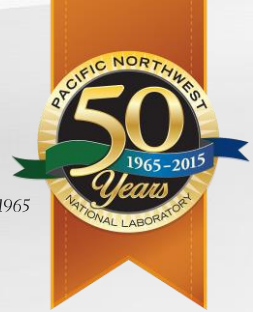
Pacific Northwest NATIONAL LABORATORY

Proudly Operated by Battelle Since 1965



VOLTTRON Primer

BRANDON CARPENTER
JEREME HAACK

Pacific Northwest National Laboratory

Software Framework for Transactive Energy: VOLTTRON™, VTARI, Arlington, VA

VOLTTRON Team

Pacific Northwest NATIONAL LABORATORY
Proudly Operated by Battelle Since 1965

Software Development Team

- Bora Akyol
- Jereme Haack
- Brandon Carpenter
- Kyle Monson
- Craig Allwardt
- Poorva Sharma
- Tim Kang
- Robert Lutes
- Casey Neubauer
- Dan Johnson

Application Development Team

- Srinivas Katipamula
- Robert Lutes
- Wooyun Kim
- Rick Pratt
- Carl Miller
- Weimin Wang
- Siddartha Goyal
- Michael Brambley
- Lucy Huang
- Chad Corbin
- He Hao

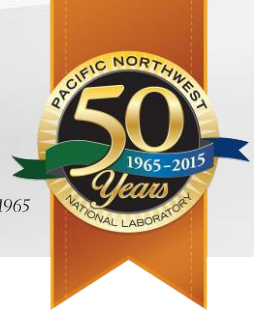
Topics Covered

Pacific Northwest NATIONAL LABORATORY

Proudly Operated by Battelle Since 1965

- Basics
- Historian
- Drivers
- VOLTTRON Management Central
- 2.0 to 3.0 Agent

Installing the Platform

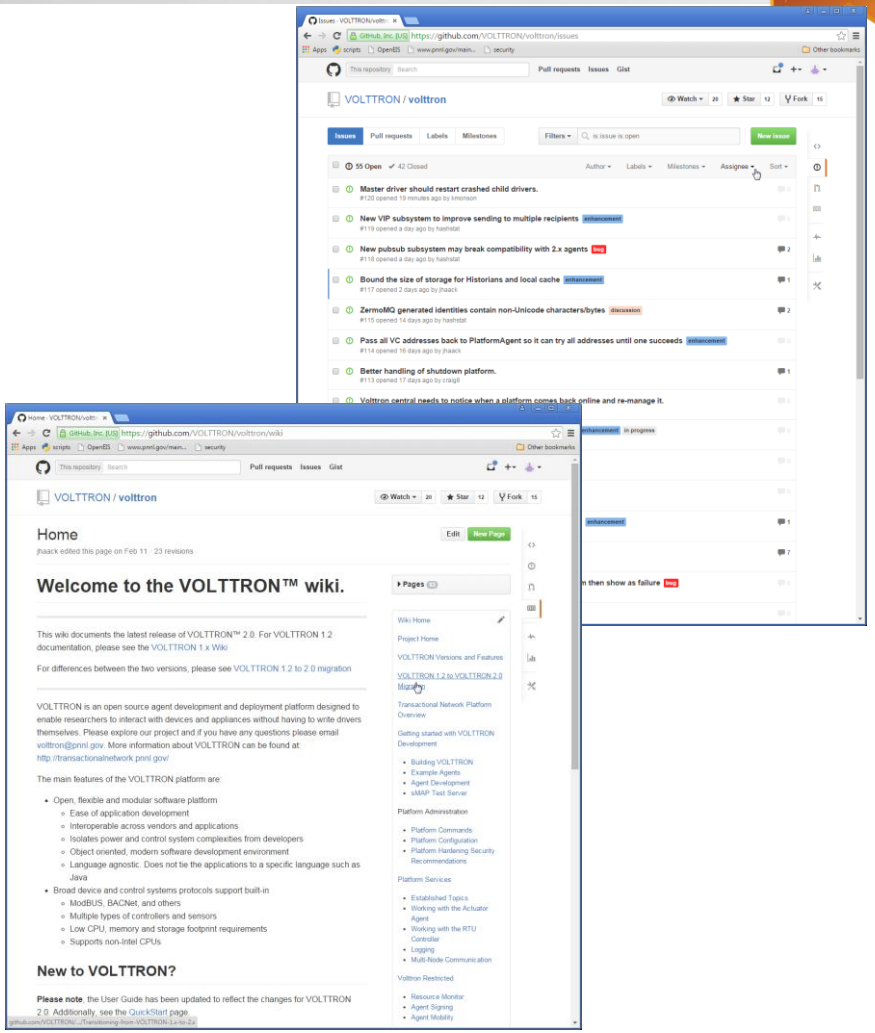


- Clone project from Git: https://github.com/VOLTTRON/volttron.git
- Install pre-reqs:

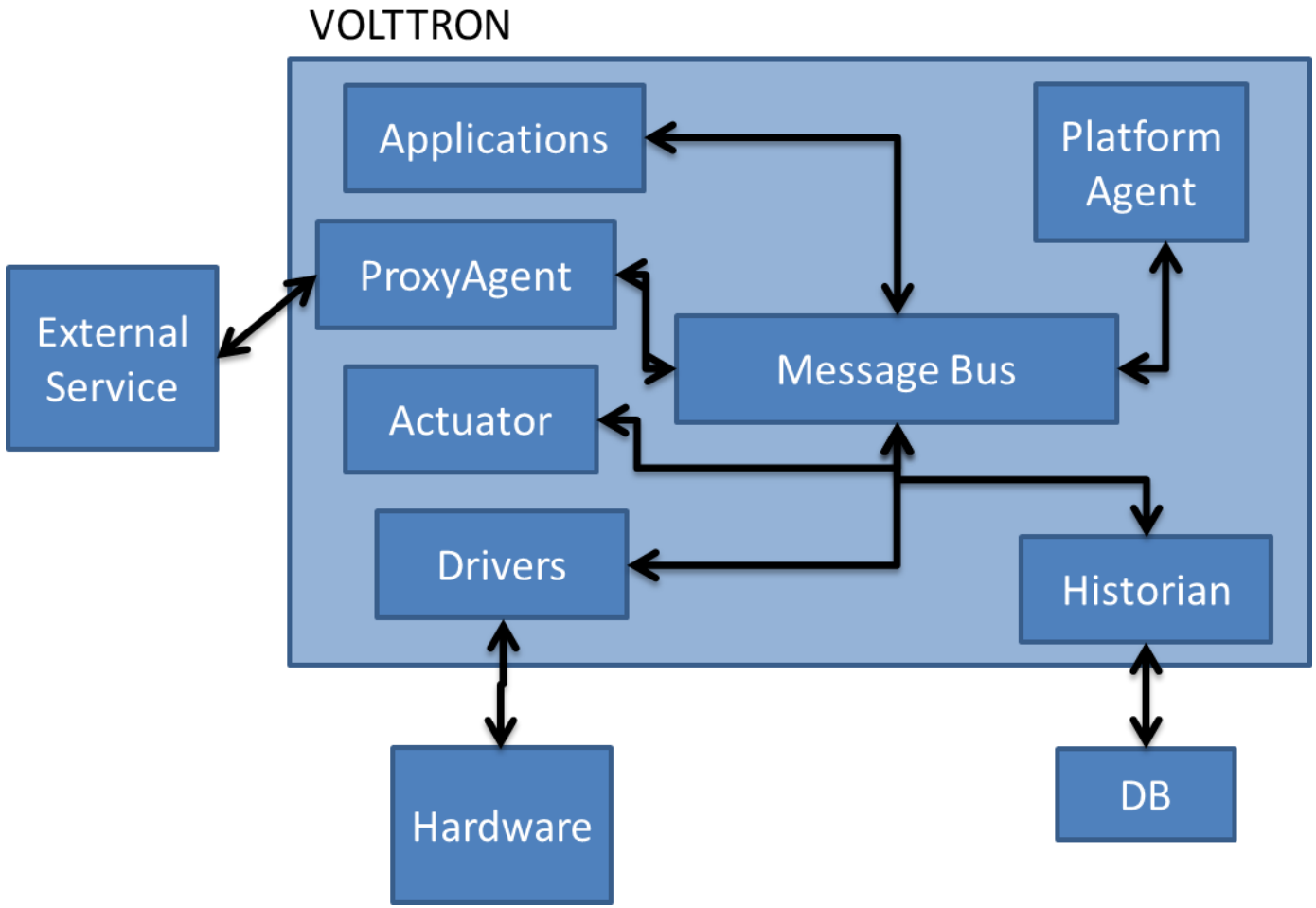
- Run: python bootstrap.py
- Activate: . env/bin/activate
- Build and Install Listener: volttron/scripts/pack_and_install Agents/ListenerAgent Agents/ListenerAgent listener
- Start Agent: volttron-ctl start –tag listener
- See it running: volttron-ctl status

GitHub

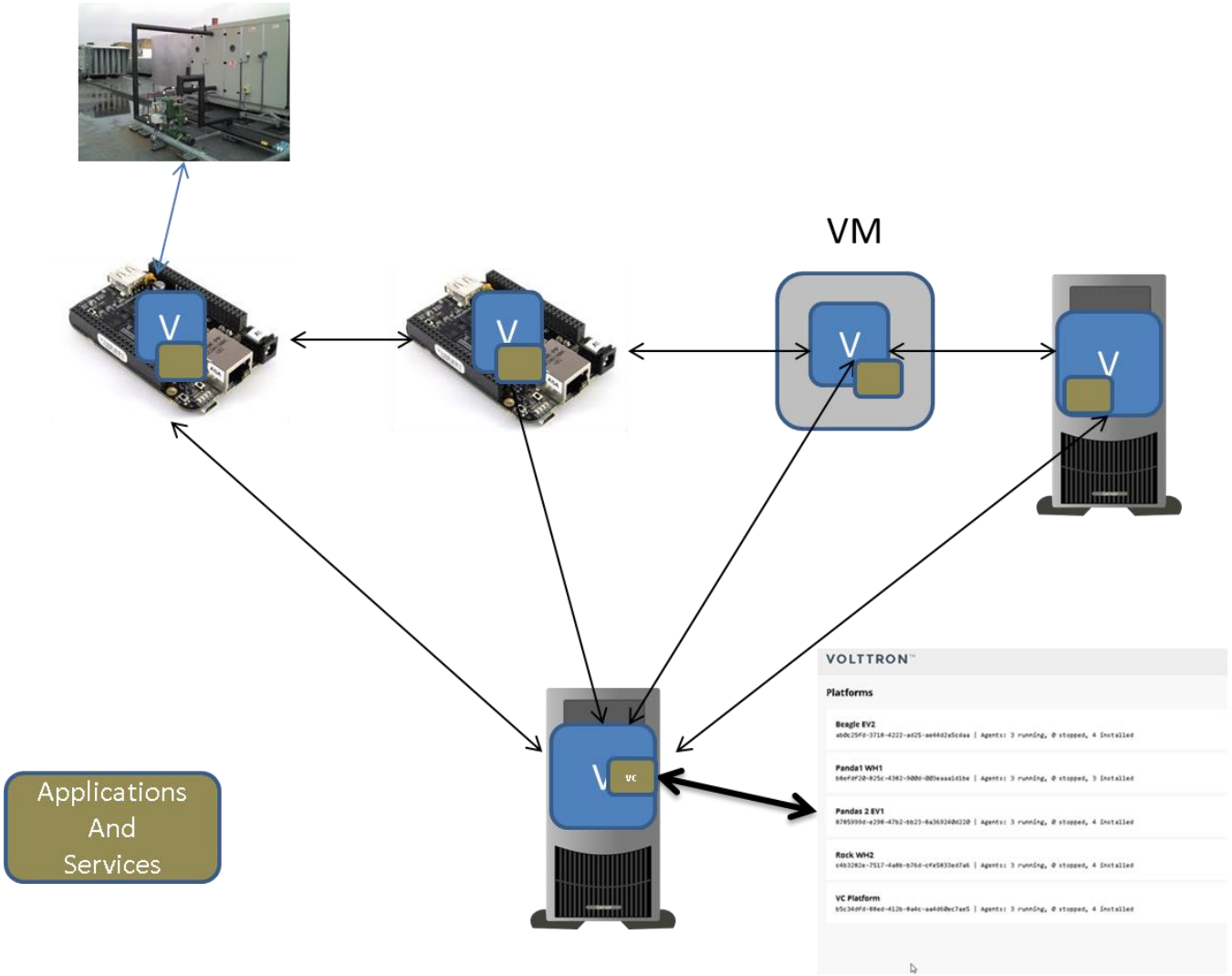
- Code repository
- Wiki
- Issue tracking
  - Bugs
  - Enhancements
  - Used for development planning



VOLTTRON Overview

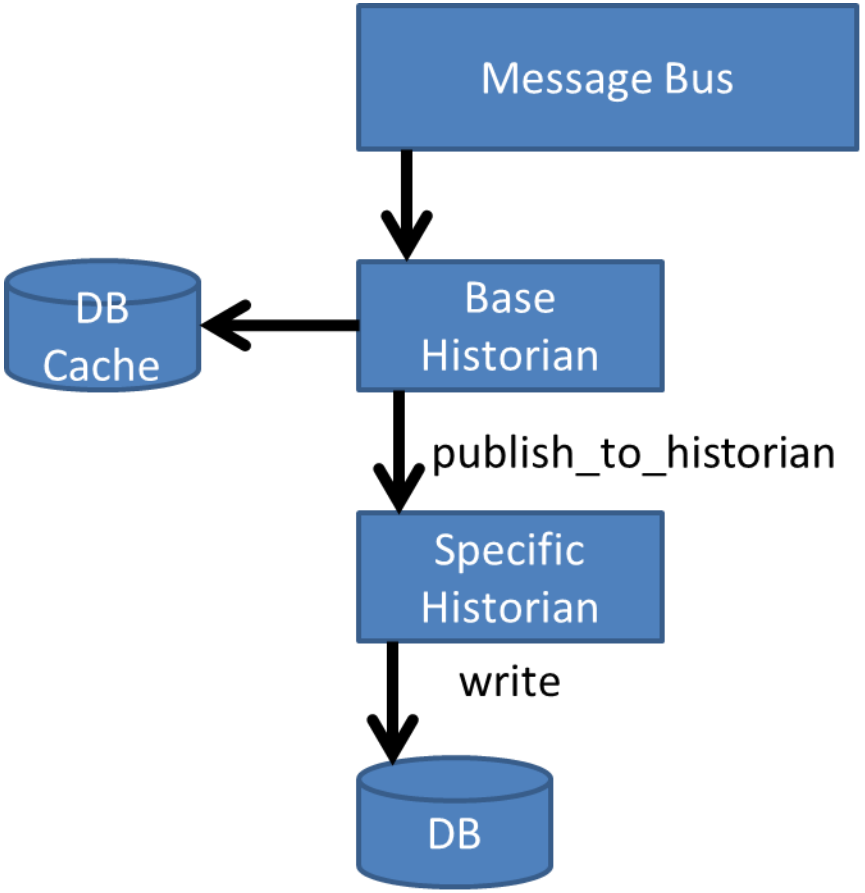


Deployment Components



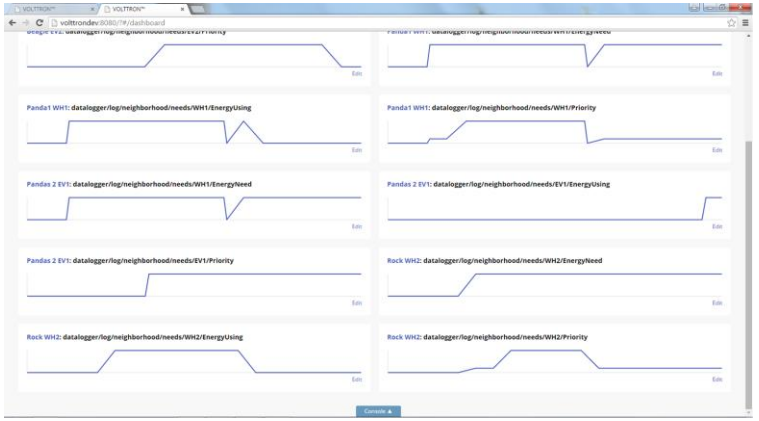
Working with the Historian

- base_historian.py
  - handles getting device and agent data from the message bus
  - Writes data to local cache until successful write
- Specific implementations should extend this class and only need to implement specifics functions
  - historian_setup
  - publish_to_historian: store data in db, external service, file, etc.
  - query_historian



VOLTTRON Central: Managing the Platform

- VOLTTRON Central Manager
  - Agent presenting web interface
  - Platform registry for deployment
  - Provides graphs of points on other platforms
- Platform Agent
  - Point of contact for addressing platform
  - Provides access to other agents on platform
- PlatformHistorian
  - Records data from Message Bus
  - Exposes query interface for Central Manager Agent



Base Agent Definition

Pacific Northwest NATIONAL LABORATORY
Proudly Operated by Battelle Since 1965

__init__.py (~/devel/volttron/volttron/platform/vip/agent) – GVIM

```
+-- 56 lines: -*- coding: utf-8 -*- ----------------------------------------

from __future__ import absolute_import

from .core import *
from .errors import *
from .decorators import *
from .subsystems import *


class Agent(object):
    class Subsystems(object):
        def __init__(self, owner, core):
            self.ping = Ping(core)
            self.rpc = RPC(core, owner)
            self.hello = Hello(core)
            self.pubsub = PubSub(core, self.rpc, owner)
            self.channel = Channel(core)

    def __init__(self, identity=None, address=None, context=None):
        self.core = Core(
            self, identity=identity, address=address, context=context)
        self.vip = Agent.Subsystems(self, self.core)
~
:0                                                    1,1           All
```

Agent Core

- Main event loop handler and VIP message dispatcher
- Namespace: *Agent.core*
- Methods:
  - *register(name, handler, error_handler)*
    - Register a subsystem handler
  - *run(running_event=None)*
    - Connects VIP socket and starts processing of VIP messages
  - *stop(timeout=None)*
    - Stop the agent (can be called from any context)
  - *send(func, *args, **kwargs)* and *send_async(func, *args, **kwargs)*
    - Send functions from any thread to execute
  - *spawn(func, *args, **kwargs)* and *spawn_in_thread(func, *args, **kwargs)*
    - Spawn function in new greenlet or thread

Agent Core (continued)

- Decorators:
  - *periodic(period, args=None, kwargs=None, wait=0)*
    - Execute a method on a regular interval
  - *schedule(deadline, *args, **kwargs)*
    - Execute a method at a specific time
  - receiver(signal)
    - Register a method as a callback for the named signal
- Signals:
  - *onsetup* – used for instantiation and configuration
    - VIP messaging is not running
    - All receivers run serially
  - *onstart* – used to spawn tasks as VIP loop starts
  - *onstop* – signaled just before VIP loop stops
  - *onfinish* – signaled after VIP loop stops
    - Used for teardown and cleanup

PubSub Subsystem

- Namespace: *Agent.vip.pubsub*
- Methods:
  - *add_bus(name) / remove_bus(name)*
    - Add or remove a pub/sub bus
  - *subscribe(peer, prefix, callback, bus='')*
    - Subscribe to *bus* on *peer* for topics starting with *prefix*.
    - *callback(peer, sender, bus, topic, headers, message)*
  - *unsubscribe(peer, prefix, callback, bus='')*
    - Unsubscribe from *bus* on *peer*
    - Wildcard one or both of *prefix* or *callback* by passing None
  - *publish(peer, topic, headers=None, message=None, bus='')*
    - Publish *headers* and *message* to *topic* on *bus* on *peer*
    - *peer* may be None to publish to self
- Decorators:
  - *PubSub.subscribe(peer, prefix, bus='')*
    - Register method ass callback for subscription

RPC Subsystem

- Namespace: *Agent.vip.rpc*
- Methods:
  - *call(peer, method, *args, **kwargs)*
    - Call remote method exported by peer
    - Returns AsyncResult object
  - *notify(peer, method, *args, **kwargs)*
    - Send notification to peer via exported method
- Decorators:
  - *export(name=None)*
    - Make method remotely callable

Porting Listener Agent from 2.x to 3.0

Pacific Northwest NATIONAL LABORATORY
Proudly Operated by Battelle Since 1965

PACIFIC NORTHWEST 50 1965-2015 Years NATIONAL LABORATORY

```
from datetime import datetime
import logging
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

import settings


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


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

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

    def setup(self):
        # Demonstrate accessing a value from the config file
        _log.info(self.config['message'])
        self._agent_id = self.config['agentid']
        # Always call the base class setup()
        super(ListenerAgent, self).setup()

    @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))

    # 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)


def main(argv=sys.argv):
    '''Main method called by the eggsecutable.'''
```

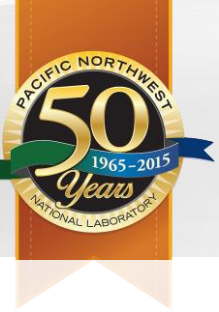
```
from datetime import datetime
import logging
import sys

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

import settings


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


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

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

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

    @PubSub.subscribe('pubsub', '')
    def on_match(self, peer, sender, bus,  topic, headers, message):
        '''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))

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

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


def main(argv=sys.argv):
    '''Main method called by the eggsecutable.'''
```

Porting Listener Agent (Continued)

- Update imports to use VIP modules

```
-from volttron.platform.agent import BaseAgent, PublishMixin, periodic
-from volttron.platform.agent import utils, matching
+from volttron.platform.vip.agent import Agent, Core, PubSub
+from volttron.platform.agent import utils
```

Porting Listener Agent (Continued)



- Modify class definition to use VIP base agent

```
-class ListenerAgent(PublishMixin, BaseAgent):
+class ListenerAgent(Agent):
```

Porting Listener Agent (Continued)

- Use core signals instead of overriding event methods

```
-     def setup(self):
+     @Core.receiver('onsetup')
+     def setup(self, sender, **kwargs):
```

```
-         # Always call the base class setup()
-         super(ListenerAgent, self).setup()
```

Porting Listener Agent (Continued)

- Use PubSub subscribe decorator to subscribe to topics

```
-     @matching.match_all
-     def on_match(self, topic, headers, message, match):
+     @PubSub.subscribe('pubsub', '')
+     def on_match(self, peer, sender, bus,  topic, headers, message):
```

Porting Listener Agent (Continued)

- Periodic decorator is now namespaced

```
-    @periodic(settings.HEARTBEAT_PERIOD)
+    @Core.periodic(settings.HEARTBEAT_PERIOD)
```

Porting Listener Agent (Continued)

- VIP subsystem methods are now namespaced in agent class

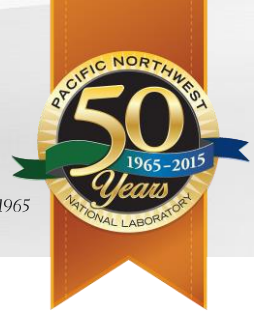
```
-  self.publish('heartbeat/listeneragent', headers, now)
+          self.vip.pubsub.publish(
+              'pubsub', 'heartbeat/listeneragent', headers, now)
```

Porting Listener Agent (Continued)


```
from datetime import datetime
import logging
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

import settings


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


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

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

    def setup(self):
        # Demonstrate accessing a value from the config file
        _log.info(self.config['message'])
        self._agent_id = self.config['agentid']
        # Always call the base class setup()
        super(ListenerAgent, self).setup()

    @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))

    # 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)


def main(argv=sys.argv):
    '''Main method called by the eggsecutable.'''
```

```
from datetime import datetime
import logging
import sys

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

import settings


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


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

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

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

    @PubSub.subscribe('pubsub', '')
    def on_match(self, peer, sender, bus,  topic, headers, message):
        '''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))

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

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


def main(argv=sys.argv):
    '''Main method called by the eggsecutable.'''
```

gevent for Cooperative Multitasking



- According to gevent.org:

gevent is a coroutine-based Python networking library that uses greenlet to provide a high-level synchronous API on top of the libev event loop.

Features include:

- Fast event loop based on libev (epoll on Linux, kqueue on FreeBSD).
- Lightweight execution units based on greenlet.
- API that re-uses concepts from the Python standard library (for example there are Events and Queues).
- Cooperative sockets with SSL support
- DNS queries performed through threadpool or c-ares.
- Monkey patching utility to get 3rd party modules to become cooperative

Tips for using gevent

- gevent is cooperative
  - Greenlet's own the thread until explicitly relinquished or blocking operation
  - Must use gevent-aware routines when blocking
  - Locking is not necessary
  - Use *gevent.sleep(0)* to yield thread
- Use caution when using gevent across threads
  - Agent core has methods to assist
    - *send()*, *send_async()*, and *spawn_in_thread()*
  - This use case is rare
  - Can monkey-patch threading module
- When reading files use gevent.fileobject.FileObject proxy
- Use zmq.green in place of zmq
  - *from zmq import green as zmq*
- Provides socket, ssl, select, etc. modules

VOLTTRON Resources

- GitHub
  - https://github.com/VOLTTRON/volttron.git
  - https://github.com/VOLTTRON/volttron/wiki
- Email: volttron@pnnl.gov
- Bi-weekly office hours, email to be added