Canceling a Task

A task may be canceled by publishing to the "RTU/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.

2.11.4 Actuator Error Reply

If something goes wrong, the Actuator agent will reply to both **get** and **set** on the **error** topic for an actuator:

```
'RTU/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>
}
```

2.11.5 Task Preemption and Schedule Failure

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

"RTU/actuators/schedule/response" with the following header:

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

And , the following message (after parsing the json):

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

2.11.5.1 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>
}
```

2.11.5.2 Failure Reasons

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

2.11.5.3 Failure Responses from Actuator Agent

The following list contains possible errors 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 taskID.

MISSING_AGENT_ID - AgentID not supplied.

TASK_ID_ALREADY_EXISTS - The supplied taskID already belongs to an existing task.

MISSING_PRIORITY - Failed to supply a priority for a task schedule request.

INVALID_PRIORITY - Priority not one of "HIGH", "LOW", or "LOW_PREEMPT".

MALFORMED_REQUEST_EMPTY - Request list is missing or empty.

REQUEST_CONFLICTS_WITH_SELF - Requested time slots on the same device overlap.

MALFORMED_REQUEST - Reported when the request parser raises an unhandled exception. The exception name and info are appended to this information string.

CONFLICTS_WITH_EXISTING_SCHEDULES - Schedule conflicts with an existing schedule that it cannot preempt. The data item for the results will contain info about the conflicts in this form (after parsing JSON):

```
{
  '<agentID1>':
  {
    '<taskID1>':
    [
      ["campus/building/device1",
       "2013-12-06 16:00:00",
       "2013-12-06 16:20:00"],
      ["campus/building/device1",
       "2013-12-06 18:00:00",
       "2013-12-06 18:20:00"]
    ]
    '<taskID2>': [ ... ]
  }
  '<agentID2>': { ... }
}
```

TASK_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

2.11.6 Actuator Agent Interaction

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.

2.11.6.1 Getting Values

While the sMAP driver 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:

```
'RTU/actuators/get/<full device path>/<actuation point>'
```

With this header:

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

2.11.6.2 Setting Values

Values are set in a similar manner. To set a value, publish a message to the following topic:

```
'RTU/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.

2.11.6.3 *Actuator Reply*

The Actuator agent will reply to both **get** and **set** on the value topic for an actuator point:

```
'RTU/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 `zmq.utils import jsonapi`).

2.11.6.4 *Common Error Types*

The following list contains possible error messages an agent may receive from the Actuator agent. This field corresponds to the **info** 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 sMAP.

2.11.7 *Device Schedule State Announcements*

Periodically the Actuator agent will publish the state of all currently used devices. For each device, the Actuator agent will publish to an associated topic:

```
'RTU/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.

2.11.8 *Launching the Actuator Agent*

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

1. Package the agent:
volttron-pkg package Agents/ActuatorAgent
2. Set the configuration file:

**volttron-pkg configure /tmp/volttron_wheels/actuatoragent-0.1-py2-none-any.whl
Agents/ActuatorAgent/actuator-deploy.service**

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

volttron-ctl install /tmp/volttron_wheels/actuatoragent-0.1-py2-none-any.whl

- Upon successful completion of this command, the terminal output will inform one of 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/actuatoragent-0.1-py2-none-any.whl as 9b45b2ad-d51d-402f-89f1-d4b21613de9d actuatoragent-0.1**

4. Start the agent:

volttron-ctl start --name actuatoragent-0.1

- Agent commands can also use the uuid as an identifier (i.e., **volttron-ctl start --uuid 9b45b2ad-d51d-402f-89f1-d4b21613de9d**). This is helpful when managing multiple instances of the same agent.

5. Verify that agent is running:

**volttron-ctl status
tail 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 --name actuatoragent-0.1
volttron-ctl remove --name actuatoragent-0.1**

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.

2.11.9 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 “messaging” with a devices’ state. The device state announcements can be used as a reminder as to when to start watching.

2.12 Multi-Building (Multi-Node) Communication

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.

2.12.1 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 31):

- 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):

```
sudo ifconfig eth0 down
sudo ifconfig 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 32 boxed in red):

```
ifconfig
```

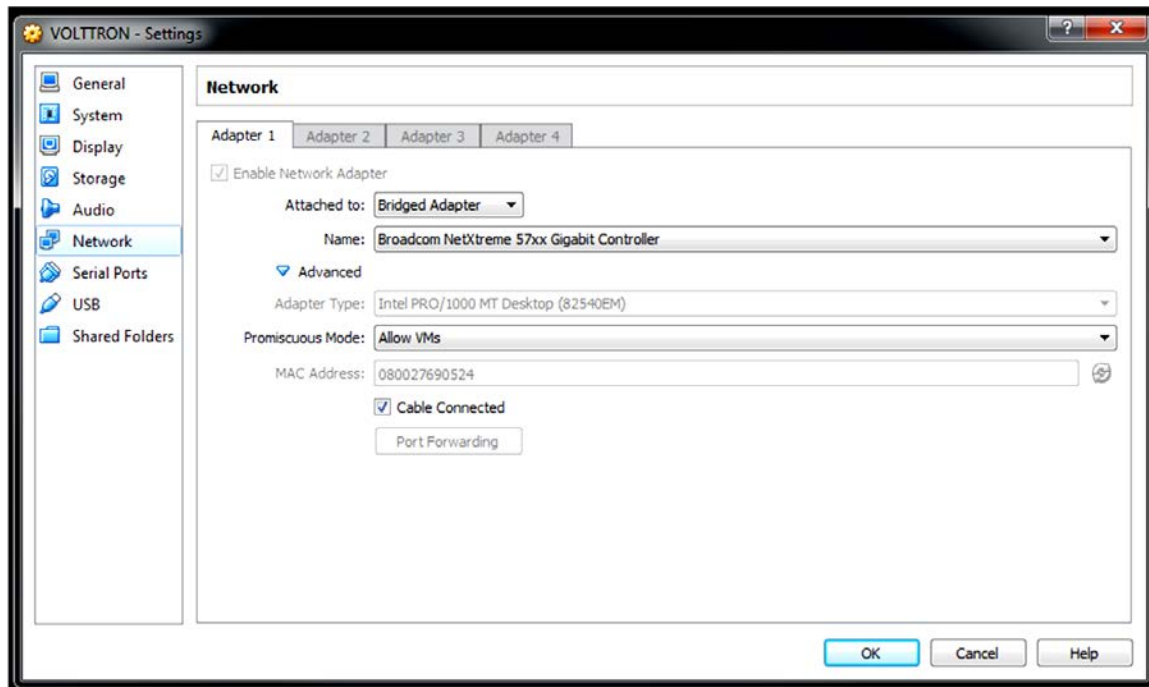


Figure 31: Configuration of VirtualBox VM for Multi-Node Communication

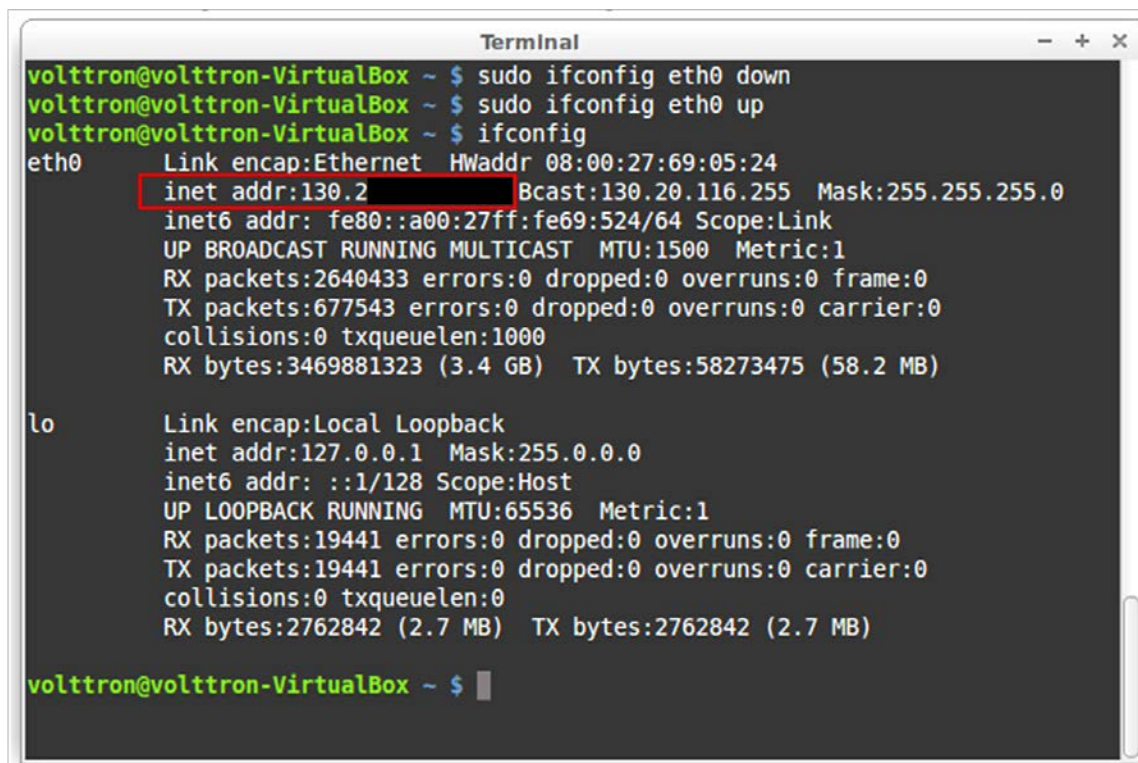


Figure 32: Identifying IP Address on VirtualBox

Figure 33 and Figure 34 shows the configuration file for two instances of the MultiBuilding Agent, each running on a separate instance of VOLTTRON on different campus and different buildings (the agents could run on the same campus and building and only run on separate instances of VOLTTRON). The top

red boxes in Figure 33 and Figure 34 give information about the agents native message bus, IP, and preferred port. The bottom red box in Figure 33 and Figure 34 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. The oval boxed in selections in Figure 33 and Figure 34 show the complimentary nature of the configuration files. 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 33: 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 34: Example of MultiBuilding Agent Configuration File for “campus1/platform2”

2.12.2 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 platform1 data to platform2"
                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

```


3 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.⁷

3.1 Automated Fault Detection and Diagnostic Agent

The automated fault detection and diagnostic (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 AFDD requires the following data fields to perform the fault detection and diagnostics: outside-air temperature, return-air temperature, mixed-air temperature, outside-air damper position/signal, supply fan status, mechanical cooling status, heating status. The AFDD supports both real-time data via a Modbus or BACnet device, or input of data from a `csv` style text document.

The following section will detail how to configure the 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 AFDD agent.

3.1.1 Configuring the AFDD Agent

Before launching the 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 35 shows an example configuration file for the AFDD agent.

The parameters boxed in black (in Figure 35) are the pre-configured fault detection thresholds; these do not require any modification to run the 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:

⁷ 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 35: Example 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 2.10 Configuring sMAP driver.

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 indicate 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)

3.1.2 Launching the AFDD Agent

The 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. Package the agent:
volttron-pkg package Agents/PassiveAFDD
2. Set the configuration file:
volttron-pkg configure /tmp/volttron_wheels/passiveafdd-0.1-py2-none-any.whl Agents/PassiveAFDD/passiveafdd.launch.json
3. Install agent into platform (with the platform running):
volttron-ctl install /tmp/volttron_wheels/passiveafdd-0.1-py2-none-any.whl
 - Upon successful completion of this command, the terminal output will inform one of 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/passiveafdd-0.1-py2-none-any.whl as 5df00517-6a4e-4283-8c70-5f0759713c64 passiveafdd-0.1**
4. Start the agent:
volttron-ctl start --name passiveafdd-0.1
 - Agent commands can also use the UUID as an identifier (i.e., **volttron-ctl start --uuid 5df00517-6a4e-4283-8c70-5f0759713c64**). This is helpful when managing multiple instances of the same agent.
5. Verify that agent is running:
volttron-ctl status
tail 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 --name passiveafdd-0.1  
volttron-ctl remove --name passiveafdd-0.1
```

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 "PassiveAFDD/passiveafdd/" directory. Below is a key that describes how to interpret the diagnostic results:

Diagnostic	Code Message
code	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 it 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

3.1.2.1 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 3.1.2.
- 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 36 shows the dialog box that is used to input the csv data file.

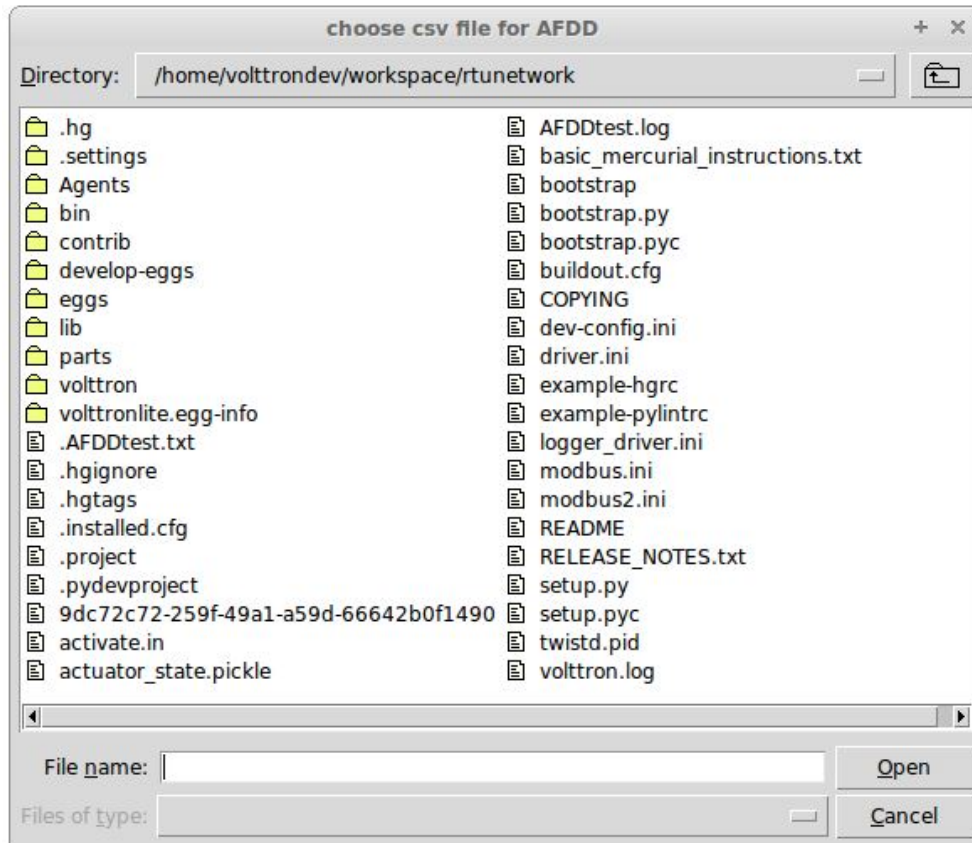


Figure 36: File Selection Dialog Box when Inputting Data in a csv File

If "Cancel" is pushed on the file input dialog box, the AFDD will acknowledge that no file was selected. The 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 37 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 37: Sample of CSV Data for 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 35, boxed in red.

3.2 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 head, 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°F below the normal. Rather than change the set point to a value that is 3 to 5°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°F above the normal, the damper is commanded to a position that is slightly below the normal minimum (half the 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.

3.2.1 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 2.11). Figure 38 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 38: Example Configuration File for the DR Agent

The parameters boxed in black (Figure 38) 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]).

3.2.2 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 format of 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, one may simulate such an event by manually publishing a DR signal on the message bus. From the volttron directory, in a terminal window, enter the following commands:

- Activate project:
 . env/bin/activate (not the space after the period)
- Start Python interpreter:
 python (this activates the Python interpreter)

- Import PublishMixin module:
from volttron.platform.agent.base import PublishMixin
- Create PublishMixin object:
p=PublishMixin('ipc:///tmp/volttron-lite-agent-publish')
- Publish simulated OpenADR message:
p.publish_json('openadr/event',{},{ 'id': 'event_id','status': 'active', 'start_at': '06-10-14 14:00', 'end_at': '06-10-14 18:00'})

To cancel this event, enter the following command:

```
p.publish_json('openadr/event',{},{ 'id': 'event_id','status': 'cancelled', 'start_at': '06-10-14 14:00', 'end_at': '06-10-14 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.

3.2.3 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.

3.2.4 Launching the Demand Response Agent

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

1. Package the agent:
volttron-pkg package Agents/DemandResponseAgent
2. Set the configuration file:
volttron-pkg configure /tmp/volttron_wheels/DemandResponseagent-0.1-py2-none-any.whl Agents/DemandResponseAgent/demandresponseagent.launch.agent
3. Install agent into platform (with the platform running):
volttron-ctl install /tmp/volttron_wheels/DemandResponseagent-0.1-py2-none-any.whl
 - Upon successful completion of this command the terminal output will inform one of 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/passiveafdd-0.1-py2-none-any.whl as**
a7efd6f5-a0d9-4e4b-9de4-b13edd5a4ace5f0759713c64
DemandResponseagent-0.1
4. Start the agent:
volttron-ctl start --name passiveafdd-0.1
 - Agent commands can also use the UUID as an identifier (i.e., **volttron-ctl start --uuid 5df00517-6a4e-4283-8c70-5f0759713c64**). This is helpful when managing multiple instances of the same agent.

5. Verify that agent is running:
volttron-ctl status
tail 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 --name passiveafdd-0.1  
volttron-ctl remove --name passiveafdd-0.1
```

3.3 Other VOLTTRON Applications

The following section will provide a brief description of the other applications available for deployment within VOLTTRON.

3.3.1 Autonomous Control of Rooftop Units

The Autonomous Control application will facilitate sensor data aggregation from various components of the building equipment and building environment. The primary objective of this application is to develop and demonstrate an optimal controller that evaluates sensor data, performs short-term prediction and optimizes the operation of multiple RTUs. The controller framework will facilitate optimization against two sets of parameters simultaneously - grid signals (load reduction, voltage regulation, renewable integration) and building-level energy-efficiency applications (occupancy, weather forecast). Within this framework, three different control strategies are investigated. The first strategy is a rule-based control that considers only the interior temperature of the building and the setting of the thermostat. The second strategy extends the first by adding rules for considering the availability of power from a photovoltaic generator. The third strategy is a model predictive control that accounts for inside- and outside-air temperatures, the thermostat set point, and the availability of energy from a photovoltaic power generator.

This application is based on a network of rooftop unit (RTU) thermostats and a centralized controller that coordinates their operation to achieve substantive reductions in peak energy use. A prototype of this new control system was built and deployed in a large gymnasium to coordinate four RTUs. Based on real-time data collected, it is estimated that the cost savings achieved by reducing peak power consumption was sufficient to repay the cost of the prototype within 1 year. This remarkably short payoff period suggests a significant commercial potential for the proposed control technology. Figure 39 shows a high-level overview of the Autonomous Control application:

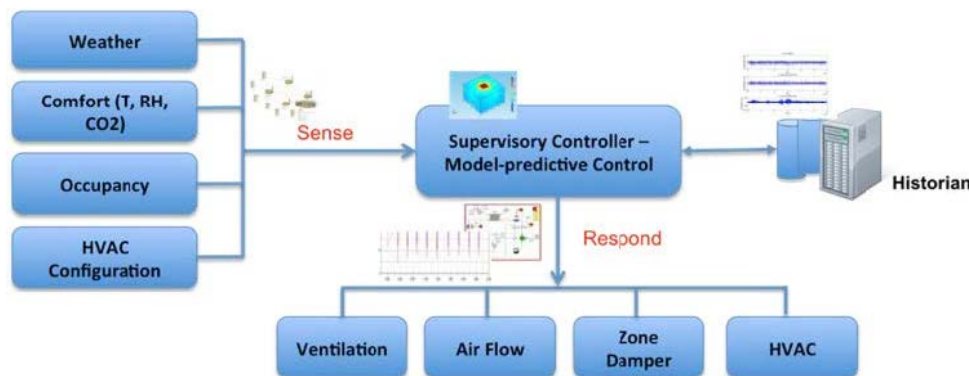


Figure 39: Autonomous Control Application

The following describes the process used by the Autonomous Control application:

- Initialize application instance and load configurable VOLTTRON parameters.
- Create Python classes that wrap the C++ code for control logic used by Modbus thermostats.
- Subscribe to sensor data (from the Weather agent).
- Execute the control logic every 10 minutes. This polls the thermostat temperature and determines which RTUs should run (based on priority and control logic).
- Thermostat sends the control signals to the RTUs.

Figure 40 shows the deployment architecture for the Autonomous Control application.

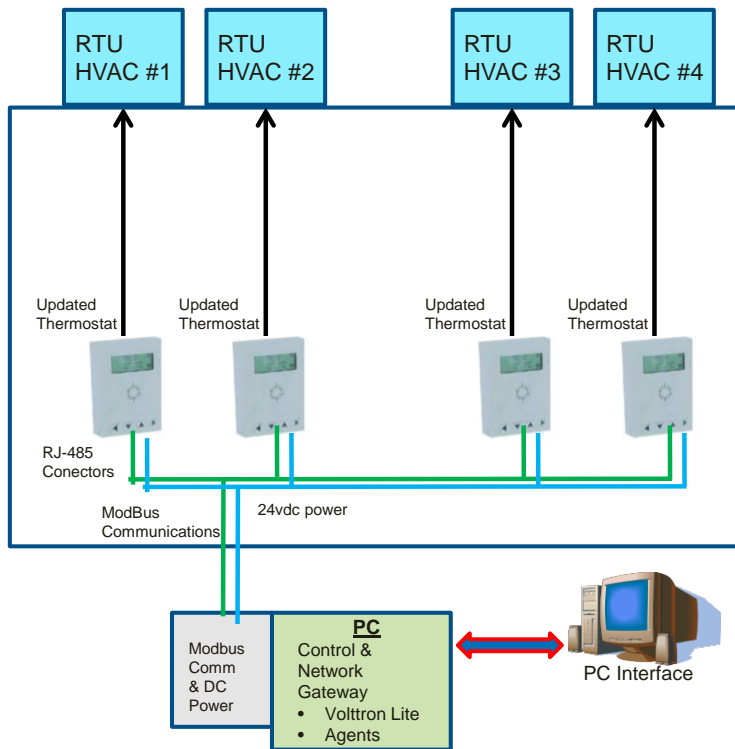


Figure 40: Autonomous Control Applications Deployment Architecture

Figure 41 shows the software components of the Autonomous Control application.

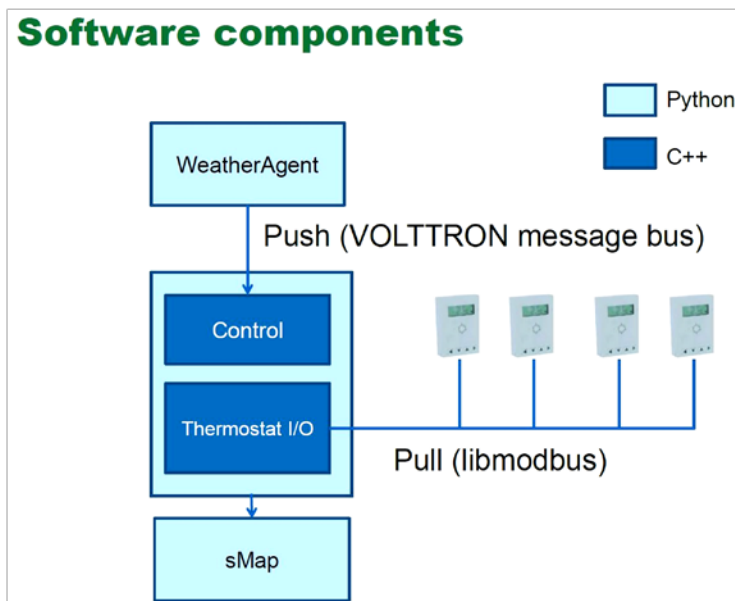


Figure 41: Autonomous Control Application Software Components

3.3.2 Supermarket Refrigeration Application

Supermarkets are an energy intensive operation and stores with floor areas between 3700 – 5600 m² consume between 2 and 3 million kWh annually per store. Refrigeration systems in these stores consume around 50% of the total store energy use. Often the peak energy consumption of the refrigeration systems coincides with the peak demand for electricity, as seen by utilities (mid to late afternoon). These systems can participate in demand response strategies using advanced controller formulations that can respond to utility signals by 1) reducing the number of compressor cycles, 2) shifting peak load time to offset utility peak loads by scheduling and inherent thermal storage, and 3) robust control of the refrigeration systems for efficiency and reduction in faults. Apart from advanced controls to optimize the refrigeration systems, fault detection and diagnosis to improve the refrigeration process can be very beneficial. For example, adjusting the suction pressure based on the ambient temperature for multistage refrigeration systems can improve the energy efficiency by between 8 and 15%. Adaptive fine-grained synchronization of the refrigeration loads provides opportunities for building-to-grid integration potentially beneficial to building owners and local energy markets.

The Supermarket Refrigeration application (developed in collaboration with Emerson) utilizes refrigeration systems to provide energy services to grid and improve the energy efficiency of these systems. Figure 42 shows the deployment architecture for this application:

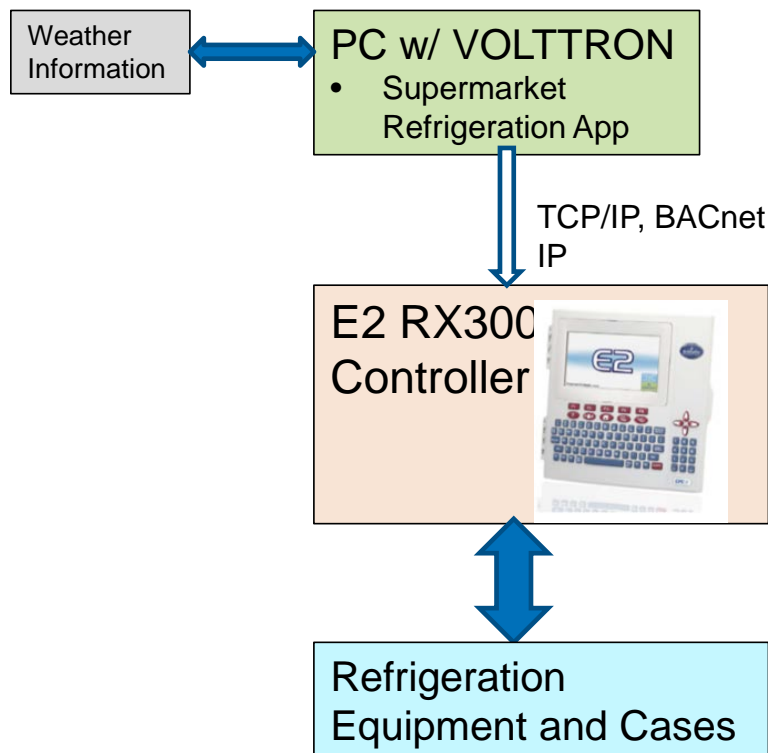


Figure 42: Supermarket Refrigeration Application Deployment Architecture

The Supermarket Refrigeration application has the following features (capabilities):

1. Smart defrost cycle:
 - Demand response and control based on utility signal
 - Intelligent defrost (sensing, algorithms, etc.)

2. Peak power reduction:
 - Coordination of defrost and operation of all cases
 - Capacity modulation for peak power reduction
3. Prognosis of faulty equipment or sensors

3.3.3 Renewable Energy Integration Application

This application is a system for forecasting the average output of a photovoltaic (PV) array one hour in the future. A significant feature of this system is its use of publicly available data, which is obtained through a transactional network, to generate forecasts that are accurate enough to guide control applications. If the forecasting interval can be made more precise, there would be significant potential to use this type of technology to coordinate the availability of building PV arrays with energy intensive building functions; in particular, the autonomous Control of RTUs application could take advantage of such forecasts to improve peak reductions and reduce overall energy usage.

Responsive load can provide benefits both to distribution and transmission systems depending on the scale of implementation, control, and automation. In providing power system support, Oak Ridge National Laboratory (ORNL) will deploy automated control systems on RTUs to provide integration of renewable resources. This will involve accessing sensor information within the RTU and utilizing local renewable resources (photovoltaic solar cells) along with a forecasting application that optimally consume energy when renewables peak in generation.

Figure 43 shows the deployment architecture for the Renewable Energy Integration application.

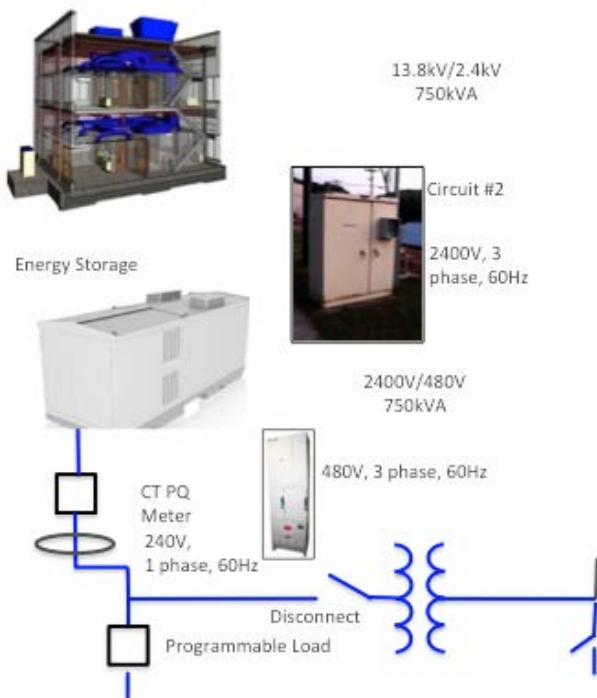


Figure 43: Renewable Integration Application Deployment Architecture

3.3.4 Lighting Diagnostic Application

The Lighting Diagnostic application, developed by Lawrence Berkeley National Laboratory, detects faults in lighting control systems where manual or scheduled control is wasting energy (e.g., lights left on

when room is unoccupied). The application suggests improvements to schedules to better match lighting to occupant needs. The fault detection models operate on usage data like relay state, override times, programmed lighting schedule, and lighting power load.

Figure 44 shows an overview of the lighting diagnostic's functionality including input and output information.

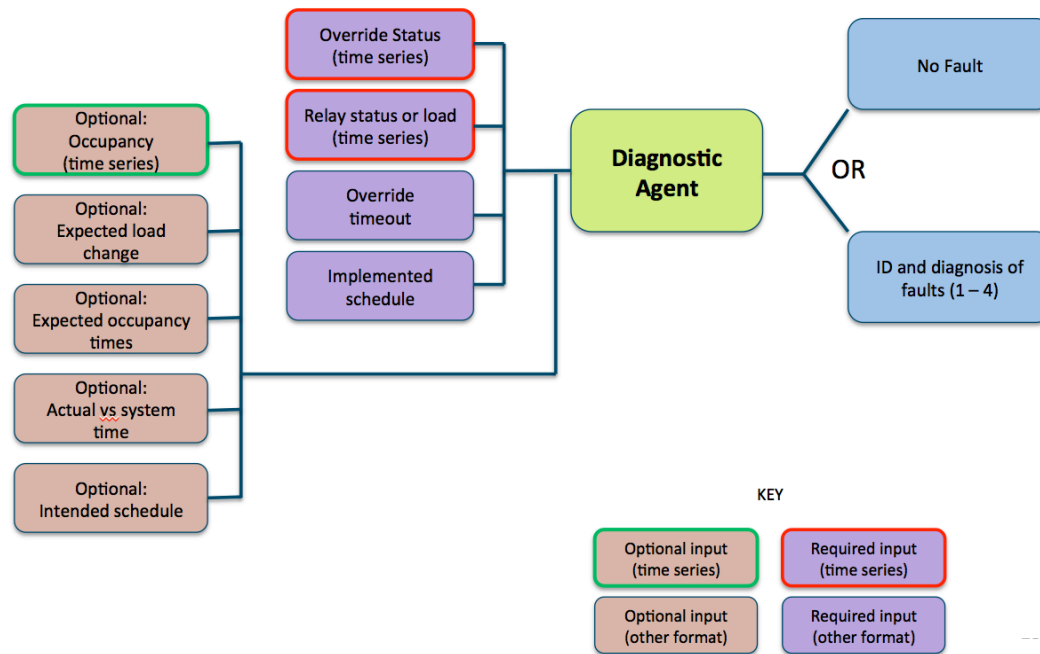


Figure 44: Lighting Diagnostic Overview

Figure 45 shows the deployment architecture for the Lighting Diagnostic application.

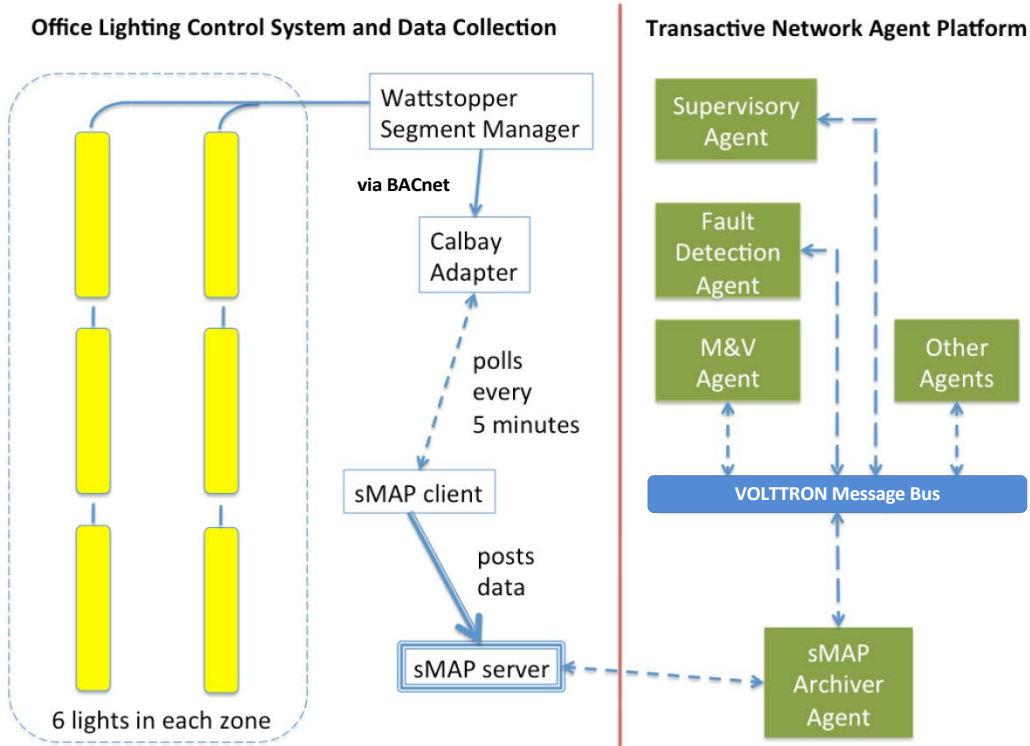


Figure 45: Deployment Architecture for the Lighting Diagnostic Application

The work-flow for the Lighting Diagnostic application is as follows:

- “Supervisory agent” is run - one of its tasks is to schedule device interaction (device interaction via the Actuator agent)
- Supervisory agent requests data from sMAP data archive
- Upon provision of data, the Supervisory agent calls the measurement and verification (M&V) application and the Lighting Diagnostic application
- Data for the agents is delivered via the VOLTTRON message bus
- Applications send the results back to the Supervisory agent via the VOLLTRON message bus
- The Supervisory agent reads the results from the message bus and writes them to local files for review.

3.3.5 Baseline Load Shape Application

The objective of the Baseline Load Shape application is to provide a baseline model that accounts for the influence of the outside-air temperature and previous loads by time of day and day of week. Baseline models provide a comparison for determining the energy savings arising from efficiency efforts or power savings for demand response events, by predicting building electric loads based on recent information. Measuring the response, in the form of energy use (kWh) or peak demand (kW), to a change in building energy operations requires a reference case based on historical energy use. The Baseline Load Shape application produces two baselines, one based on long term trends for energy efficiency measures, and one based on the last 2 weeks of data for demand response events. The short-term model accounts for recent changes in load shapes or schedules.

The following list describes the Baseline Load Shape application inputs:

- Historical whole building power (for at least 2 weeks), sampled in intervals of 1 hour or less
- Outside-air temperature, sampled at least hourly
- Event period information

The following list describes the Baseline Load Shape application outputs:

- Long-term predicted whole-building baseline load as a function of time.
- Short-term predicted whole-building baseline load as a function of time.
- Goodness of fit statistics:
 - Standard error of the residuals during the “training” period
 - Correlation coefficient.

Figure 46 shows an example of a baseline model developed by the Baseline Load Shape application:

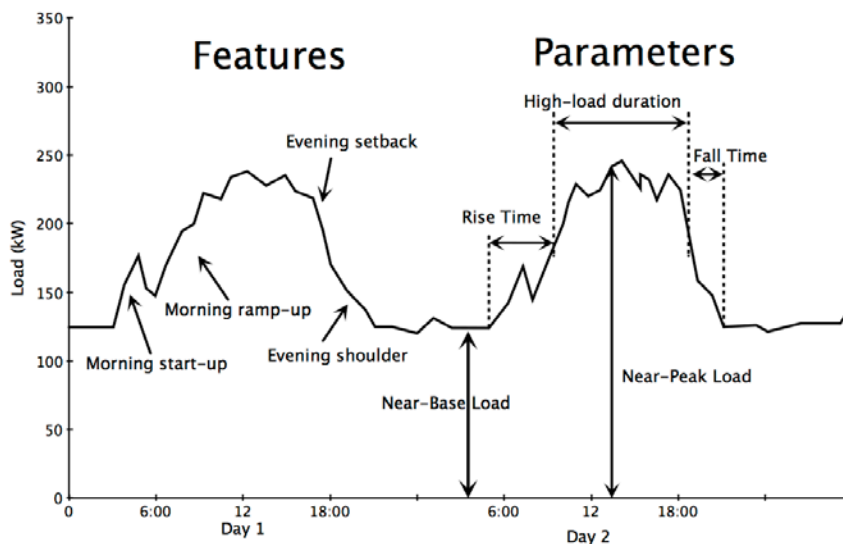


Figure 46: Regression Model Developed by the Baseline Load Shape Application

3.3.6 Measurement and Verification Application

The Measurement and Verification (M&V) application provides a measurement tool that automates the development of a standard baseline model to measure changes in whole-building energy use over a given period before and after an intervention (energy efficiency improvement measure). Measuring the savings in energy use (kWh) from a change in the building operation provides whole-building performance data to help evaluate energy efficiency strategies. The total energy saved or the reduction in peak demand can also be translated into financial savings to help evaluate the importance of the energy efficiency improvement measure.

The following list describes the M&V application inputs:

- Interval load data (1-hour frequency or faster)

- Long-term baseline load shape predicted before an efficiency action (developed by the baseline Load Shape application)
- Date at which the efficiency action was undertaken
- Tariff data – time of use costs. \$/kWh by time of use period

The following list describes the M&V application outputs:

- Difference between actual load and baseline load for each time interval after the efficiency action
- Cumulative savings from the efficiency action to the present
- Energy savings expressed in kWh/ft²
- Energy savings expressed as a percent of whole-building baseline load
- Total electricity cost for each 15-minute time interval
- Total electricity cost for baseline operation, for each 15-minute time interval
- Extra cost or reduced cost (\$) for each 15-minute time interval
- Cumulative savings (\$) over a specified set of time intervals
- Cumulative savings for an entire efficiency action

3.3.7 Smart Monitoring and Diagnostic System Application

The objective of the Smart Monitoring and Diagnostic System (SMDS) application is to detect degradation (or improvement) in the cooling performance of packaged air conditioners and heat pumps entirely automatically and using a minimum number of sensed variables. The SMDS application resides in the Cloud. Measured data are collected at 1-minute intervals for outside-air temperature, RTU total power consumption, and supply-fan speed signal (for RTUs with variable-speed supply fans only) by the VOLTRON platform for each RTU connected to the network. The data are then stored in the network's Data Historian from which the data are periodically retrieved by an agent in the platform and sent to the application in the Cloud, where the data are stored in a local application database. The data are processed once per day to provide updated results. The SMDS detects when changes occur in the performance of the refrigerant side of RTUs. These changes can be degradations in performance from faults or improvements in performance associated with servicing. When a change in performance is detected, the SMDS estimates the energy and cost impacts of the change. In the case of improved performance, the SMDS can be used to determine the impacts of servicing on energy use and costs. For degradations in performance, the SMDS provides information that supports the building owner, manager or operator in deciding when to schedule servicing based on the impacts of the performance degradation that occurred.

The following list describes the SMDS applications approach for detecting changes in heating, ventilation and air conditioning (HVAC) equipment performance:

- Characterize the performance of the RTU cooling cycle with an empirically determined relationship between electric power demand (P) and outside-air temperature (OAT) for times when the unit is operating at steady state.
- This is possible because steady-state power consumption is not affected by changes in cooling load (e.g., from changes in thermostat setting).

- Define steady-state operation as times when the power demand does not change appreciably between successive measurement times.
- Detect changes in the P vs. OAT relationship over time, which correspond to changes in the performance of the RTU cooling function (efficiency, capacity or both).
- Determine the energy consumption impact of the detected change to establish whether performance has degraded or improved.
- The sign (positive or negative) of the detected average change in the P-OAT relationship does not reveal whether performance degradation or improvement has occurred; the sign of the change in energy use reveals this.
- Monitor P vs. OAT characteristics over time to detect when changes occur and quantify energy impacts when detected.

Figure 47 shows the basic methodology for the SMDS application:

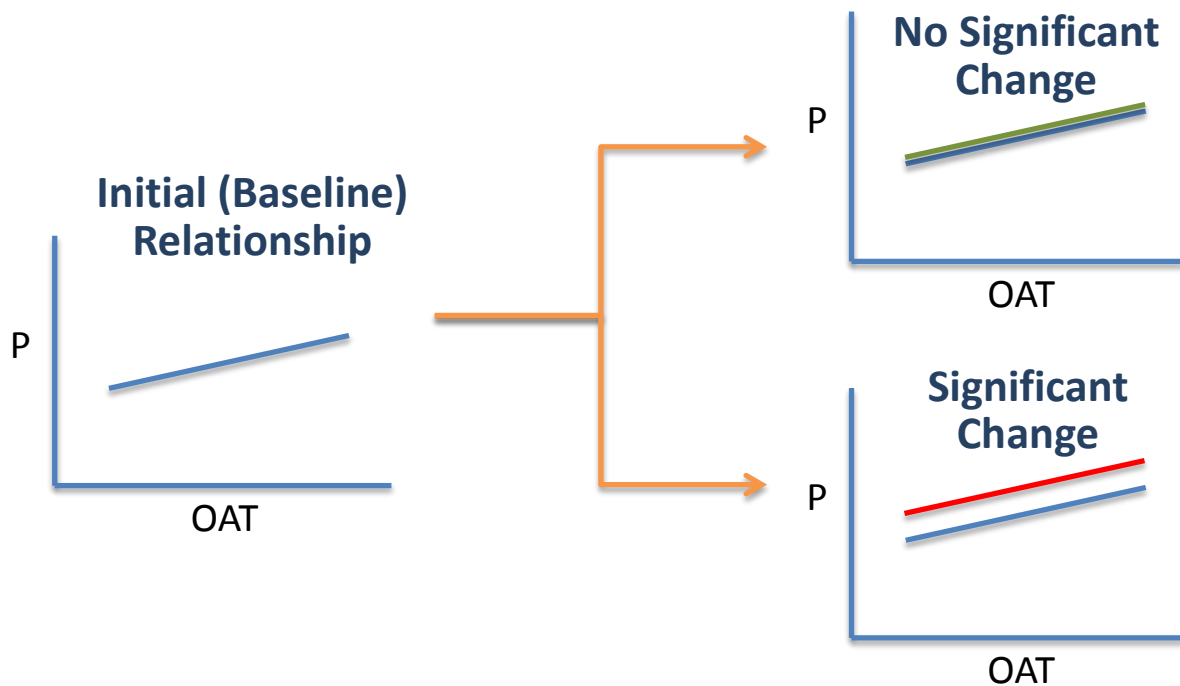


Figure 47: Basic Methodology of the SMDS Application

The SMDS also detects the following operational faults that usually can be corrected with simple changes in control parameters:

- Supply fan always on (24 hours per day)
- Compressor always on (24 hours per day)
- RTU always off
- Supply fan cycles with the compressor only.

When any of these faults is detected, the SMDS provides an alert to users through the user interface. The measured inputs for the SMDS are:

1. outside-air temperature
2. total RTU power use
3. supply-fan speed signal as a percentage of full speed, all measured at 1-minute time intervals.

The outputs from the SMDS include:

1. indicators that a performance change has occurred
2. the increase or decrease in energy use resulting from the performance change
3. indications that each of the operation faults has occurred or not occurred
4. supplemental information regarding the nature of each operation fault and the significance of its impacts.

No configuration or set-up data entry by the user is required. All information on the nature of each RTU is measured or inferred. Figure 48 shows the deployment architecture for the SMDS application:

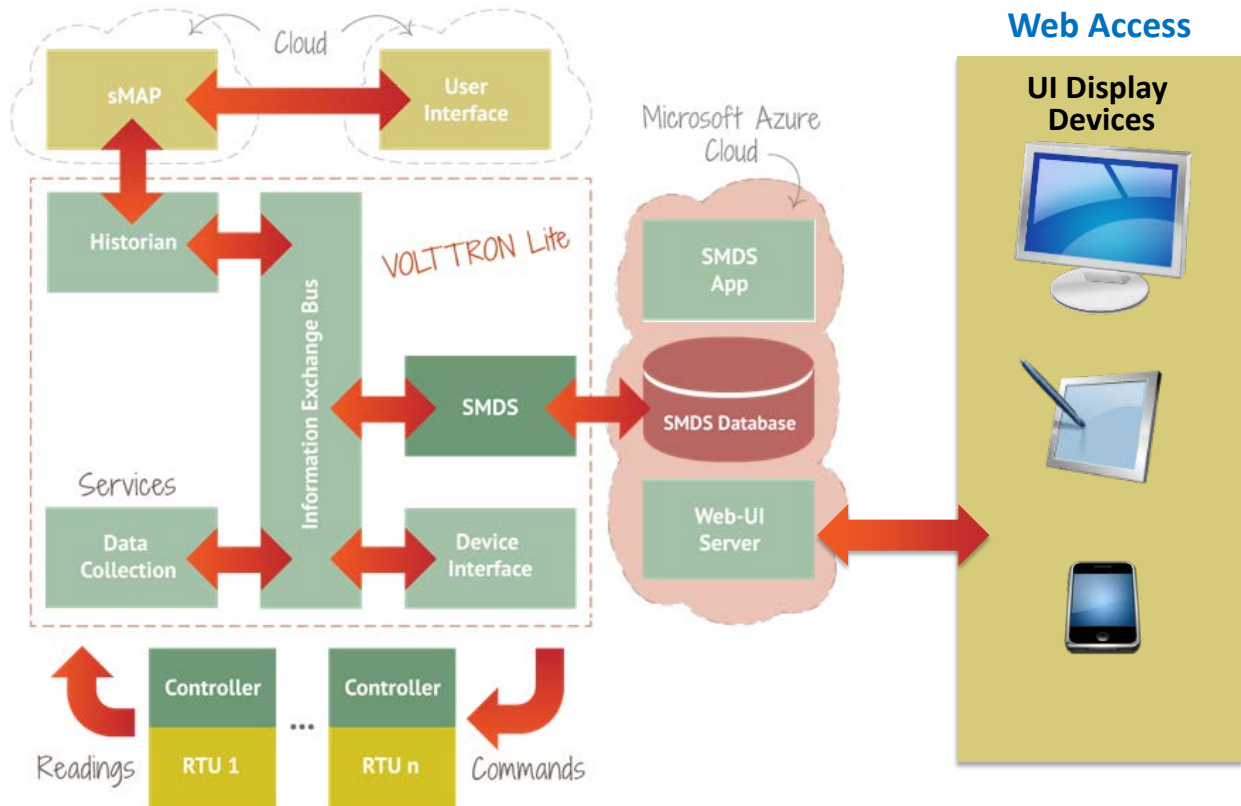


Figure 48: Deployment Architecture for the SMDS Application

3.3.8 Analytical Hierarchy Process for Load Curtailment Application

The Analytical Hierarchy Process (AHP) application implements a load curtailment strategy as an energy cost saving approach to control electric power usage when the peak electric usage is above a user-specified maximum electric usage. This approach has been used to limit an electric demand load where a demand electric charge is a significant percentage of the total energy cost in the building or when a

building has to maintain a certain level of maximum demand in response to changes in the price of electricity over time. The AHP approach allows the HVAC system to provide the comfort space cooling or heating while curtailing the electric usage during the expected peak hours. Duty-cycling control is a typical approach to control the peak load curtailment by controlling the ratio of on-period to total cycle time. This approach has long been used in rooftop units (RTUs) to save energy and extend the life of system. There are two methods to provide the duty cycling. The first method is parallel duty cycling, which cycles RTUs on and off at the same time. The second method is a stagger duty cycling. This approach provides the duty cycling to stagger the on and off times while spreading the distribution time to each RTU. For example, in case of the staggered duty cycling only some (e.g., 1/3 or 2/3) of the RTUs operate at any given time. Although both duty cycling approaches provide electric demand relief, those approaches do not dynamically prioritize the loads to be curtailed. This indiscriminate curtailment of RTUs can lead to comfort issues in the zone.

The load curtailment algorithm utilized by the AHP application is the staggered duty cycling control with dynamic prioritization of the loads. The prioritization is established via a priori list of curtailable RTUs. The RTUs can be curtailed by the prioritization based on a number of parameters including human comfort, cooling or heating load and system input power. The AHP approach is used to determine the curtailment load priority.

The curtailment decision for RTU system can be implemented at a low cost within a supervisor controller of RTUs without adding any extra physical sensors. The supervisor controller would only need to receive utility signals corresponding to each RTU to generate the priority list of RTUs. Once the priority list of load curtailment is created, the number of RTUs that need to be curtailed to meet the peak electric usage can be determined. The AHP process can also be extended to some advanced control feature such as precooling and preheating conditions that can alleviate uncomfortable conditions when the operation of RTUs are limited during demand limiting periods.

4 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.agent`” package contains these utilities (`~/volttron/volttron/platform/agent/`). For the details of using them, please see the Wiki or read the comments in the code. These comments contain explanations and examples of usage.

The utility classes in this package, with a brief description, follow:

- **base.py**: BaseAgent class handles much of the low-level requirements for working in the platform. An agent that extends the BaseAgent can focus on its own functionality and override only the methods it needs.
- **matching.py**: Allows agents to easily subscribe to topics using decorators⁸
- **cron.py**: Allows agents to schedule their actions ahead of time
- **green.py**: Utilities for using greenlets⁹ with the BaseAgent
- **sched.py**: Used for scheduling objects
- **utilities.py**: Utilities for loading config files, parsing command line arguments, formatting log messages, etc.

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

4.1 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.

4.2 Explanation of Listener Agent

The Listener agent utilizes the PublishMixin and BaseAgent classes for its base functionality. Please see `volttron/volttron/agent/base.py` for the details of these classes.

The Listener agent publishes a heartbeat message using the PublishMixin. It also extends BaseAgent to get default functionality such as responding to platform commands. When creating agents, Mixins¹⁰ should be first in the class definition.

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

⁸ <https://wiki.python.org/moin/PythonDecorators>

⁹ <https://pypi.python.org/pypi/greenlet>

¹⁰ <http://python.dzone.com/articles/mixins-python>

The Listener agent subscribes to all topics by using `volttron/volttron/agent/matching.py`. This package contains decorators for simplifying subscriptions. The Listener agent uses `match_all` to receive all messages:

```
@matching.match_all
def on_match(self, topic, headers, message, match):
    '''Use match_all to receive all messages and print them out.'''
    _log.debug("Topic: {topic}, Headers: {headers}, "
               "Message: {message}".format(
                   topic=topic, headers=headers, message=message))
```

The Listener agent uses the `@periodic` decorator to execute the `pubheartbeat` method every `HEARTBEAT_PERIOD` seconds where `HEARTBEAT_PERIOD` is specified in the `settings.py` file. To publish, it creates a Header object to set the `ContentType` of the message, the time the event was created, and the ID of the agent sending it. This allows other agents to filter messages of a certain type or from a certain agent. It also allows them to interpret the content appropriately. The message it then publishes on the heartbeat topic.

```
#Demonstrate periodic decorator and settings access
@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.publish('heartbeat/listeneragent', headers, now)
```

To see the Listener agent in action, please see Section 2.7.

4.3 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:

<https://www.eclipse.org/downloads/download.php?file=/technology/epp/downloads/release/luna/SR1/eclipse-java-luna-SR1-linux-gtk.tar.gz>

For 64-bit machines:

https://www.eclipse.org/downloads/download.php?file=/technology/epp/downloads/release/luna/SR1/eclipse-java-luna-SR1-linux-gtk-x86_64.tar.gz

This link will take you to the main Eclipse webpage:

<http://www.eclipse.org/>

4.3.1 Installing Eclipse

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

1. Install Eclipse dependency:
sudo 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-luna-SR1-linux-gtk-x86_64.tar.gz
sudo mv eclipse /opt/
 - For 32-bit machines, remove “x86_64” from the previous command.
3. Create desktop shortcut:
sudo touch /usr/share/applications/eclipse.desktop
sudo nano /usr/share/applications/eclipse.desktop
 - Enter the following text, as shown in Figure 49, and save the file.

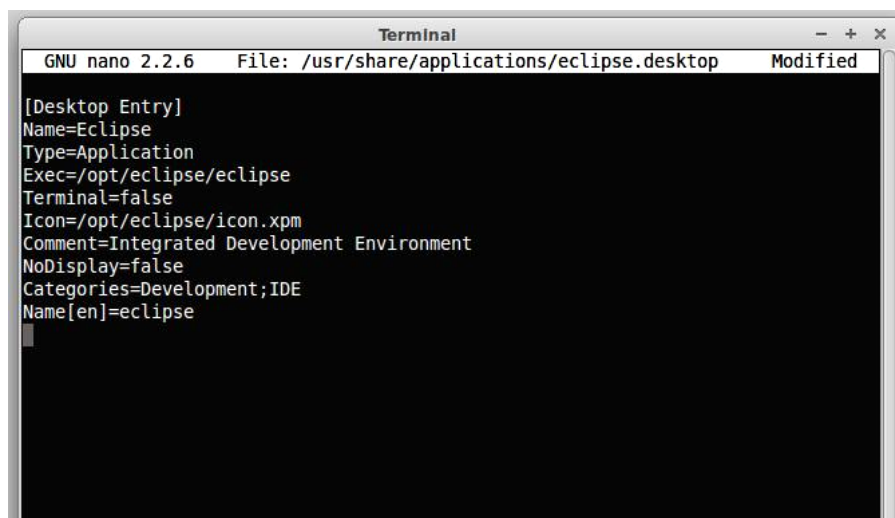


Figure 49: Eclipse Desktop File

4. Copy to shortcut to the desktop:
cp /usr/share/applications/eclipse.desktop ~/<USER>/Desktop/

Eclipse is now installed and ready to use.

4.3.2 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 50:
 - Select: Help -> Install New Software

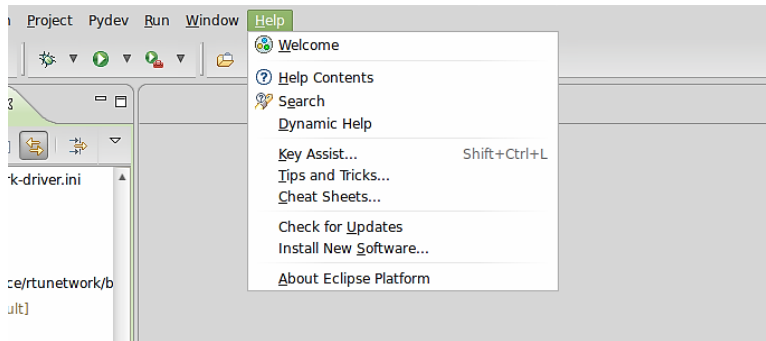


Figure 50: Installing Eclipse EGit Plug-in

- Click on the "Add" button as shown in Figure 51.

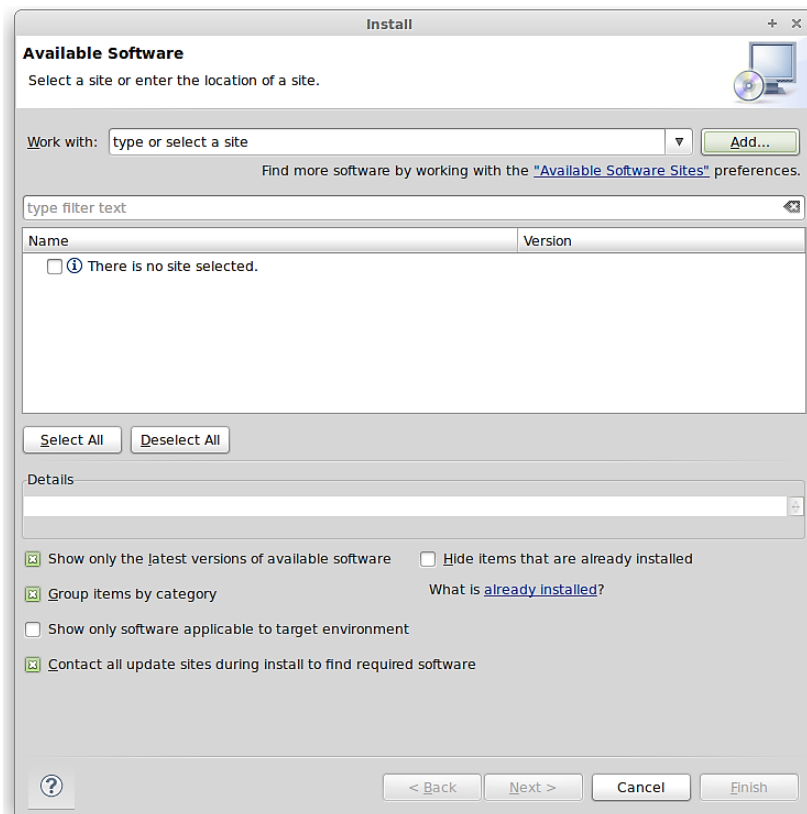


Figure 51: Installing Eclipse Egit Plug-in (continued)

- As shown in Figure 52 enter the following:
 - For name use: EGit
 - For location: <http://download.eclipse.org/egit/updates>

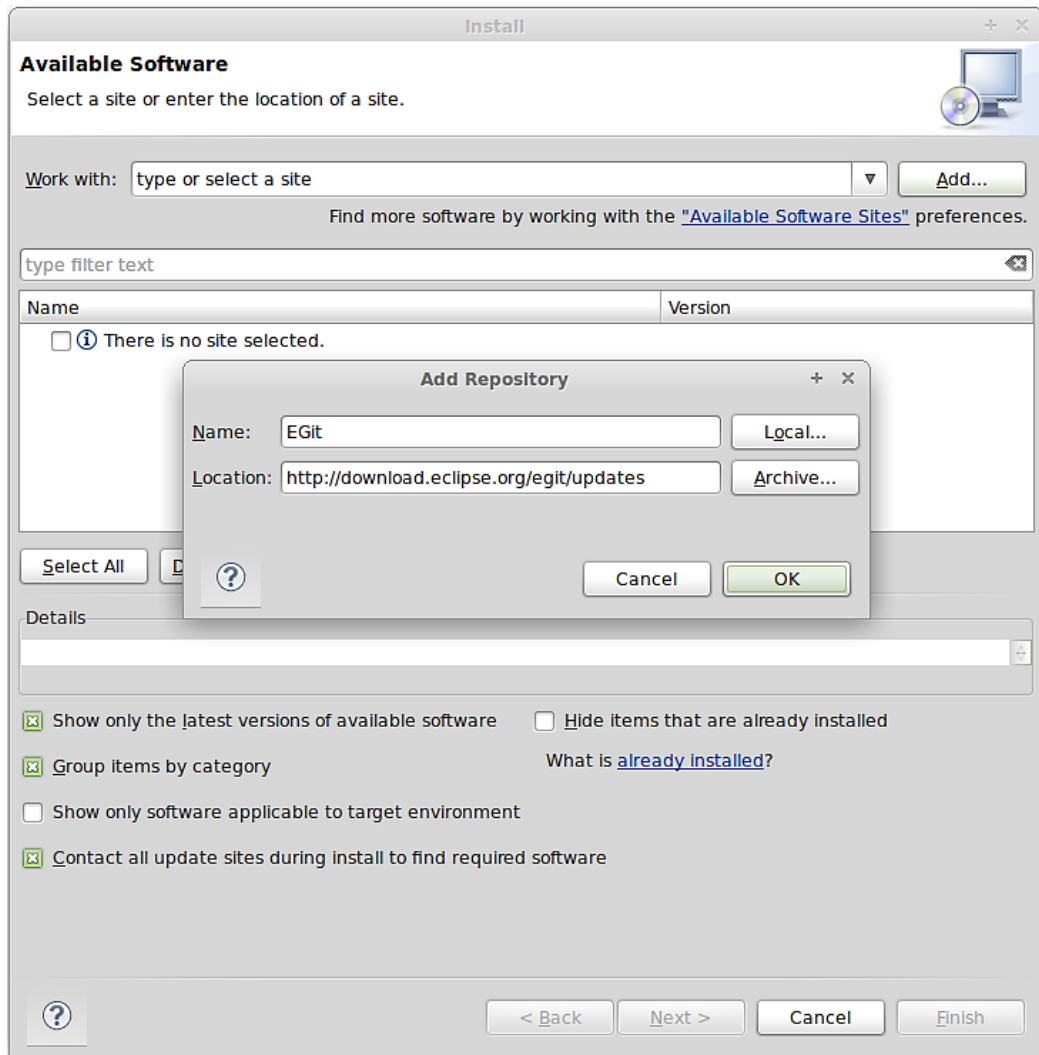


Figure 52: Installing Eclipse Egit Plug-in (continued)

- After hitting OK, check the Select All button
- Click through Next, Agree to Terms, then 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 53 enter the following:
 - For name use: PyDev
 - For location: <http://pydev.org/updates>

- Click OK

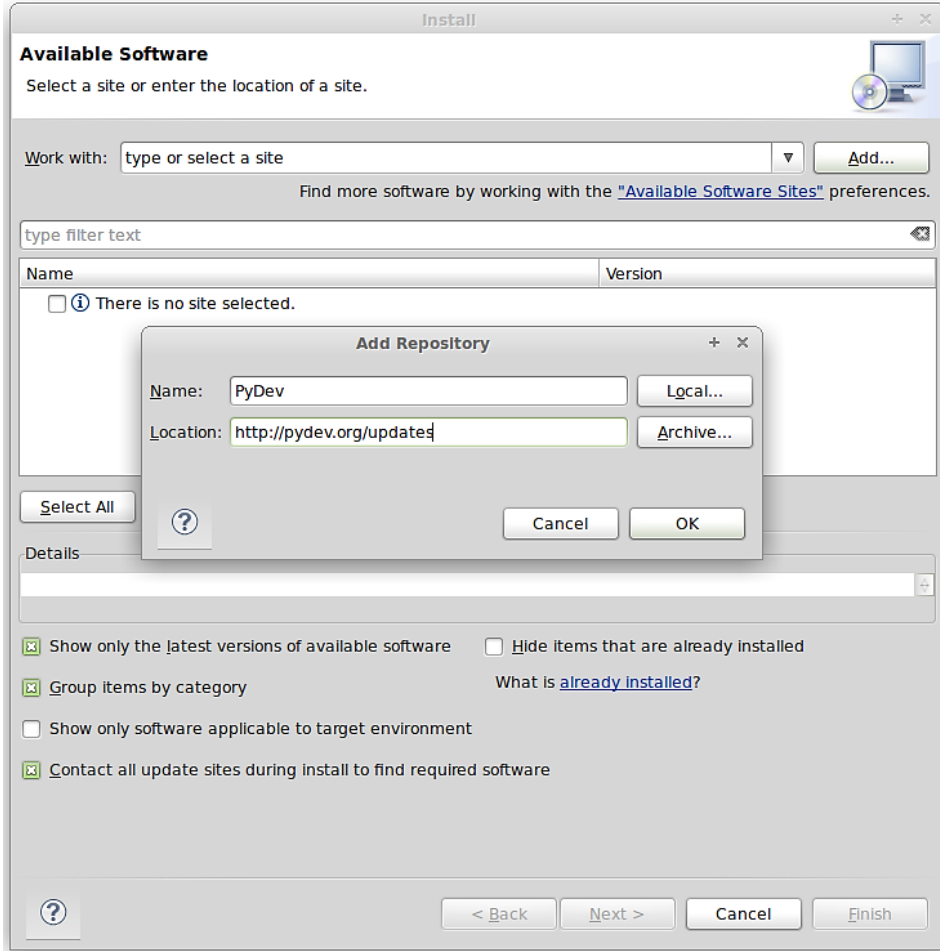


Figure 53: Installing Eclipse PyDev Plug-in

- Check the box for PyDev
- Click through Next, Agree to Terms, Finish
- Allow Eclipse to restart

4.3.3 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.

4.3.3.1 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 54

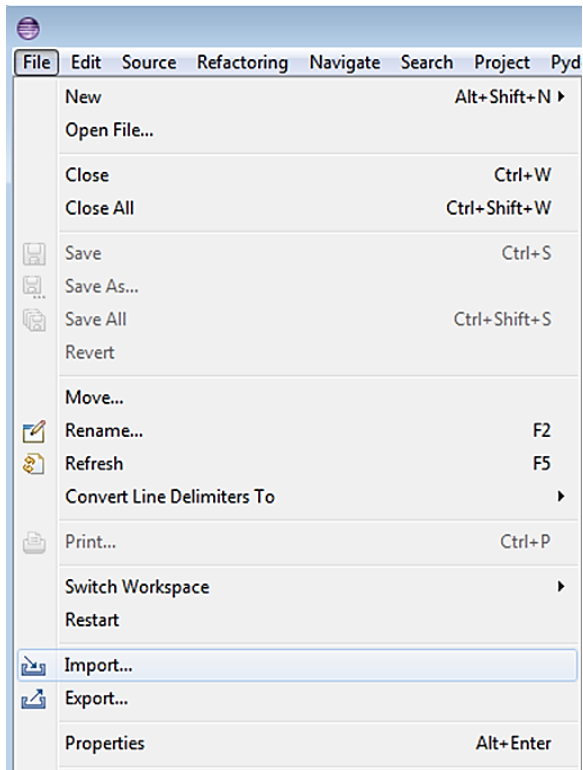


Figure 54: Checking VOLTRON with Eclipse From Local Source

2. Select Git -> Projects from Git, then click the Next button (Figure 55)

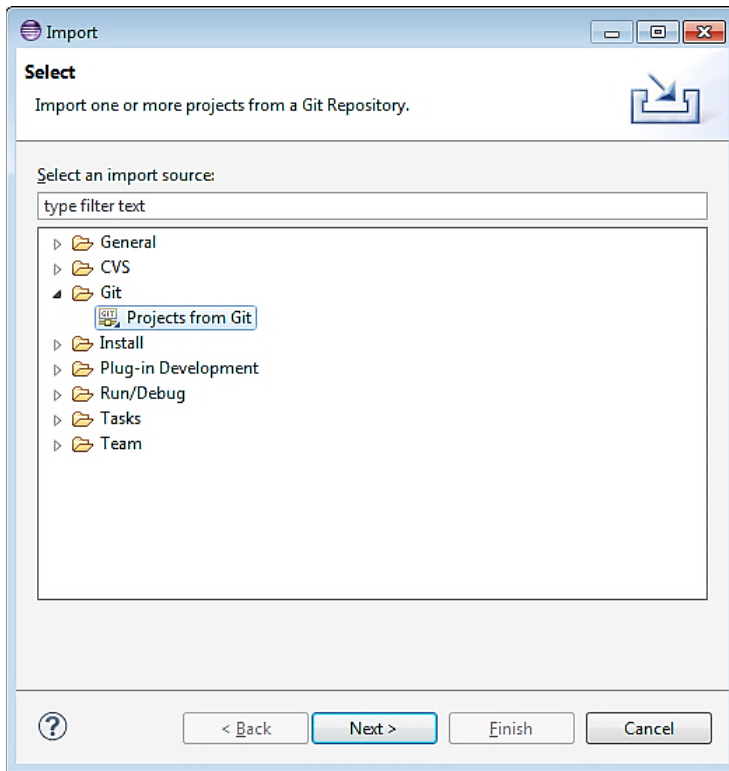


Figure 55: Checking VOLTRON with Eclipse from Local Source (continued)

3. As shown in Figure 56:
 - o Select Existing local repository -> Next >

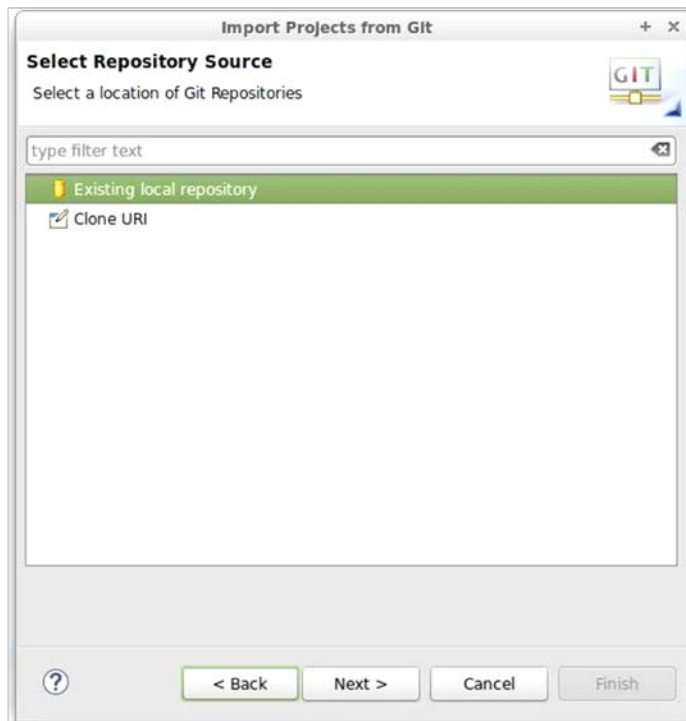


Figure 56: Checking VOLTRON with Eclipse from Local Source (continued)

4. Select Add (Figure 57)

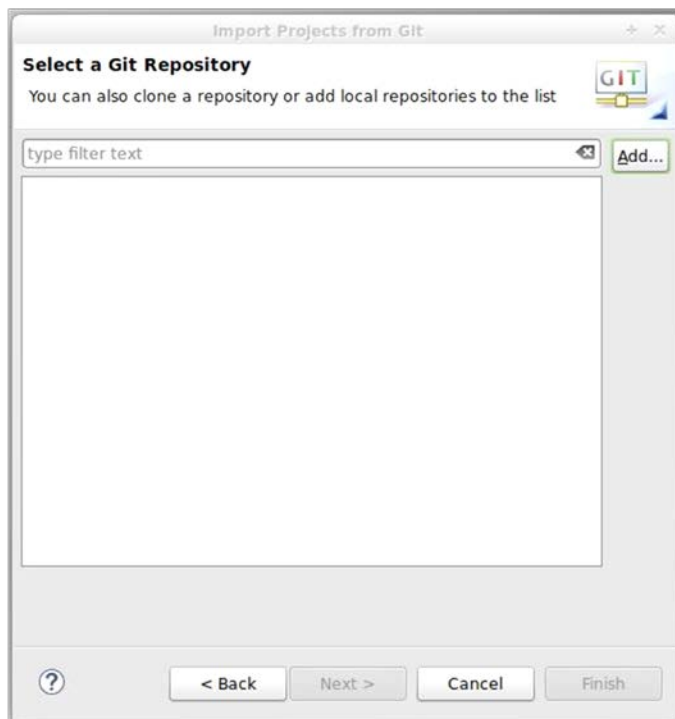


Figure 57: Checking VOLTRON with Eclipse from Local Source (continued)

5. Select Browse -> navigate to the top-level volttron directory and select OK (Figure 58)

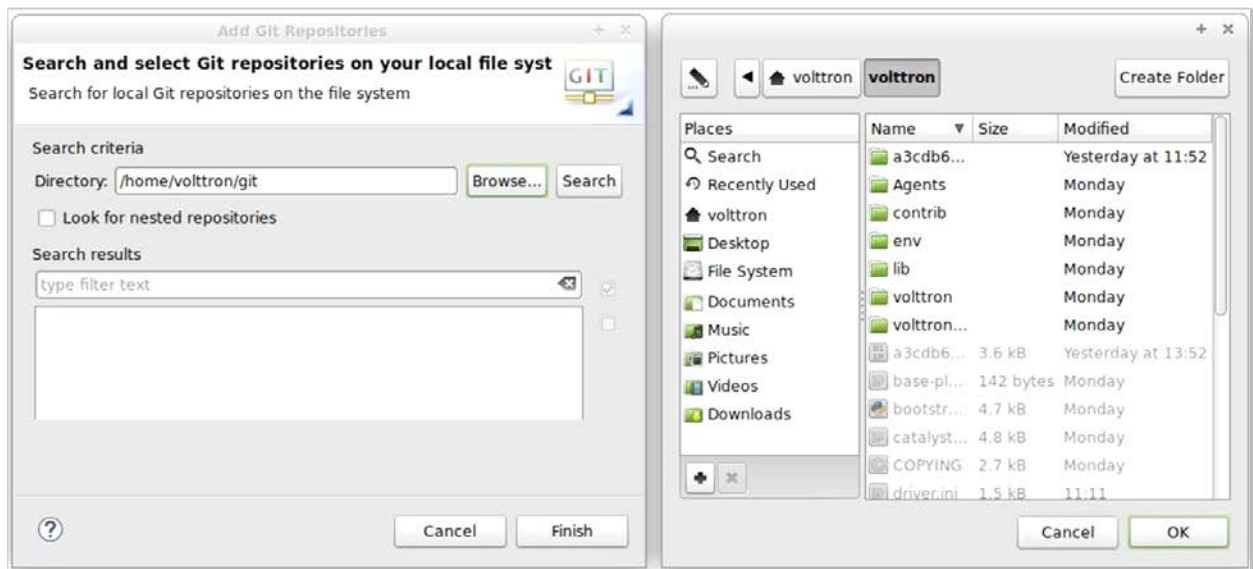


Figure 58: Checking Out VOLTRON with Eclipse from Local Source (continued)

6. Select Finish (Figure 59)

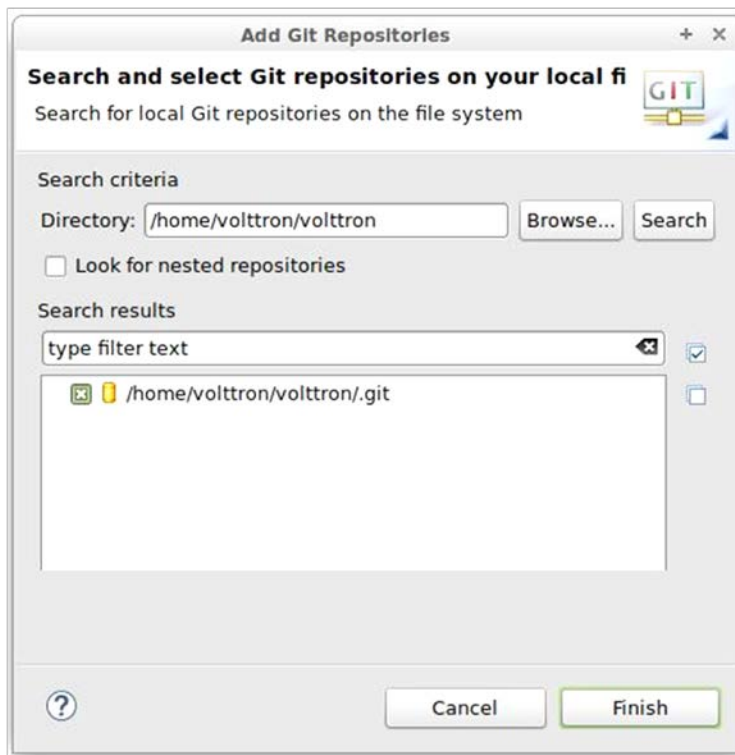


Figure 59: Checking Out VOLTRON with Eclipse from Local Source (continued)

7. Choose Import as general project and click Next -> Finish, the project will be imported into the workspace (Figure 60)

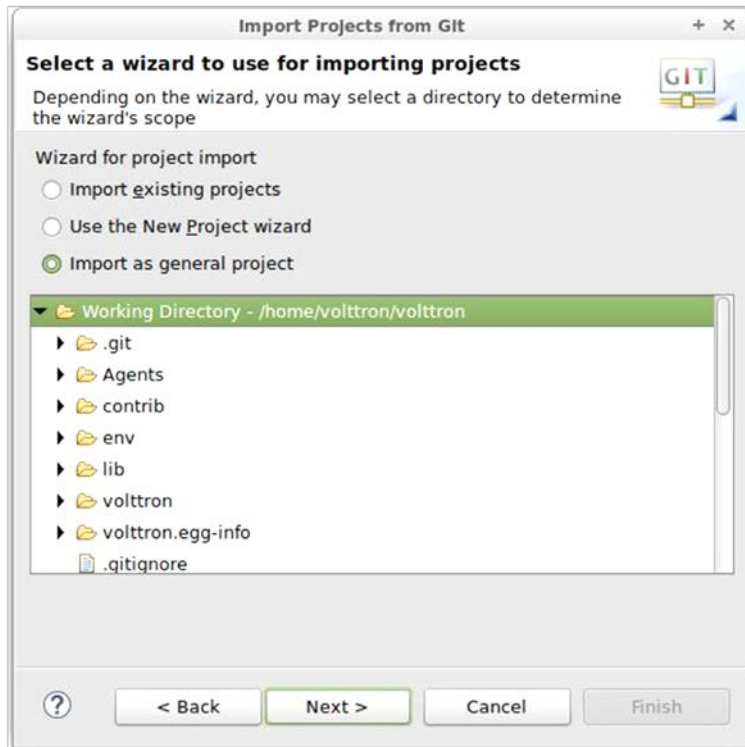


Figure 60: Checking Out VOLTTRON with Eclipse from Local Source (continued)

4.3.3.2 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 61)

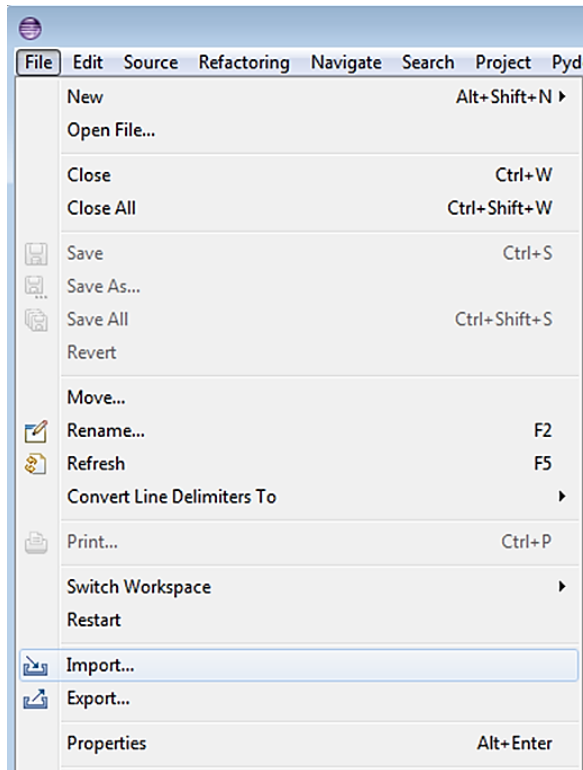


Figure 61: Checking Out VOLTTRON with Eclipse from GitHub

2. Select Git -> Projects from Git, then click the Next button (Figure 61)

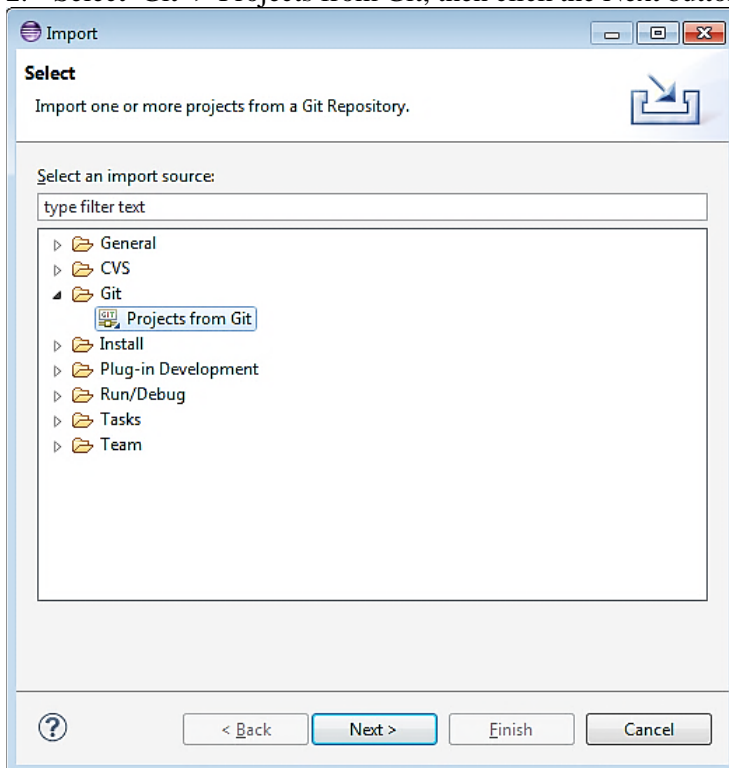


Figure 62: Checking Out VOLTTRON with Eclipse from GitHub (continued)

3. As shown in Figure 63 select Clone URI -> Next >

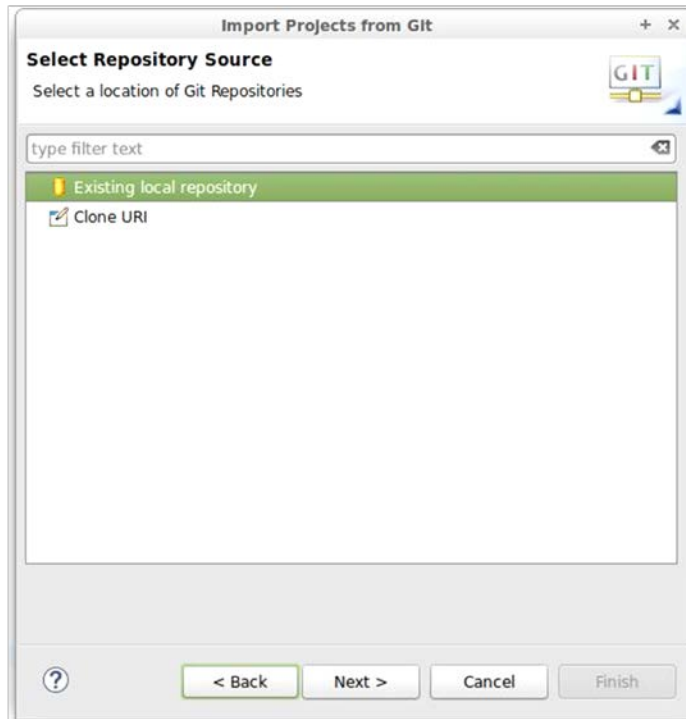


Figure 63: Checking Out VOLTTRON with Eclipse GitHub (continued)

4. Fill in <https://github.com/VOLTTRON/volttron.git> for the URI, and use your GitHub account login (GitHub account username and password in the User and Password, as shown in Figure 64)

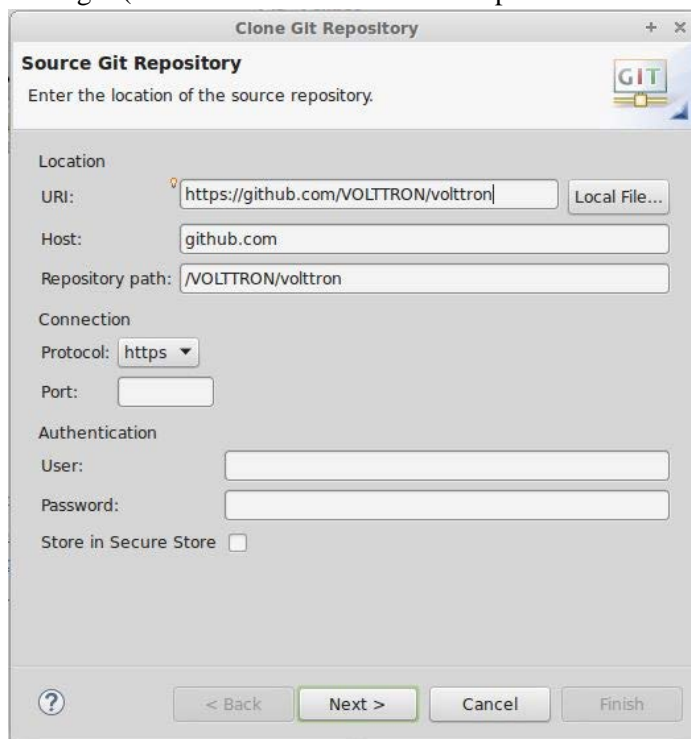


Figure 64: Checking Out VOLTTRON with Eclipse from GitHub (continued)

5. Select the 2.x branch (Figure 65)

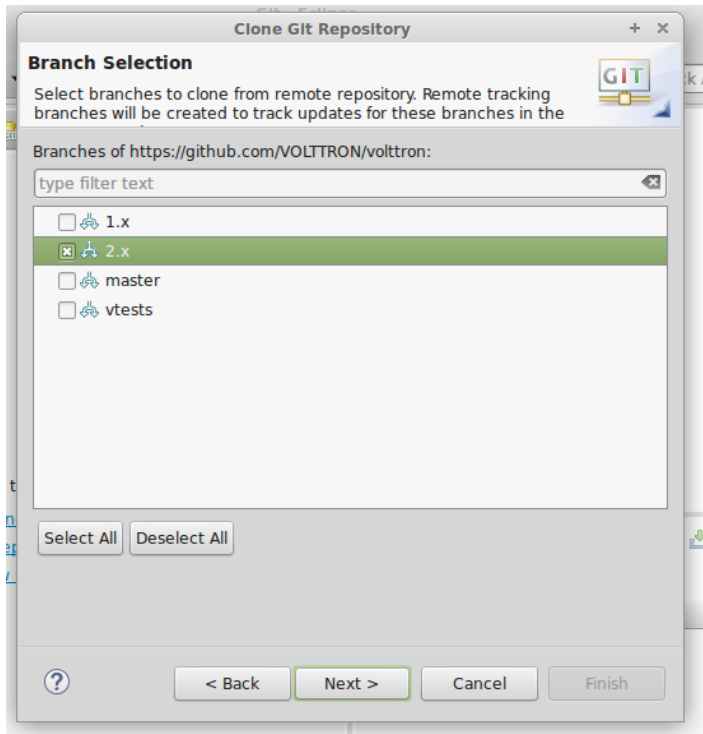


Figure 65: Checking Out VOLTRON with Eclipse from GitHub (continued)

6. Select a location to save the local repository (Figure 66)

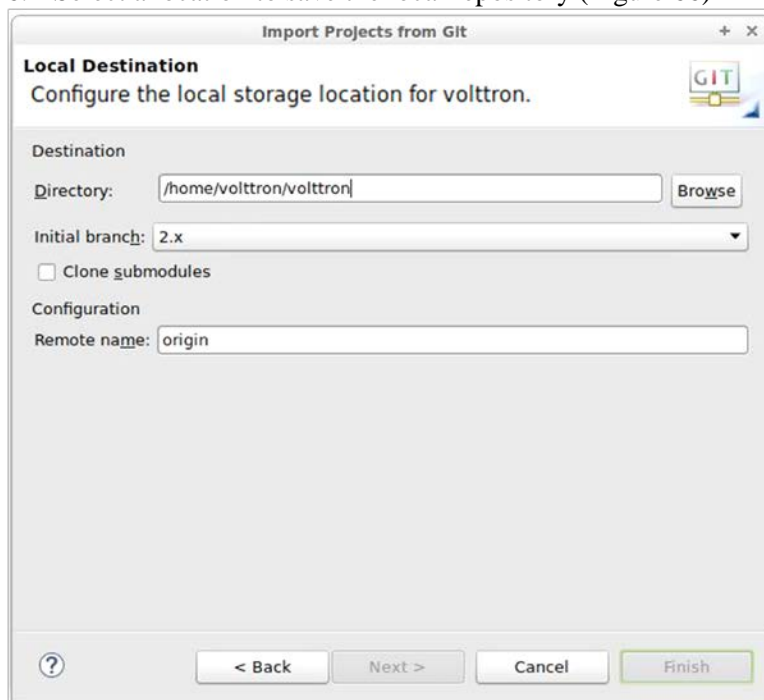


Figure 66: Checking Out VOLTRON with Eclipse from GitHub (continued)

7. Select Import as general project, select Next, then select Finish (Figure 67), the project will now be imported into the workspace

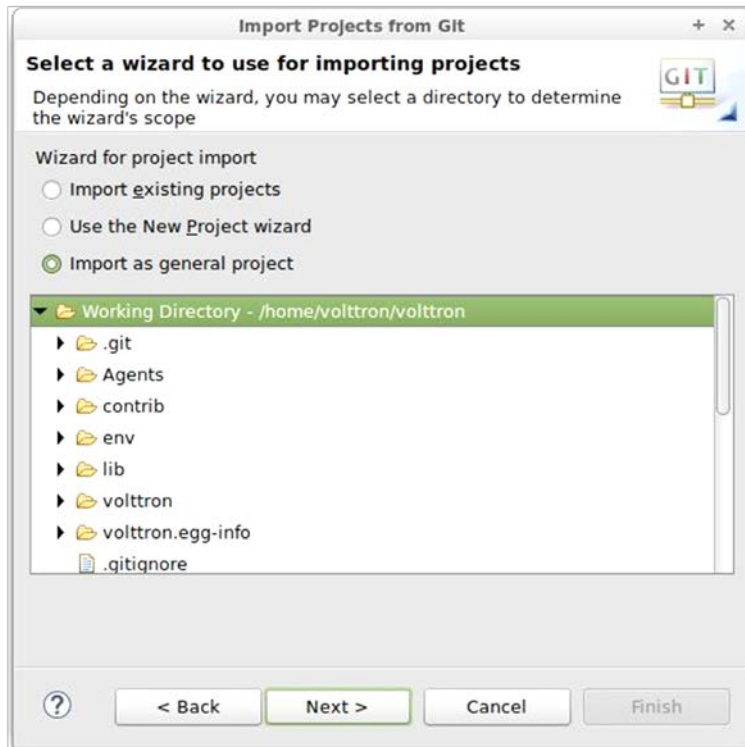


Figure 67: Checking Out VOLTTRON with Eclipse from GitHub (continued)

The project must now be built outside Eclipse. Please follow the directions in Section 2.6. After changing the file system outside Eclipse, right-click on the project name and select Refresh.

4.3.4 Configuring Eclipse

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

- Select Window-> Preferences
- Expand PyDev
- Select Interpreter-Python
- Hit New
- Use Python as the Interpreter Name, and hit Browse. In the bin directory for the VOLTTRON project, select the file named Python, Then hit OK (Figure 68)

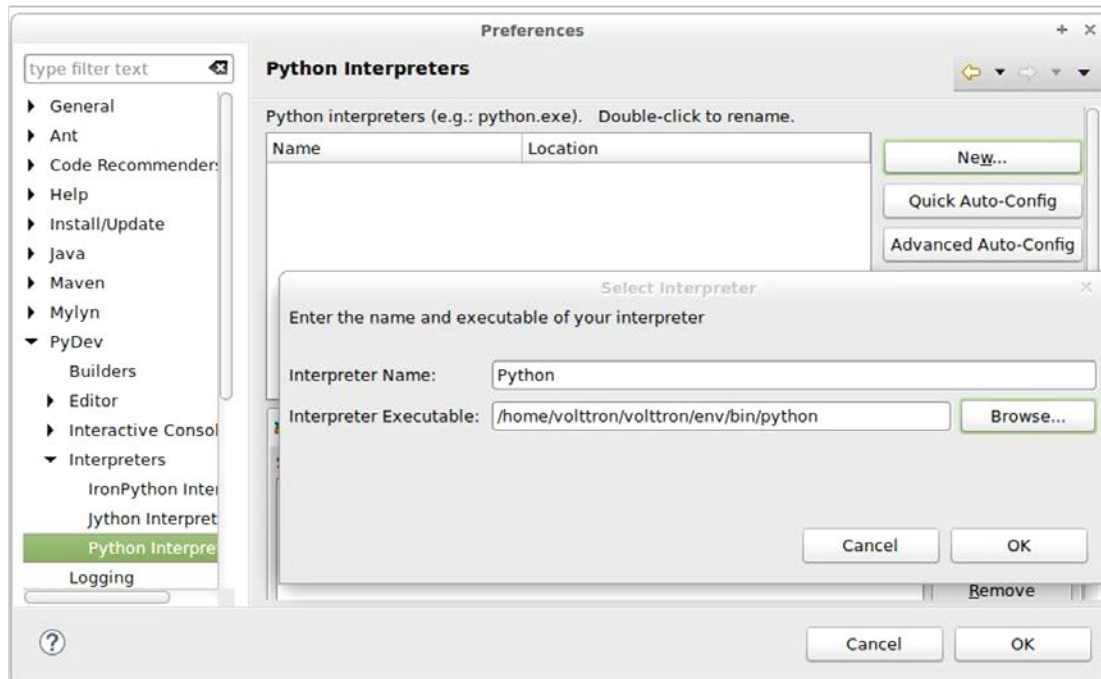


Figure 68: Configuring PyDev

- Select All, then select OK (Figure 69)

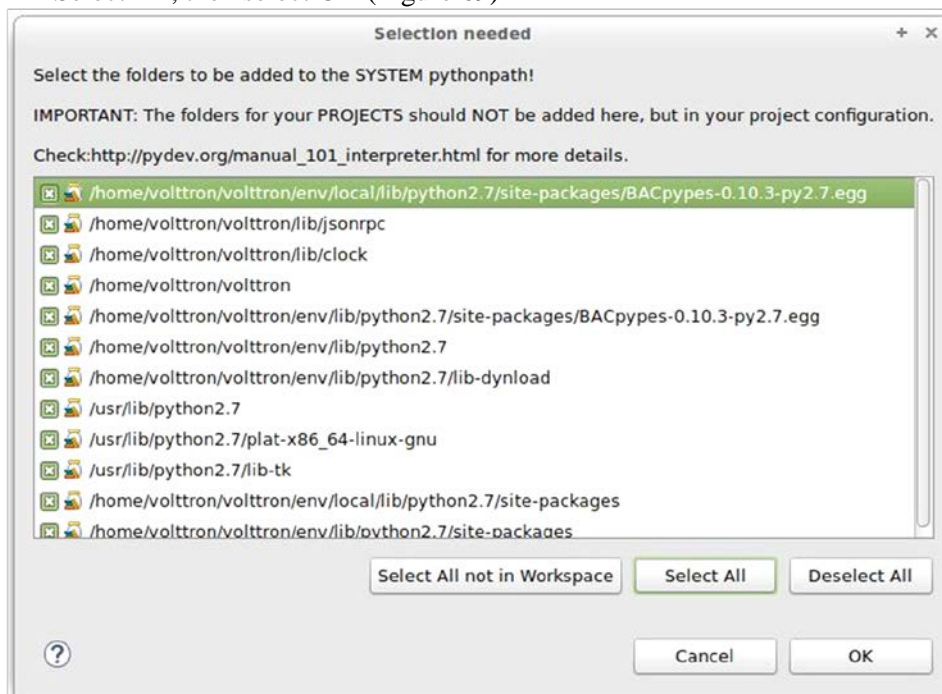


Figure 69: Configuring PyDev (Continued)

- In the Project/PackageExplorer view on the left, right-click on the project, PyDev-> Set as PyDev Project (Figure 71)

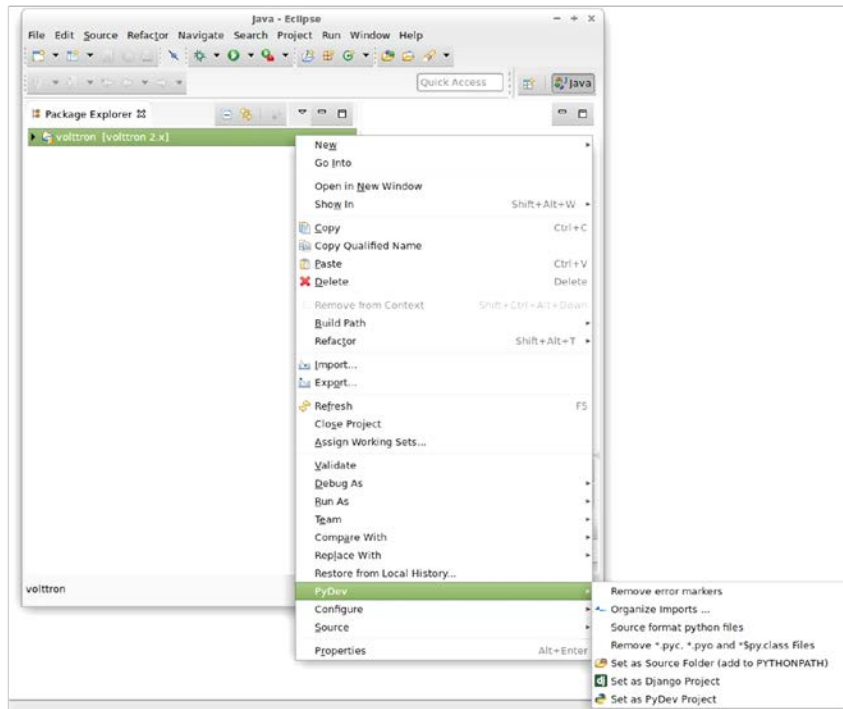


Figure 70: Setting as PyDev Project

Switch to the PyDev perspective (Figure 71).

- Window -> Open Perspective -> PyDev or Window -> Open Perspective -> Other -> PyDev

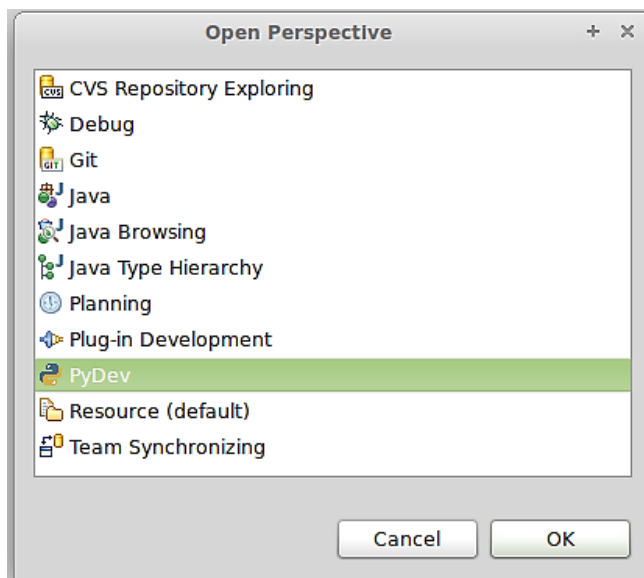


Figure 71: Setting PyDev Perspective in Eclipse

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

4.3.5 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.

4.3.5.1 Setup a Run Configuration for the Platform

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

1. In the PyDev Package Explorer view, open expand the env/bin folder and right click the VOLTTRON file -> select Run As -> Run Configurations...
2. Select the Main tab (Figure 72), in the Name field and enter a name (any name can be chosen, we have chosen VOLTTRON). Also, ensure that the Main Module field contains the same text as Figure 72.

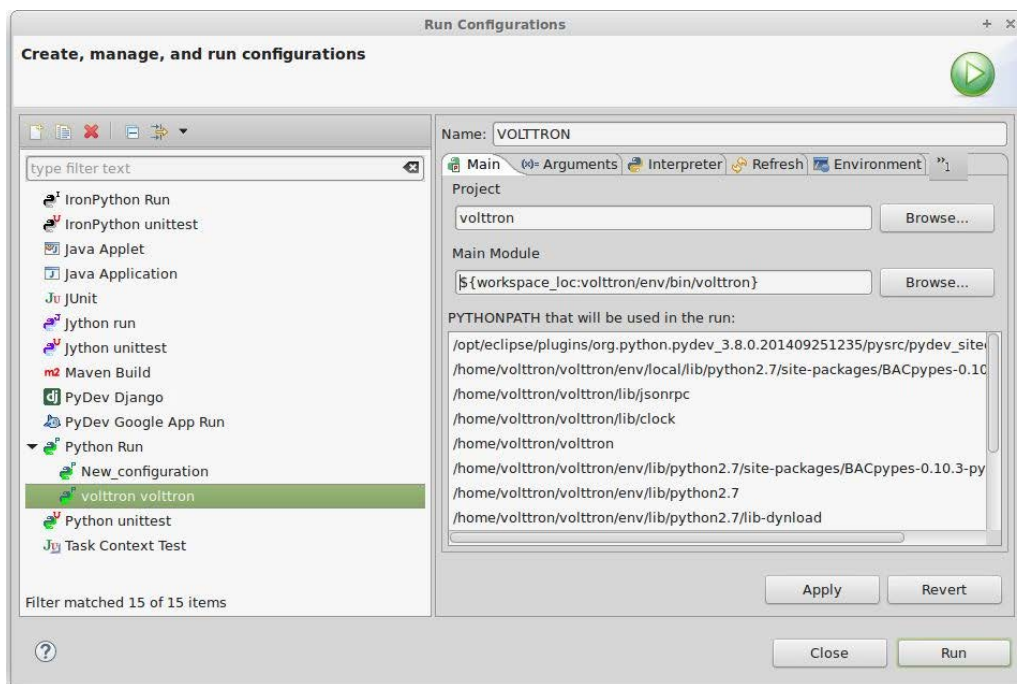


Figure 72: Running VOLTTRON Platform, Setting Up a Run Configuration

3. Select the Arguments tab (Figure 73), and select Default under the Working directory heading -> select Run at the bottom right corner of the window (Figure 73)

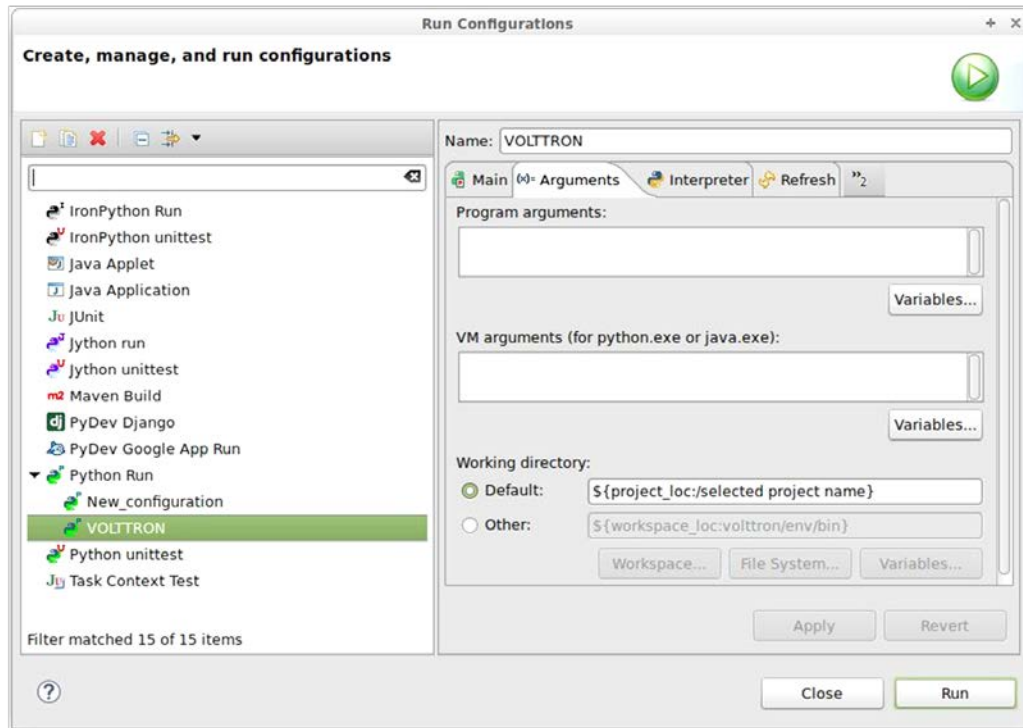


Figure 73: Running VOLTRON Platform, Setting Up a Run Configuration (Continued)

4. If the run is successful, the console should appear similar to Figure 74. 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.

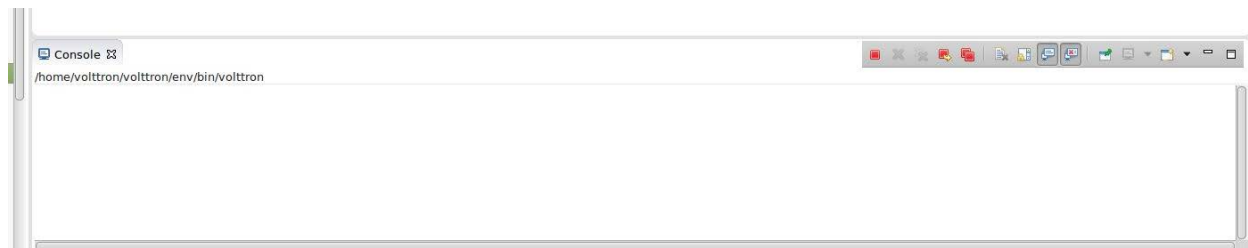


Figure 74: Running VOLTRON Platform, Console View on Successful Run

4.3.5.2 Configure a Run Configuration for the Listener Agent

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

1. In the PyDev Package Explorer view, open Agents -> ListenerAgent -> listener
2. Right-click on agent.py and select Run As -> Python Run (this will create a run configuration but fail)
3. Right-click on agent.py and select Run As -> Run Configurations
4. Click on the Argument tab

5. Change Working directory to Default
6. In the Environment tab, select New -> add the following environment variables (bulleted list below), as shown in Figure 75:

- AGENT_CONFIG = /home/<USER>/volttron/Agents/ListenerAgent/config
- AGENT_PUB_ADDR = ipc:///home/<USER>/.volttron/run/publish
- AGENT_SUB_ADDR = ipc:///home/<USER>/.volttron/run/subscribe

AGENT_CONFIG is the absolute path the agent's configuration file. AGENT_PUB_ADDR and AGENT_SUB_ADDR inform the platform where to publish and subscribe to messages. The AGENT_CONFIG variable will be unique for each agent and/or desired configuration. The AGENT_PUB_ADDR and AGENT_SUB_ADDR will be the same for all agents running within a single instance of VOLTTRON.

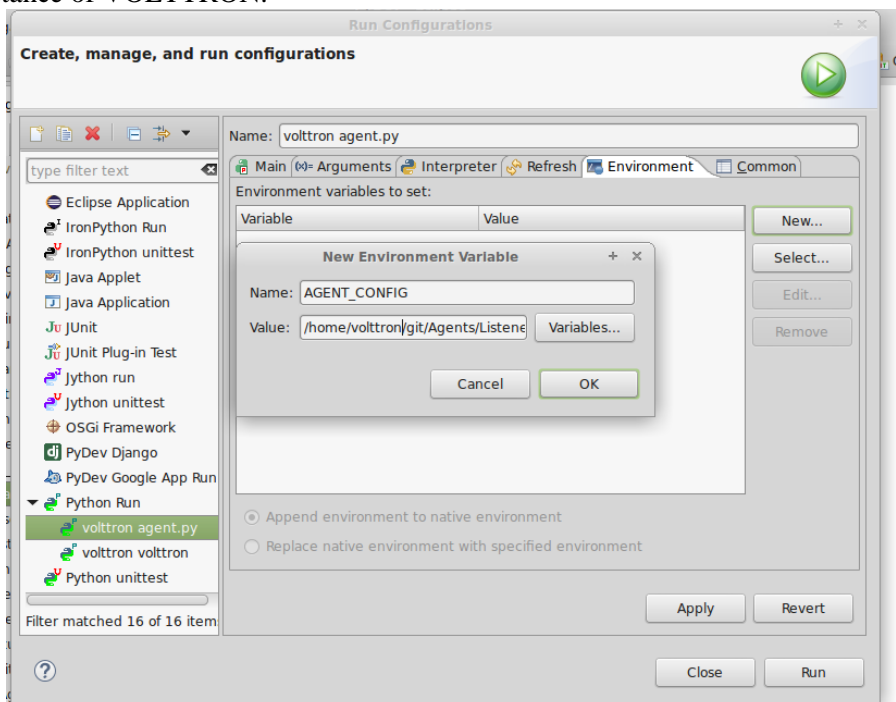


Figure 75: Running the Listener Agent, Setting Up a Run Configuration

7. Click Run, this launches the agent (Figure 76)

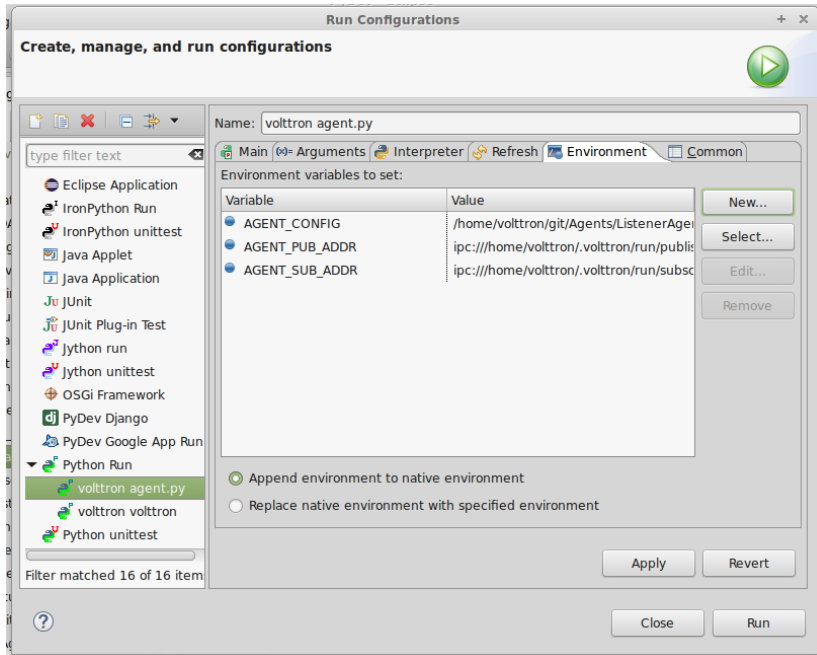


Figure 76: Configuring the Listener Agent, Setting Up a Run Configuration (Continued)

You should see the agent start to publish and receive its own heartbeat message (Figure 77).

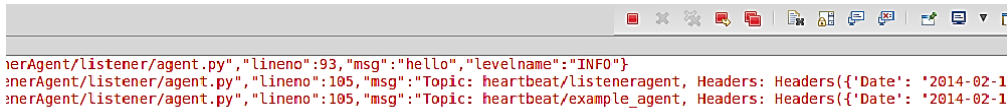


Figure 77: Listener Agent Output on Eclipse Console

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.

4.3.6 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.

4.3.6.1 Agent Folder Setup

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

1. In the Agents 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 78)

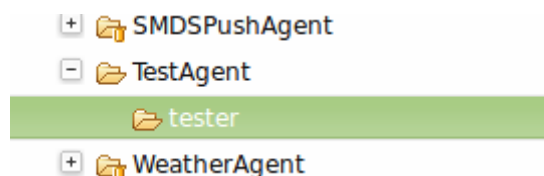


Figure 78: Creating an Agent Test Folder

4.3.6.2 *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 agent.py
- Create a class called TestAgent
- Import the packages and classes needed:

```
import sys
from volttron.platform.agent import BaseAgent, PublishMixin, periodic
from volttron.platform.agent import utils, matching
from volttron.platform.messaging import headers as headers_mod
```

This agent will inherit features from the BaseAgent extending the agent's default functionality. To enable an agent to publish on the message bus PublishMixin should be used as the first class definition. The class definition for the TestAgent will be configured as shown below.

```
class TestAgent(PublishMixin, BaseAgent):
```

The BaseAgent has several methods that could be overwritten for application-specific actions (init, setup, etc.). For purposes of this demonstration, use default behavior.

4.3.6.3 *Setting up a Subscription*

We will set our agent up to listen to heartbeat messages (published by Listener agent). Using the matching package, we declare we want to match all topics that start with "heartbeat/listeneragent". This will give us all heartbeat messages from all Listener agents but no others.

```
@matching.match_start('heartbeat/listeneragent')
def on_heartbeat_topic(self, topic, headers, message, match):
    print('TestAgent got\nTopic: {topic}, {headers}, '
          'Message: {message}').format(topic=topic,
                                       headers=headers,
                                       message=message)
```

4.3.6.4 *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.default_main(TestAgent,
                           description='Test Agent',
                           argv=argv)
    except Exception as e:
        print e
        _log.exception('unhandled exception')
```

```

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

```

4.3.6.5 *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.

4.3.6.6 *Packaging Configuration*

In the TestAgent folder, create a file called “setup.py” (or copy the setup.py in Listener agent). 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 + '.agent:main',
        ]
    }
)

```

4.3.6.7 *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 information for the agent.

```

{
    "agentid": "Test1",
    "message": "hello"
}

```

4.3.6.8 *Testing the Agent*

From a terminal, enter the following commands

1. Package the agent:
volttron-pkg package Agents/TestAgent
2. Set the configuration file:
volttron-pkg configure /tmp/volttron_wheels/testeragent-0.1-py2-none-any.whl Agents/TestAgent/config

3. Install agent into platform (with the platform running):
volttron-ctl install /tmp/volttron_wheels/testeragent-0.1-py2-none-any.whl
4. Start the agent:
volttron-ctl start --name testeragent-0.1
5. Verify that agent is running (Figure 79):
volttron-ctl status
tail volttron.log

```
2014-02-11 13:21:57,462 (testagent.launch.json 7832) <stdout> INFO: TestAgent go
t
2014-02-11 13:21:57,477 (testagent.launch.json 7832) <stdout> INFO: Topic: heart
beat/listeneragent, Headers({'Date': '2014-02-11 21:21:57.446051Z', 'AgentID': '
listener1', 'Content-Type': 'text/plain'}), Message: ['2014-02-11 21:21:57.44605
1Z']
```

Figure 79: TestAgent Output in “volttron.log”

4.3.6.9 *Reloading the Agent*

If changes are made to the agent’s code or configuration file after the agent is launched, it is necessary to stop and reload the agent. In a terminal, enter the following commands:

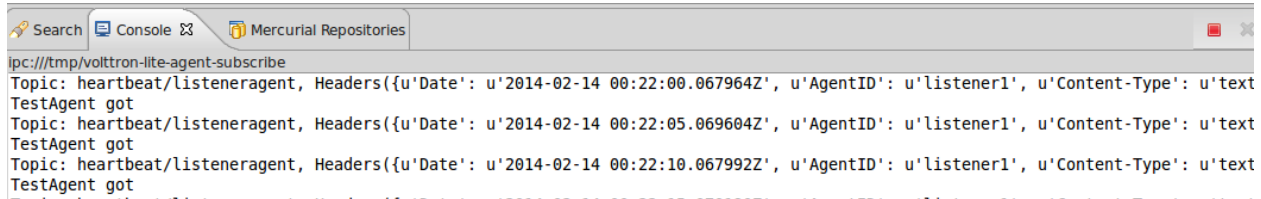
1. Activate VOLTTRON if it is not already activated:
. env/bin/activate
 Note there is a space between the “.” and “env.”
1. Stop the agent:
volttron-ctl stop --name testeragent-0.1
2. Remove the agent:
volttron-ctl remove --name testeragent-0.1
3. Package the agent:
volttron-pkg package Agents/TestAgent
4. Set the configuration file:
volttron-pkg configure /tmp/volttron_wheels/testeragent-0.1-py2-none-any.whl
Agents/TestAgent/config
5. Install agent into platform (with the platform running):
volttron-ctl install /tmp/volttron_wheels/testeragent-0.1-py2-none-any.whl
6. Start the agent:
volttron-ctl start --name testeragent-0.1

4.3.6.10 *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 4.3.5).

- Launch the platform (Section 4.3.5.1)

- Launch the TestAgent:
 - (follow steps outlined in Section 4.3.5.2 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 80)



```
ipc:///tmp/volttron-lite-agent-subscribe
Topic: heartbeat/listeneragent, Headers({'u'Date': u'2014-02-14 00:22:00.067964Z', u'AgentID': u'listener1', u'Content-Type': u'text'})
TestAgent got
Topic: heartbeat/listeneragent, Headers({'u'Date': u'2014-02-14 00:22:05.069604Z', u'AgentID': u'listener1', u'Content-Type': u'text'})
TestAgent got
Topic: heartbeat/listeneragent, Headers({'u'Date': u'2014-02-14 00:22:10.067992Z', u'AgentID': u'listener1', u'Content-Type': u'text'})
TestAgent got
```

Figure 80: Console Output for TestAgent

4.3.7 Adding Additional Features to the TestAgent

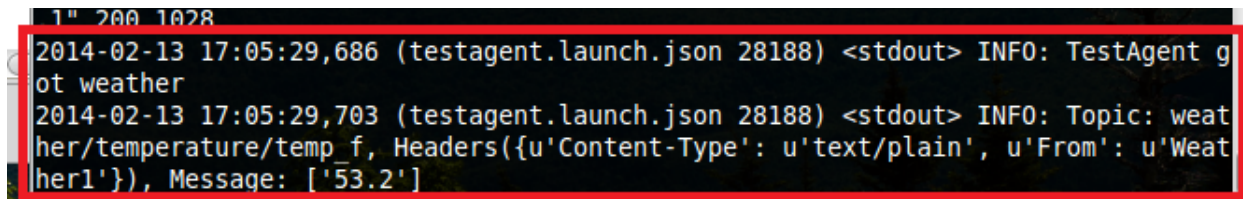
Additional code can be added to the TestAgent to utilize additional services in the platform. The following sections will show how to use the weather and device scheduling service within the TestAgent.

4.3.7.1 Subscribing to Weather Data

This agent can be modified to listen to weather data from the Weather agent by adding the following method. This will subscribe the agent to the temperature topic. For the full list of topics available, please see: <https://github.com/VOLTTRON/volttron/wiki/WeatherAgentTopics>

```
@matching.match_exact('weather/temperature/temp_f')
def on_weather(self, topic, headers, message, match):
    print (("TestAgent got weather\nTopic: {topic}, {headers}, "
           "Message: {message}").format(topic=topic,
                                         headers=headers,
                                         message=message))
```

The platform log file should appear similar to Figure 81.



```
1" 200 1028
2014-02-13 17:05:29,686 (testagent.launch.json 28188) <stdout> INFO: TestAgent g
ot weather
2014-02-13 17:05:29,703 (testagent.launch.json 28188) <stdout> INFO: Topic: weat
her/temperature/temp f, Headers({'u'Content-Type': u'text/plain', u'From': u'Weat
her1'}), Message: ['53.2']
```

Figure 81: TestAgent Output when Subscribing to Weather Topic

4.3.7.2 Utilizing the Scheduler Agent

The TestAgent can be modified to publish a schedule to the Actuator agent by reserving time on fake 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 2.11. Then, import the messaging package so the TestAgent can use the topics for the scheduler:

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

Then, create `__init__` and `setup` methods so it can pull the agent id from its config file. These methods will override the BaseAgent methods, so one should call them with “super” so the default actions still happen. Then call the `publish_schedule` command, which is detailed later.

```

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

def setup(self):
    self._agent_id = self.config['agentid']
    #Always call the base class setup()
    super(TestAgent, self).setup()
    self.publish_schedule()

```

This calls the following method, which pushes 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.publish_json(topics.ACTUATOR_SCHEDULE_REQUEST,
                      headers,
                      msg)

```

The agent can listen to the results of its request and get schedule announcements on the following topic:

```

@matching.match_start(topics.ACTUATOR_SCHEDULE_RESULT)
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))

```

4.3.7.3 Full TestAgent Code

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

```

import sys
from volttron.platform.agent import BaseAgent, PublishMixin
from volttron.platform.agent import utils, matching
from volttron.platform.messaging import headers as headers_mod
from volttron.platform.messaging import topics

class TestAgent(PublishMixin, BaseAgent):

```



```

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

def setup(self):
    self._agent_id = self.config['agentid']
    # Always call the base class setup()
    super(TestAgent, self).setup()
    self.publish_schedule()

@matching.match_start('heartbeat/listeneragent')
def on_heartbeat_topic(self, topic, headers, message, match):
    print('TestAgent got\nTopic: {topic}, {headers}, '
          'Message: {message}').format(topic=topic,
                                       headers=headers,
                                       message=message)

@matching.match_exact('weather/temperature/temp_f')
def on_weather(self, topic, headers, message, match):
    print(("TestAgent got weather\nTopic: {topic}, {headers}, "
          "Message: {message}").format(topic=topic,
                                       headers=headers,
                                       message=message))

@matching.match_start(topics.ACTUATOR_SCHEDULE_RESULT)
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 publish_schedule(self):
    headers = {
        'AgentID': self._agent_id,
        'type': 'NEW_SCHEDULE',
        'requesterID': self._agent_id, #Requesting agent
        'taskID': self._agent_id + "-TASK", #Unique task ID
        'priority': 'LOW', #Task priority
    }
    msg = [
        ["campus/building/device1", #First time slot.
         "2014-2-11 16:27:00",      #Start of time slot.
         "2014-2-11 16:29:00"],     #End of time slot.
        ["campus/building/device1", #Second time slot.
         "2014-2-11 16:36:00",      #Start of time slot.
         "2014-2-11 16:39:00"],     #End of time slot.
        ["campus/building/device2", #Third time slot.
         "2014-2-11 16:30:00",      #Start of time slot.
         "2014-2-11 16:32:00"]     #End of time slot.
        #etc...
    ]
    self.publish_json(topics.ACTUATOR_SCHEDULE_REQUEST, headers,
msg)

def main(argv=sys.argv):

```

```
'''Main method called by the eggsecutable.'''
utils.default_main(TestAgent,
                    description='Test Agent',
                    argv=argv)

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

5 New 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 5.2.4.

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 "Ping Pong" agent

5.1 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 volttron directory:

- Install additional VOLTTRON Restricted dependency:
sudo 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>

5.2 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 82 shows the structure of the agent signing feature:

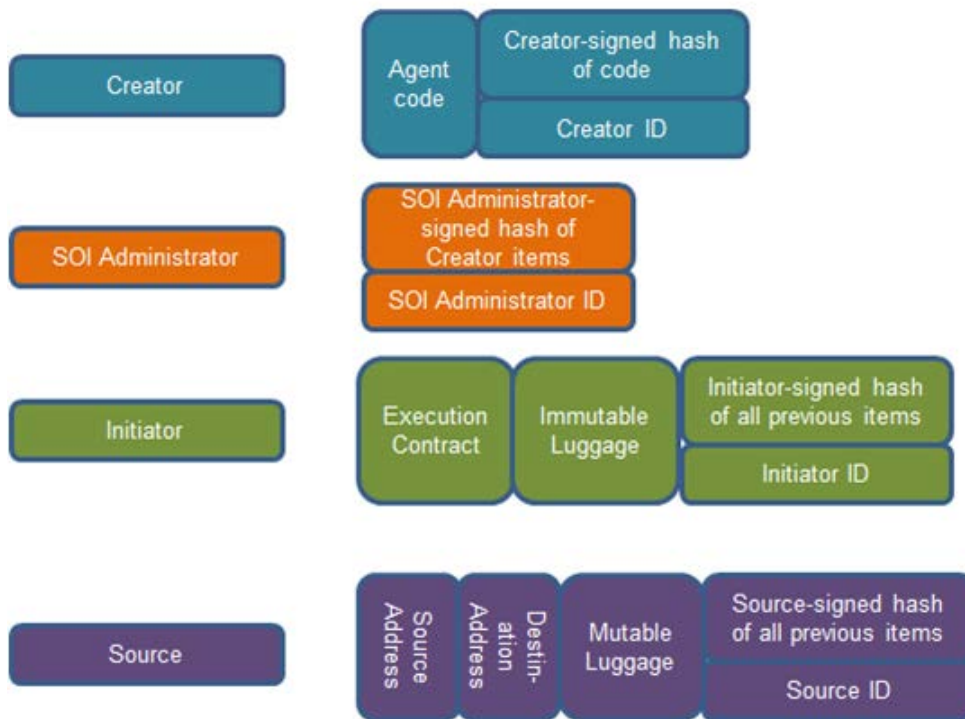


Figure 82: Structure for the Agent Signing Security Feature in VOLTTRON Restricted

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] [--verbooseness 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 in order 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, an 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.

5.2.1 Creating Required Security Certificates

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

1. Activate the VOLTTRON platform:
 - **.env/bin/activate** (note the space after the period)
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)

5.2.2 Enabling Agent Mobility Feature

To create the required keys to (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 2.7 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 known 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

5.2.3 Enabling Resource Monitoring

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

1. Run cgroup setup script:
sudo volttron/scripts/cgroup_setup.sh
2. Create cgroups:
sudo env/bin/volttron-ctl create-cgroups -u \$USER

5.2.4 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 5.2.2. 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.

5.2.4.1 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.

5.2.4.1.1 Soft Requirements

Soft 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 ($N \geq 1$) in MIPS (millions of instructions per second) or a floating-point percentage ($0.0 < N < 1.0$) of the total available bogo-MIPS 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.

5.2.4.1.2 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.

5.2.4.1.3 Signing and Launching Agents with VOLTTRON Restricted Enabled

If VOLTTRON Restricted is installed and the security features are enabled (Section 5.2.4.1.4), 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 2.8 of this document). From a terminal, in the `volttron` directory, enter the following commands:

1. Package the agent:
`volttron-pkg package Agents/ListenerAgent`
2. Sign the agent as creator (`resource_contract` is a text file containing the hardware and software requirements for the agent, see Section 5.2.4):
**`volttron-pkg sign --creator --contract resource_contract
/tmp/volttron_wheels/listeneragent-0.1-py2-none-any.whl`**
3. Sign the agent as soi:

volttron-pkg sign --soi /tmp/volttron_wheels/listeneragent-0.1-py2-none-any.whl

4. Sign the agent as initiator:

**volttron-pkg sign --initiator --config-file Agents/ListenerAgent/config
/tmp/volttron_wheels/listeneragent-0.1-py2-none-any.whl**

5. Set the configuration file:

**volttron-pkg configure /tmp/volttron_wheels/listeneragent-0.1-py2-none-any.whl
Agents/ListenerAgent/config**

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

volttron-ctl install /tmp/volttron_wheels/listeneragent-0.1-py2-none-any.whl

- Upon successful completion of this command the terminal output will inform one on 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-3c1ff8a81740listeneragent-0.1**

7. Start the agent:

volttron-ctl start --name listeneragent-0.1

- Agent commands can also use the uuid as an identifier (i.e., **volttron-ctl start --uuid a9d67c55-7f58-4591-80af-3c1ff8a81740**). This is helpful when managing multiple instances of the same agent.

8. Verify that agent is running:

**volttron-ctl status
tail volttron.log**

5.2.4.1.4 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 2.7):

- 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