BACpypes Documentation

Release 1.0

Joel Bender

Feb 17, 2018

Contents

BACpypes library for building BACnet applications using Python. Installation is easy, just:

```
$ sudo easy_install bacpypes
or
$ sudo pip install bacpypes
```

You will be installing the latest released version from PyPI (the Python Packages Index), located at pypi.python.org

---

Note: You can also check out the latest version from GitHub:

```
$ git clone https://github.com/JoelBender/bacpypes.git
```

And then use the setup utility to install it:

```
$ cd bacpypes
$ python setup.py install
```

Tip: If you would like to participate in its development, please join:

- the developers mailing list,
- the chat room on Gitter, and
- add Google+ to your circles to have release notifications show up in your stream.

Welcome aboard!

CHAPTER 1

Getting Started

This section is a walk through of the process of installing the library, downloading the sample code and communicating with a test device.

1.1 Getting Started

Ah, so you are interested in getting started with BACnet and Python. Welcome to BACpypes, I hope you enjoy your journey. This tutorial starts with just enough of the basics of BACnet to get a workstation communicating with another device. We will cover installing the library, downloading and configuring the samples applications.

1.1.1 Basic Assumptions

I will assume you are a software developer and it is your job to communicate with a device from another company that uses BACnet. Your employer has given you a test device and purchased a copy of the BACnet standard. I will need...

- a development workstation running some flavor of Linux or Windows, complete with the latest version of Python (2.7 or >3.4) and setup tools.
- a small Ethernet hub into which you can plug both your workstation and your mysterious BACnet device, so you won't be distracted by lots of other network traffic.
- a BACnetIP/BACnet-MSTP Router if your mysterious device is an MSTP device (BACpypes is actually BACnet/IP software)
- if you are running on Windows, installing Python may be a challenge. Some Python packages make your life easier by including the core Python plus many other data processing toolkits, so have a look at Continuum Analytics Anaconda or Enthought Canopy.

Before getting this test environment set up and while you are still connected to the internet, install the BACpypes library:

```
$ sudo easy_install bacpypes
```

or:

```
$ sudo pip install bacpypes
```

And while you are at it, get a copy of the BACpypes project from GitHub. It contains the library source code, sample code, and this documentation. Install the Git software from here, then make a local copy of the repository by cloning it:

```
$ git clone https://github.com/JoelBender/bacpypes.git
```

No protocol analysis workbench would be complete without an installed copy of Wireshark:

```
$ sudo apt-get install wireshark
```

or if you use Windows, download it here.

Caution: Don't forget to **turn off your firewall** before beginning to play with BACpypes! It will prevent you from hours of researches when your code won't work as it should!

1.1.2 Configuring the Workstation

The mystery BACnet device you have is going to come with some configuration information by default and sometimes it is easier to set up the test environment with my set of assumptions than come up with a fresh set from scratch.

IP Address The device will probably come with an IP address, let's assume that it is 192.168.0.10, subnet mask 255.255.0.0, gateway address 192.168.0.1. You are going to be joining the same network, so pick 192.168.0.11 for your workstation address and use the same subnet mask 255.255.0.0.

If working with MSTP devices, base your workstation address on the address of the BACnetIP Router.

Network Number If working with a BACnetIP router and an MSTP device, you will need to know the network number configured inside the router. Every BACnet network **must** have a unique numeric identifier. You will often see the magical number **2000** but you can choose anything between 1 to 0xFFFE.

Device Identifier Every BACnet device on a BACnet network **must** have a unique numeric identifier. This number is a 22-bit unsigned non-zero value. It is critical this identifier be unique. Most large customers will have someone or some group responsible for maintaining device identifiers across the site. Keep track of the device identifier for the test device. Let's assume that this device is **1000** and you are going to pick **1001** for your workstation.

Device Name Every BACnet device on a BACnet network should also have a unique name, which is a character string. There is nothing on a BACnet network that enforces this uniqueness, but it is a real headache for integrators when it isn't followed. You will need to pick a name for your workstation. My collegues and I use star names, so in the sample configuration files you will see the name "Betelgeuse". An actual customer's site will use a more formal (but less fun) naming convention.

There are a few more configuration values that you will need, but you won't need to change the values in the sample configuration file until you get deeper into the protocol.

Maximum APDU Length Accepted BACnet works on lots of different types of networks, from high speed Ethernet to "slower" and "cheaper" ARCNET or MS/TP (a serial bus protocol used for a field bus defined by BACnet). For devices to exchange messages they need to know the maximum size message the other device can handle.

Segmentation Supported A vast majority of BACnet communications traffic fits in one message, but there are times when larger messages are convenient and more efficient. Segmentation allows larger messages to be broken up into segments and spliced back together. It is not unusual for "low power" field devices to not support segmentation.

There are other configuration parameters in the INI file that are also used by other applications, just leave them alone for now.

Updating the INI File

Now that you know what these values are going to be, you can configure the BACnet portion of your workstation. Change into the samples directory that you checked out earlier, make a copy of the sample configuration file, and edit it for your site:

```
$ cd bacpypes/samples
$ cp BACpypes~.ini BACpypes.ini
```

Tip: The sample applications are going to look for this file. You can direct the applications to use other INI files on the command line, so it is simple to keep multiple configurations.

At some point you will probably running both "client" and "server" applications on your workstation, so you will want separate configuration files for them. Keep in mind that BACnet devices communicate as peers, so it is not unusual for an application to act as both a client and a server at the same time.

A typical BACpypes.ini file contains:

```
[BACpypes]
objectName: Betelgeuse
address: 192.168.1.2/24
objectIdentifier: 599
maxApduLengthAccepted: 1024
segmentationSupported: segmentedBoth
maxSegmentsAccepted: 1024
vendorIdentifier: 15
foreignPort: 0
foreignBBMD: 128.253.109.254
foreignTTL: 30
```

1.1.3 UDP Communications Issues

BACnet devices communicate using UDP rather than TCP. This is so devices do not need to implement a full IP stack (although many of them do because they support multiple protocols, including having embedded web servers).

There are two types of UDP messages; *unicast* which is a message from one specific IP address (and port) to another device's IP address (and port); and *broadcast* messages which are sent by one device and received and processed by all other devices that are listening on that port. BACnet uses both types of messages and your workstation will need to receive both types.

The BACpypes.ini file has an *address* parameter which is an IP address in CIDR notation and can be followed by a port number. For example, **192.168.0.11/16** specifies both the IP address and the number of bits in the network portion, which in turn implies a subnet mask, in this case **255.255.0.0**. Unicast messages will be sent to the IP address, and broadcast messages will be sent to the broadcast address **192.168.255.255** which is the network portion of the address with all 1's in the host portion. In this example, the default port 47808 (0xBAC0) is used but you could provide and different one, **192.168.0.11:47809/16**.

To receive both unicast and broadcast addresses, BACpypes opens two sockets, one for unicast traffic and one that only listens for broadcast messages. The operating system will typically not allow two applications to open the same socket at the same time so to run two BACnet applciations at the same time they need to be configured with different ports.

Note: The BACnet protocol has been assigned port 47808 (hex 0xBAC0) by by the Internet Assigned Numbers Authority, and sequentially higher numbers are used in many applications (i.e. 47809, 47810,...). There are some BACnet routing and networking issues related to using these higher unoffical ports, but that is a topic for another tutorial.

1.1.4 Starting An Application

The simplest BACpypes sample application is the **WhoIsIAm.py** application. It sends out Who-Is and I-Am messages and displays the results it receives. What are these things?

As mentioned before, BACnet has unique device identifiers and most applications use these identifiers in their configuration to know who their peers are. Once these identifiers are given to a device they typically do not change, even as the network topology changes.

BACnet devices use the Who-Is request to translate device identifiers into network addresses. This is very similar to a decentralized DNS service, but the names are unsigned integers. The request is broadcast on the network and the client waits around to listen for I-Am messages. The source address of the I-Am response is "bound" to the device identifier and most communications are unicast thereafter.

First, start up Wireshark on your workstation and a capture session with a BACnet capture filter:

```
udp and port 47808
```

You might start seeing BACnet traffic from your test device, and if you wait to power it on after starting your capture you should see at least a broadcast I-Am message. By looking in the I-Am packet decoding you will see some of its configuration parameters that should match what you expected them to be.

Now start the simplest tutorial application:

```
$ python samples/Tutorial/WhoIsIAm.py
```

Note: The samples folder contains a Tutorial folder holding all the samples that you will need too follow along this tutorial. Later, the folder *HandsOnLabs* will be used as it contains the samples that are fully explained in this document (see table of content)

You will be presented with a prompt (>), and you can get help:

```
> help

Documented commands (type help <topic>):
========================================
EOF  buggers  bugin  bugout  exit  gc  help  iam  shell  whois
```

The details of the commands are described in the next section.

1.1.5 Generating An I-Am

Now that the application is configured it is nice to see some BACnet communications traffic. Generate the basic I-Am message:

```
> iam
```

You should see Wireshark capture your I-Am message containing your configuration parameters. This is a "global broadcast" message. Your test device will see it but since your test device probably isn't looking for you, it will not respond to the message.

1.1.6 Binding to the Test Device

Next we want to confirm that your workstation can receive the messages the test device sends out. We do this by generating a generic Who-Is request. The request will be "unconstrained", meaning every device that hears the message will respond with their corresponding I-Am messages.

> **Caution:** Generating **unconstrained** Who-Is requests on a large network will create a LOT of traffic, which can lead to network problems caused by the resulting flood of messages.

To generate the Who-Is request:

```
> whois
```

You should see the Who-Is request captured in Wireshark along with the I-Am response from your test device, and then the details of the response displayed on the workstation console.:

```
> whois
> pduSource = <RemoteStation 50009:9>
iAmDeviceIdentifier = ('device', 1000)
maxAPDULengthAccepted = 480
segmentationSupported = segmentedBoth
vendorID = 8
```

There are a few different forms of the *whois* command supported by this simple application. You can see these with the help command:

```
> help whois
whois [ <addr>] [ <lolimit> <hilimit> ]
```

This is like a BNF syntax, the **whois** command is optionally followed by a BACnet device address, and then optionally followed by a low (address) limit and high (address) limit. The most common use of the Who-Is request is to look for a specific device given its device identifier:

```
> whois 1000 1000
```

If the site has a numbering scheme for groups of BACnet devices (i.e. grouped by building), then it is common to look for all the devices in a specific building as a group:

```
> whois 203000 203099
```

Every once in a while a contractor might install a BACnet device that hasn't been properly configured. Assuming that it has an IP address, you can send an **unconstrained Who-Is** request to the specific device and hope that it responds:

```
> whois 192.168.0.10

> pduSource = <Address 192.168.0.10>
iAmDeviceIdentifier = ('device', 1000)
maxAPDULengthAccepted = 1024
segmentationSupported = segmentedBoth
vendorID = 15
```

There are other forms of BACnet addresses used in BACpypes, but that is a subject of an other tutorial.

1.1.7 What's Next

The next tutorial describes the different ways this application can be run, and what the commands can tell you about how it is working. All of the "console" applications (i.e. those that prompt for commands) use the same basic commands and work the same way.

1.2 Running BACpypes Applications

All BACpypes sample applications have the same basic set of command line options so it is easy to move between applications, turn debugging on and and use different configurations. There may be additional options and command parameters than just the ones described in this section.

1.2.1 Getting Help

Whatever the command line parameters and additional options might be for an application, you can start with help:

```
$ python Tutorial/WhoIsIAm.py --help
usage: WhoIsIAm.py [-h] [--buggers] [--debug [DEBUG [DEBUG ...]]] [--color] [--ini␣
↪INI]

This application presents a 'console' prompt to the user asking for Who-Is and
I-Am commands which create the related APDUs, then lines up the corresponding
I-Am for incoming traffic and prints out the contents.

optional arguments:
  -h, --help            show this help message and exit
  --buggers             list the debugging logger names
  --debug [DEBUG [ DEBUG ... ]]
        DEBUG ::= debugger [ : fileName [ : maxBytes [ : backupCount ]]]
                        add console log handler to each debugging logger
  --color               use ANSI CSI color codes
  --ini INI             device object configuration file
```

1.2.2 Listing Debugging Loggers

The BACpypes library and sample applications make extensive use of the built-in *logging* module in Python. Every module in the library, along with every class and exported function, has a logging object associated with it. By attaching a log handler to a logger, the log handler is given a chance to output the progress of the application.

Because BACpypes modules are deeply interconnected, dumping a complete list of all of the logger names is a long list. Start out focusing on the components of the WhoIsIAm.py application:

```
$ python Tutorial/WhoIsIAm.py --buggers | grep __main__
__main__
__main__.WhoIsIAmApplication
__main__.WhoIsIAmConsoleCmd
```

In this sample, the entire application is called __main__ and it defines two classes.

1.2.3 Debugging a Module

Telling the application to debug a module is simple:

```
$ python WhoIsIAm.py --debug __main__
DEBUG:__main__:initialization
DEBUG:__main__:    - args: Namespace(buggers=False, debug=['__main__'], ini=<class
 ↪'bacpypes.consolelogging.ini'>)
DEBUG:__main__.WhoIsIAmApplication:__init__ (<bacpypes.app.LocalDeviceObject object␣
 ↪at 0xb6dd98cc>, '128.253.109.40/24:47808')
DEBUG:__main__:running
>
```

The output is the severity code of the logger (almost always DEBUG), the name of the module, class, or function, then some message about the progress of the application. From the output above you can see the application initializing, setting the args variable, creating an instance of the WhoIsIAmApplication class (with some parameters), and then declaring itself - running.

1.2.4 Debugging a Class

Debugging all of the classes and functions can generate a lot of output, so it is useful to focus on a specific function or class:

```
$ python Tutorial/WhoIsIAm.py --debug __main__.WhoIsIAmApplication
DEBUG:__main__.WhoIsIAmApplication:__init__ (<bacpypes.app.LocalDeviceObject object␣
 ↪at 0x9bca8ac>, '128.253.109.40/24:47808')
>
```

The same method is used to debug the activity of a BACpypes module, for example, there is a class called UDPActor in the UDP module:

```
$ python Tutorial/WhoIsIAm.py --ini BAC0.ini --debug bacpypes.udp.UDPActor
> DEBUG:bacpypes.udp.UDPActor:__init__ <bacpypes.udp.UDPDirector 128.253.109.
 ↪255:47808 at 0xb6d40d6c> ('128.253.109.254', 47808)
DEBUG:bacpypes.udp.UDPActor:response <bacpypes.comm.PDU object at 0xb6d433cc>
    <bacpypes.comm.PDU object at 0xb6d433cc>
        pduSource = ('128.253.109.254', 47808)
        pduData = x'81.04.00.37.0A.10.6D.45.BA.C0.01.28.FF.FF.00.00.B6.01.05.FD...'
```

In this sample, an instance of a UDPActor is created and then its response function is called with an instance of a PDU as a parameter. Following the function invocation description, the debugging output continues with the contents of the PDU. Notice, the protocol data is printed as a hex encoded string (and restricted to just the first 20 bytes of the message).

You can debug a function just as easily. Specify as many different combinations of logger names as necessary. Note, you cannot debug a specific function within a class.

1.2.5 Sending Debug Log to a file

The current –debug command line option takes a list of named debugging access points and attaches a StreamHandler which sends the output to sys.stderr. There is a way to send the debugging output to a RotatingFileHandler by providing a file name, and optionally maxBytes and backupCount. For example, this invocation sends the main application debugging to standard error and the debugging output of the bacpypes.udp module to the traffic.txt file:

```
$ python Tutorial/WhoIsIAm.py --debug __main__ bacpypes.udp:traffic.txt
```

By default the *maxBytes* is zero so there is no rotating file, but it can be provided, for example this limits the file size to 1MB:

```
$ python Tutorial/WhoIsIAm.py --debug __main__ bacpypes.udp:traffic.txt:1048576
```

If *maxBytes* is provided, then by default the *backupCount* is 10, but it can also be specified, so this limits the output to one hundred files:

```
$ python Tutorial/WhoIsIAm.py --debug __main__ bacpypes.udp:traffic.txt:1048576:100
```

Caution: The traffic.txt file will be saved in the local directory (pwd)

The definition of debug:

```
positional arguments:
    --debug [DEBUG [ DEBUG ... ]]
        DEBUG ::= debugger [ : fileName [ : maxBytes [ : backupCount ]]]
```

1.2.6 Changing INI Files

It is not unusual to have a variety of different INI files specifying different port numbers or other BACnet communications paramters.

Rather than swapping INI files, you can simply provide the INI file on the command line, overriding the default BACpypes.ini file. For example, I have an INI file for port 47808:

```
$ python Tutorial/WhoIsIAm.py --ini BAC0.ini
```

And another one for port 47809:

```
$ python Tutorial/WhoIsIAm.py --ini BAC1.ini
```

And I switch back and forth between them.

CHAPTER 2

Tutorial

This tutorial is a step-by-step walk through of the library describing the essential components of a BACpypes application and how the pieces fit together.

2.1 Clients and Servers

While exploring a library like BACpypes, take full advantage of Python being an interpreted language with an interactive prompt! The code for this tutorial is also available in the *Tutorial* subdirectory of the repository.

This tutorial will be using *comm.Client*, *comm.Server* classes, and the *comm.bind()* function, so start out by importing them:

```
>>> from bacpypes.comm import Client, Server, bind
```

Since the server needs to do something when it gets a request, it needs to provide a function to get it:

```
>>> class MyServer(Server):
...     def indication(self, arg):
...         print('working on', arg)
...         self.response(arg.upper())
...
```

Now create an instance of this new class and bind the client and server together:

```
>>> c = Client()
>>> s = MyServer()
>>> bind(c, s)
```

This only solves the downstream part of the problem, as you can see:

```
>>> c.request('hi')
('working on ', 'hi')
Traceback....
```

(continues on next page)

(continued from previous page)

```
....
NotImplementedError: confirmation must be overridden
```

So now we create a custom client class that does something with the response:

```
>>> class MyClient(Client):
...     def confirmation(self, pdu):
...         print('thanks for the ', pdu)
...
```

Create an instance of it, bind the client and server together and test it:

```
>>> c = MyClient()
>>> bind(c, s)
>>> c.request('hi')
('working on ', 'hi')
('thanks for ', 'HI')
```

Success!

2.2 Stacking with Debug

This tutorial uses the same *comm.Client*, *comm.Server* classes from the previous one, so continuing on from previous tutorial, all we need is to import the class:*comm.Debug*:

```
>>> from bacpypes.comm import Debug
```

Because there could be lots of **Debug** instances, it could be confusing if you didn't know which instance was generating the output. So initialize the debug instance with a name:

```
>>> d = Debug("middle")
```

As you can guess, this is going to go into the middle of a *stack* of objects. The *top* of the stack is a client, then *bottom* of a stack is a server. When messages are flowing from clients to servers they are called *downstream* messages, and when they flow from server to client they are *upstream* messages.

The *comm.bind()* function takes an arbitrary number of objects. It assumes that the first one will always be a client, the last one is a server, and the objects in the middle are hybrids which can be bound with the client to its left, and to the server on its right:

```
>>> bind(c, d, s)
```

Now when the client generates a request, rather than the message being sent to the MyServer instance, it is sent to the debugging instance, which prints out that it received the message:

```
>>> c.request('hi')
Debug(middle).indication
    - args[0]: hi
```

The debugging instance then forwards the message to the server, which prints its message. Completeing the requests *downstream* journey.:

```
working on hi
```

The server then generates a reply. The reply moves *upstream* from the server, through the debugging instance, this time as a confirmation:

```
Debug(middle).confirmation
    - args[0]: HI
```

Which is then forwarded *upstream* to the client:

```
thanks for the HI
```

This demonstrates how requests first move *downstream* from client to server; then cause the generation of replies that move *upstream* from server to client; and how the debug instance in the middle sees the messages moving both ways.

With clearly defined "envelopes" of protocol data, matching the combination of clients and servers into layers can provide a clear separation of functionality in a protocol stack.

2.3 Protocol Data Units

According to Wikipedia a *Protocol Data Unit* (PDU) is

> Information that is delivered as a unit among peer entities of a network and that may contain control information, address information, or data.

BACpypes uses a slight variation of this definition in that it bundles the address information with the control information. It considers addressing as part of how the data should be delivered, along with other concepts like how important the PDU data is relative to other PDUs.

The basic components of a PDU are the *comm.PCI* and *comm.PDUData* classes which are then bundled together to form the *comm.PDU* class.

All of the protocol interpreters written in the course of developing BACpypes have a concept of source and destination. The *comm.PCI* defines only two attributes, **pduSource** and **pduDestination**.

Note: Master/slave networks, are an exception. Messages sent by the master, contain only the destination (the source is implicit). Messages returned by the slaves have no addressing (both the source, and destination are implicit).

As a foundation layer, there are no restrictions on the form of the source and destination, they could be integers, strings or even objects. In general, the *comm.PDU* class is used as a base class for a series of stack specific components. UDP traffic have combinations of IP addresses and port numbers as source and destination, then that will be inherited by something that provides more control information, like delivery order or priority.

2.3.1 Exploring PDU's

Begin with importing the base class:

```
>>> from bacpypes.comm import PDU
```

Create a new PDU with some simple content:

```
>>> pdu = PDU(b"hello")
```

Caution: If you are not using Python 3, you don't need to specify the bytes type. >>> pdu = PDU("Hello")

We can then see the contents of the PDU as it will be seen on the network wire and by Wireshark - as a sequence of octets (printed as hex encoded strings):

```
>>> pdu.debug_contents()
    pduData = x'68.65.6C.6C.6F'
```

Now lets add some source and destination addressing information, so the message can be sent somewhere:

```
>>> pdu.pduSource = 1
>>> pdu.pduDestination = 2
>>> pdu.debug_contents()
    pduSource = 1
    pduDestination = 2
    pduData = x'68.65.6c.6c.6f'
```

Of course, we could have provided the addressing information when we created the PDU:

```
>>> pdu = PDU(b"hello", source=1, destination=2)
>>> pdu.debug_contents()
    pduSource = 1
    pduDestination = 2
    pduData = x'68.65.6C.6C.6F'
```

Tip: It is customary to allow missing attributes (be it protocol control information or data) as this allows the developer to mix keyword parameters with post-init attribute assignments.

2.3.2 BACnet PDUs

The basic PDU definition is fine for many protocols, but BACnet has two additional protocol parameters, described as attributes of the BACnet PCI information.

The *pdu.PCI* class extends the basic PCI with **pduExpectingReply** and **pduNetworkPriority**. The former is only used in MS/TP networks so the node generating the request will not pass the token before waiting some amount of time for a response, and the latter is a hint to routers, and devices with priority queues for network traffic, that a PDU is more or less important.

These two fields are assigned at the application layer and travel with the PDU as it travels through the stack.

2.3.3 Encoding and Decoding

The encoding and decoding process consists of consuming content from the source PDU and generating content in the destination. BACpypes *could* have used some kind of "visitor" pattern so the process did not consume the source, but typically when a layer has finished with PDU it will be sending some different PDU upstream or downstream so once the layer is finished, the PDU is not re-visited.

Note: This concept, where an object like a PDU is passed off to another function and is no longer "owned" by the builder, is difficult to accomplish in language environments without automatic garbage collection, but tremendiously simplifies our interpreter code.

PDUs nest the control information of one level into the data portion of the next level. So when decoding on the way up, it is customary to pass the control information along, even when it isn't strictly necessary.

The `pdu.PCI.update()` function is an example of a method that is used the way a "copy" operation might be used. The PCI classes, and nested versions of them, usually have an update function.

Decoding

Decoding always consumes some number of octets from the front of the PDU data. Lets create a pdu and then use decoding to consume it:

```
>>> pdu=PDU(b'hello!!')
>>> pdu.debug_contents()
    pduData = x'68.65.6c.6c.6f.21.21'
```

Consume 1 octet (x'68 = decimal 104'):

```
>>> pdu.get()
104
>>> pdu.debug_contents()
    pduData = x'65.6c.6c.6f.21.21'
```

Consume a short integer (two octets):

```
>>> pdu.get_short()
25964
>>> pdu.debug_contents()
    pduData = x'6c.6f.21.21'
```

Consume a long integer (four octets):

```
>>> pdu.get_long()
1819222305
>>> pdu.debug_contents()
    pduData = x''
>>>
```

And the PDU is now empty!

Encoding

We can then build the PDU contents back up through a series of *put* operations. A *put* is an implicit append operation:

```
>>> pdu.debug_contents()
    pduData = x''
>>> pdu.put(108)
>>> pdu.debug_contents()
    pduData = x'6c'

>>> pdu.put_short(25964)
>>> pdu.debug_contents()
    pduData = x'6c.65.6c'

>>> pdu.put_long(1819222305)
>>> pdu.debug_contents()
    pduData = x'6c.65.6c.6c.6f.21.21'
```

Note: There is no distinction between a PDU that is being taken apart (by get) and one that is being built up (by put).

2.4 Addressing

BACnet addresses come in five delicious flavors:

local station A message addressed to one device on the same network as the originator.

local broadcast A message addressed to all devices or nodes on the same network as the originator.

remote station A message addressed to one device on a different network than the originator.

remote broadcast A message addressed to all devices or nodes on a different network than the originator.

global broadcast A message addressed to all devices or nodes on all networks known any device on any network.

BACpypes address objects are used as the source and destination for PDUs and are also keys to dictionaries for looking up device in information and organizing requests and responses with devices.

2.4.1 Building an Address

The Address class other related classes are in the pdu module.

Local Stations

The Address class is the base class from which the other classes are derived, but for this tutorial, we'll start with the simplest:

```
>>> from bacpypes.pdu import LocalStation
```

Local station addresses are one or more octets of binary data. For the simplest networks they are a single octet, for Ethernet and BACnet/IP they are six octets long. There is no restriction on the length of an address in BACpypes.

A local station address is contructed by passing the octet string as bytes or a byte array, and their string representation is hex notation:

```
>>> addr1 = Address(b'123456')
>>> print(addr1)
0x313233343536
```

For local stations on simple networks the constructor will accept unsigned integers with the simple string output:

```
>>> addr2 = Address(12)
>>> print(addr2)
12
```

The underlying components of the address are always byte strings:

```
>>> addr1.addrAddr
b'123456'
>>> addr1.addrAddr
b'\x01'
```

When the byte string is six octets long and the next to last octet is 0xBA and the last octet is in the range 0xC0 to 0xCF, the string output and repr value will be presented as an IPv4 address:

```
>>> LocalStation(b'\1\2\3\4\xba\xc0')
<LocalStation 1.2.3.4>
```

and it will include the port number if it is not the standard port:

```
>>> LocalStation(b'\1\2\3\4\xba\xc3')
<LocalStation 1.2.3.4:47811>
```

Local Broadcast

The local broadcast address is used in the destination of a PDU that is to be sent to all of the devices on a network, and if the network layer can detect if it received a PDU as the result of another station broadcasting it. There are no parameters for constructing one:

```
>>> from bacpypes.pdu import LocalBroadcast
>>> print(LocalBroadcast())
*
```

The string output represents any address.

Remote Station

A remote station address is used in BACnet networking when the source and/or destination is on a network other than the one considered local. The first parameter is the network number, which must be a valid BACnet network number, and the second parameter is a byte string or unsigned integer like the local station:

```
>>> from bacpypes.pdu import RemoteStation
>>> print(RemoteStation(15, 75))
15:75
>>> print(RemoteStation(15, b'123456'))
15:0x313233343536
```

The string output is the network number and address separated by a colon.

Remote Broadcast

A remote broadcast station is used as a destination address when sending a PDU to all of the devices on a remote network. The only constructor parameter is the network number, which must be a valid BACnet network number:

```
>>> from bacpypes.pdu import RemoteBroadcast
>>> print(RemoteBroadcast(17))
17:*
```

The string output is the network number number, a colon, and an asterisk for any address.

GlobalBroadcast

The global broadcast address is used to send PDUs to all devices. It has no constructor parameters:

```
>>> from bacpypes.pdu import GlobalBroadcast
>>> print(GlobalBroadcast())
*:*
```

The string output is an asterisk for any network, a colon, and an asterisk for and address.

2.4.2 Address Parsing

The basic Address class can parse the string form of all of the address types and a few more for older applications and notation that has appeared in other tutorials.

Note: The Address class cannot "morph" into an instance of one of its subclasses so to determine what kind of address it is check the addrType attribute.

For example:

```
>>> from bacpypes.pdu import Address
>>> Address(1).addrType == Address.localStationAddr
True
```

And addresses created this way are identical:

```
>>> Address(1) == LocalStation(b'\01')
True
```

Unlike the LocalStation, the Address can take the string form of an integer:

```
>>> Address("2") == LocalStation(b'\02')
True
```

And can interpret hex strings of various types:

```
>>> Address("0x0304") == LocalStation(b'\3\4')
True
>>> Address("X'050607'") == LocalStation(b'\5\6\7')
True
```

It interprets the asterisk as a local broadcast:

```
>>> Address("*") == LocalBroadcast()
True
```

And remote stations and remote broadcasts mathing the other output:

```
>>> Address("1:2") == RemoteStation(1, 2)
True
>>> Address("3:*") == RemoteBroadcast(3)
True
```

And the global broadcast:

```
>>> Address("*:*") == GlobalBroadcast()
True
```

IPv4 Addresses

Because they appear so often, the address parsing has special patterns for recognizing IPv4 addresses in CIDR notation along with an optional port number:

```
>>> Address("192.168.1.2").addrAddr
b'\xc0\xa8\x01\x02\xba\xc0'

>>> Address("192.168.1.2:47809").addrAddr
b'\xc0\xa8\x01\x02\xba\xc1'
```

For addresses that also include a subnet mask to calculate broadcast addresses, the CIDR notation is available:

```
>>> hex(Address("192.168.3.4/24").addrSubnet)
'0xc0a80300'
```

And for calculating the address tuple for use with socket functions:

```
>>> Address("192.168.5.6/16").addrBroadcastTuple
('192.168.255.255', 47808)
```

2.5 Command Shell

Debugging small, short lived BACpypes applications is fairly simple with the abillity to attach debug handlers to specific components of a stack when it starts, and then reproducing whatever situation caused the mis-behaviour.

For longer running applications like gateways it might take some time before a scenario is ready, in which case it is advantageous to start and stop the debugging output, without stopping the application.

For some debugging scenarios it is beneficial to force some values into the stack, or delete some values and see how the application performs. For example, perhaps deleting a routing path associated with a network.

Python has a cmd module that makes it easy to embed a command line interpreter in an application. BACpypes extends this interpreter with some commands to assist debugging and runs the interpreter in a separate thread so it does not interfere with the BACpypes *core.run()* functionality.

2.5.1 Application Additions

Adding the console command shell is as simple as importing it:

```
from bacpypes.consolecmd import ConsoleCmd
```

And creating an instance:

```
# console
ConsoleCmd()
```

In addition to the other command line options that are typically included in BACpypes applications, this can be wrapped:

```
if '--console' in sys.argv:
    ConsoleCmd()
```

2.5.2 Command Recall

The BACpypes command line interpreter maintains a history (text file) of the commands executed, which it reloads upon startup. Pressing the *previous command* keyboard shortcut (up-arrow key) recalls previous commands so they can be executed again.

2.5.3 Basic Commands

All of the commands supported are listed in the *consolecmd* documentation. The simplest way to learn the commands is to try them:

```
$ python Tutorial/SampleConsoleCmd.py
> hi
*** Unknown syntax: hi
```

There is some help:

```
> help

Documented commands (type help <topic>):
========================================
EOF  buggers  bugin  bugout  exit  gc  help  shell
```

And getting a list of the buggers:

```
> buggers
no handlers
  __main__
  bacpypes
  bacpypes.apdu
  bacpypes.apdu.APCI
  ...
  bacpypes.vlan.Network
  bacpypes.vlan.Node
```

Attaching a debugger:

```
> bugin bacpypes.task.OneShotTask
handler to bacpypes.task.OneShotTask added
```

Then removing it later:

```
> bugout bacpypes.task.OneShotTask
handler to bacpypes.task.OneShotTask removed
```

And finally exiting the application:

```
> exit
Exiting...
```

2.5.4 Adding Commands

Adding additional commands is as simple as providing an additional function. Add these lines to SampleConsoleCmd.py:

```
class SampleConsoleCmd(ConsoleCmd):

    def do_something(self, arg):
        """something <arg> - do something"""
        print("do something", arg)
```

The ConsoleCmd will trap a help request `help something` into printing out the documnetation string.:

```
> help

Documented commands (type help <topic>):
========================================
EOF  buggers  bugin  bugout  exit  gc  help  nothing  shell  **something**

> help something
something <arg> - do something
>
```

2.5.5 Example Cache Commands

Add these functions to **SampleConsoleCmd.py**. The concept is to force values into an application cache, delete them, and dump the cache. First, setting values is a *set* command:

```
class SampleConsoleCmd(ConsoleCmd):

    my_cache= {}

    def do_set(self, arg):
        """set <key> <value> - change a cache value"""
        if _debug: SampleConsoleCmd._debug("do_set %r", arg)

        key, value = arg.split()
        self.my_cache[key] = value
```

Then delete cache entries with a *del* command:

```
def do_del(self, arg):
    """del <key> - delete a cache entry"""
    if _debug: SampleConsoleCmd._debug("do_del %r", arg)

    try:
        del self.my_cache[arg]
    except:
        print(arg, "not in cache")
```

And to verify, dump the cache:

```
def do_dump(self, arg):
    """dump - nicely print the cache"""
    if _debug: SampleConsoleCmd._debug("do_dump %r", arg)
    print(self.my_cache)
```

And when the sample application is run, note the new commands show up in the help list:

```
$ python Tutorial/SampleConsoleCmd.py
> help

Documented commands (type help <topic>):
========================================
EOF      bugin   **del**   exit  help     **set**    something
buggers  bugout  **dump**  gc    nothing  shell
```

You can get help with the new commands:

```
> help set
set <key> <value> - change a cache value
```

Lets use these new commands to add some items to the cache and dump it out:

```
> set x 12
> set y 13
> dump
{'x': '12', 'y': '13'}
```

Now add a debugger to the main application, which can generate a lot output for most applications, but this one is simple:

```
> bugin __main__
handler to __main__ added
```

Now we'll get some debug output when the cache entry is deleted:

```
> del x
DEBUG:__main__.SampleConsoleCmd:do_del 'x'
```

We can see a list of buggers and which ones have a debugger attached:

```
> buggers __main__
handlers: __main__
* __main__
  __main__.SampleApplication
  __main__.SampleConsoleCmd
```

Check the contents of the cache:

```
> dump
DEBUG:__main__.SampleConsoleCmd:do_dump ''
{'y': '13'}
```

All done:

```
> exit
Exiting...
```

2.6 Controllers and IOCB

The IO Control Block (IOCB) is an object that holds the parameters for some kind of operation or function and a place for the result. The IOController processes the IOCBs it is given and returns the IOCB back to the caller.

For this tutorial section, import the IOCB and IOController:

```
>>> from bacpypes.iocb import IOCB, IOController
```

2.6.1 Building an IOCB

Build an IOCB with some arguments and keyword arguments:

```
>>> iocb = IOCB(1, 2, a=3)
```

The parameters are kept for processing:

```
>>> iocb.args
(1, 2)
>>> iocb.kwargs
{'a': 3}
```

2.6.2 Make a Controller

Now we need a controller to process this request. This controller is just going to add and multiply the arguments together:

```
class SomeController(IOController):

    def process_io(self, iocb):
        self.complete_io(iocb, iocb.args[0] + iocb.args[1] * iocb.kwargs['a'])
```

Now create an instance of the controller and pass it the request:

```
>>> some_controller = SomeController()
>>> some_controller.request_io(iocb)
```

First, you'll notice that *request_io()* was called rather than the processing function directly. This intermediate layer between the caller of the service and the thing providing the service can be detached from each other in a variety of different ways.

For example, there are some types of controllers that can only process one request at a time and these are derived from *IOQController*. If the application layer requests IOCB processing faster than the controller can manage (perhaps because it is waiting for some networking functions) the requests will be queued.

In other examples, the application making the request is in a different process or on a different machine, so the *request_io()* function builds a remote procedure call wrapper around the request and manages the response. This is similar to an HTTP proxy server.

Similarly, inside the controller it calls *self.complete_io()* so if there is some wrapper functionality the code inside the *process_io()* function doesn't need to worry about it.

2.6.3 Check the Result

There are a few ways to check to see if an IOCB has been processed. Every IOCB has an *Event* from the *threading* built in module, so the application can check to see if the event is set:

```
>>> iocb.ioComplete
<threading._Event object at 0x101349590>
>>> iocb.ioComplete.is_set()
True
```

There is also an IOCB state which has one of a collection of enumerated values:

```
>>> import bacpypes
>>> iocb.ioState == bacpypes.iocb.COMPLETED
True
```

And the state could also be aborted:

```
>>> iocb.ioState == bacpypes.iocb.ABORTED
False
```

Almost all controllers return some kind of information back to the requestor in the form of some data. In this example, it's just a number:

```
>>> iocb.ioResponse
7
```

But we can provide some invalid combination of arguments and the exception will show up in the *ioError*:

```
>>> iocb = IOCB(1, 2)
>>> some_controller.request_io(iocb)
>>> iocb.ioError
KeyError('a',)
```

The types of results and errors depend on the controller.

2.6.4 Getting a Callback

When a controller completes the processing of a request, the IOCB can contain one or more functions to be called. First, define a callback function:

```
def call_me(iocb):
    print("call me, %r or %r" % (iocb.ioResponse, iocb.ioError))
```

Now create a request and add the callback function:

```
>>> iocb = IOCB(1, 2, a=10)
>>> iocb.add_callback(call_me)
```

Pass the IOCB to the controller and the callback function is called:

```
>>> some_controller.request_io(iocb)
call me, 21 or None
```

2.6.5 Threading

The IOCB module is thread safe, but the IOController derived classes may not be. The thread initiating the request to the controller may simply wait for the completion event to be set:

```
>>> some_controller.request_io(iocb)
>>> iocb.ioComplete.wait()
```

But for this to work correctly, the IOController must be running in a separate thread, or there won't be any way for the event to be set.

If the iocb has callback functions, they will be executed in the thread context of the controller.

2.7 Capabilities

The *capabilty* module is used to mix together classes that provide both separate and overlapping functionality. The original design was motivated by a component architecture where collections of components that needed to be mixed together were specified outside the application in a database.

The sample applications in this section are available in tutorial folder. Note that you can also find them in the unit test folder as they are part of the test suites.

Start out importing the classes in the module:

```
>>> from bacpypes.capability import Capability, Collector
```

2.7.1 Transforming Data

Assume that the application needs to transform data in a variety of different ways, but the exact order of those functions isn't specified, but all of the transformation functions have the same signature.

First, create a class that is going to be the foundation of the transformation process:

```
class BaseCollector(Collector):

    def transform(self, value):
        for fn in self.capability_functions('transform'):
            value = fn(self, value)

        return value
```

If there are no other classes mixed in, the *transform()* function doesn't do anything:

```
>>> some_transformer = BaseCollector()
>>> some_transformer.transform(10)
10
```

2.7.2 Adding a Transformation

Create a *Capability* derived class that transforms the value slightly:

```
class PlusOne(Capability):

    def transform(self, value):
        return value + 1
```

Now create a new class that mixes in the base collector:

```
class ExampleOne(BaseCollector, PlusOne):
    pass
```

And our transform function incorporates the new behavior:

```
>>> some_transformer = ExampleOne()
>>> some_transformer.transform(10)
11
```

2.7.3 Add Another Transformation

Here is a different transformation class:

```
class TimesTen(Capability):

    def transform(self, value):
        return value * 10
```

And the new class works as intended:

```
class ExampleTwo(BaseCollector, TimesTen):
    pass

>>> some_transformer = ExampleTwo()
>>> some_transformer.transform(10)
100
```

And the classes can be mixed in together:

```
class ExampleThree(BaseCollector, PlusOne, TimesTen):
    pass

>>> some_transformer = ExampleThree()
>>> some_transformer.transform(10)
110
```

The order of the classes makes a difference:

```
class ExampleFour(BaseCollector, TimesTen, PlusOne):
    pass

>>> some_transformer = ExampleFour()
>>> some_transformer.transform(10)
101
```

CHAPTER 3

Migration

If you are upgrading your BACpypes applications to a newer version there are guidelines of the types of changes you might need to make.

3.1 Version 0.14.1 to 0.15.0

This update contains a significant number of changes to the way the project code is organized. This is a guide to updating applications that use BACpypes to fit the new API.

The guide is divided into a series of sections for each type of change.

3.1.1 LocalDeviceObject

There is a new *service* sub-package where the functionality to support a specific type of behavior is in a separate module. The module names within the *service* sub-package are inspired by and very similar to the names of Clauses 13 through 17.

The *bacpypes.service.device* module now contains the definition of the *LocalDeviceObject* as well as mix-in classes to support Who-Is, I-Am, Who-Has, and I-Have services.

If your application contained this:

```
from bacpypes.app import LocalDeviceObject, BIPSimpleApplication
```

Update it to contain this:

```
from bacpypes.app import BIPSimpleApplication
from bacpypes.service.device import LocalDeviceObject
```

3.1.2 Application Subclasses

The *Application* class in the *bacpypes.app* module no longer supports services by default, they are mixed into derived classes as needed. There are very few applications that actually took advantage of the *AtomicReadFile* and *AtomicWriteFile* services, so when these were moved to their own service module *bacpypes.service.file* it seems natural to move the implementations of the other services to other modules as well.

Moving this code to separate modules will facilitate BACpypes applications building additional service modules to mix into the default ones or replace default implementations with ones more suited to their local application requirements.

The exception to this is the *BIPSimpleApplication*, is the most commonly used derived class from *Application* and I anticipated that by having it include *WhoIsIAmServices* and *ReadWritePropertyServices* allowed existing applications to run with fewer changes.

If your application contained this:

```
class MyApplication(Application):
    ...
```

And you want to keep the old behavior, replace it with this:

```
from bacpypes.service.device import WhoIsIAmServices
from bacpypes.service.object import ReadWritePropertyServices

class MyApplication(Application, WhoIsIAmServices, ReadWritePropertyServices):
    ...
```

3.1.3 Client-only Applications

The *Application* class no longer requires a value for the *localDevice* or *localAddress* parameters. BACpypes applications like that omit these parameters will only be able to initiate confirmed or unconfirmed services that do not require these objects or values. They would not be able to respond to Who-Is requests for example.

Client-only applications are useful when it would be advantageous to avoid the administrative overhead for configuring something as a device, such as network analysis applications and very simple trend data gather applications. They are also useful for BACpypes applications that run in a Docker container or "in the cloud".

Sample client-only applications will be forthcoming.

3.1.4 Simplified Requests

Some of the service modules now have additional functions that make it easier to initiate requests. For example, in the *WhoIsIAmServices* class there are functions for initiating a Who-Is request by a simple function:

```
def who_is(self, low_limit=None, high_limit=None, address=None):
    ...
```

Validating the parameters, building the *WhoIsRequest* PDU and sending it downstream is all handled by the function.

If your application builds common requests then you can use the new functions or continue without them. If there are common requests that you would like to make and have built into the library your suggestions are always welcome.

CHAPTER 4

Hands-on Lab

BACpypes comes with a variety of sample applications. Some are a framework for building larger applications. Some are standalone analysis tools that don't require a connection to a network.

The first samples you should have a look too are located inside the *samples/HandsOnLab* folder. Those samples are fully explained in the documentation so you can follow along and get your head around BACpypes.

Other less documented samples are available directly in the *samples* folder.

CHAPTER 5

Glossary

5.1 Glossary

upstream Something going up a stack from a server to client.

downstream Something going down a stack from a client to a server.

stack A sequence of communication objects organized in a semi-linear sequence from the application layer at the top to the physical networking layer(s) at the bottom.

discoverable Something that can be determined using a combination of BACnet objects, properties and services. For example, discovering the network topology by using Who-Is-Router-To-Network, or knowing what objects are defined in a device by reading the *object-list* property.

CHAPTER 6

Release Notes

6.1 Release Notes

This page contains release notes.

6.1.1 Version 0.13.6

There have been lots of changes in the span between the previous published version and this one and I haven't quite figured out how to extract the relevent content from the git log. More to come.

6.1.2 Version 0.13.0

This is a big release, with no API changes since the 0.12.1 version, but the setup now detects which version of Python is running and switches between source directories: *py25*, *py27*, and *py34*.

There is now a *test* directory, so in addition to the *build* and *install* options there is *test*, which uses nose for running the scripts:

```
$ python setup.py test
```

If you have more than one version of Python installed on your machine you can use tox to run the tests will all of the supported versions (currently limited to Python2.7 and Python3.4 due to substantial changes in unittest):

```
$ tox
```

At some point there will be a documentation page that decribes the changes between the distributions, as well as a guide for new applications.

6.1.3 Version 0.12.1

- Add backup-in-progress to the Device Status enumeration r331

- Correct the restoreFailure in BackupState r332
- Check for read-only object when writing to a file r333
- Wrong initial value for no segmentation (old enumeration syntax) r334
- Wrong parameter r335
- Missed variable name change r336
- Mask errors writing the history file like they are when reading r337
- Make sure that the vendor identifier is provided, and that localDate and localTime are not r338
- Add simple string parsing to Date and Time r339
- Bump the version number, provide more focused classifiers, include release notes r340

6.1.4 Version 0.12.0

- Switch from distutils to setuptools to build a wheel r323
- Updated to use twine to upload after building both an egg and a wheel r324
- ReallyLongCamelCaseTypo r325
- The pieces inside the AtomicReadFileACK should not have been context encoded, but the choice is context encoded r326
- Additional properties and object types to get closer to 2012 edition r327
- Additional properties and enumerations r328
- Replace 'except X, T:' with 'except X as T:' for more modern code r329
- Bump the version number and include release notes this time r330

6.1.5 Version 0.11.0

- Merge the 0.10.6 release r311
- Examples of a RecurringTask and using that to read property values. r312
- Minor documentation update, adding –color option r313
- IP-to-IP router sample r314
- Additional helper application for decoding UDP packet contents in hex r315
- The 'description' property is optional, by giving it a default value it was always being created. r316
- Spelling typo r317
- Missing enumerations r318
- WhatIsNetworkNumber and NetworkNumberIs decoding (no other support yet) r319
- typo r320
- reStructured text version of readme r321
- Bump the version number r322

6.1.6 Version 0.10.6

- Release notes from previous version. r304
- The accessCredential object type was missing. r305
- Incorrect number of formatting parameters to match actual parameters, only appeared as warnings during debugging, but is definitely annoying. r306
- New ReadRange sample code to assist with a developer question, keep them coming! r307
- The ClientCOV components are not supposed to be context encoded. r308
- A change to make sure that an array property isn't None (uninitialized) before attempting to index into it. r309
- Bump the version number and update these release notes. r310

6.1.7 Version 0.10.5

- Bill Roberts submitted a patch to clean up an old underscore, and I missed the edit earlier. Thanks Bill! r302
- Bump the version number, release notes to come later. r303

6.1.8 Version 0.10.4

This version contains bug fixes.

- Some BACneteer had an issue with MultiState Value Objects so I added some sample code to present one of these on the network so I could check to make sure the encoding/decoding of property values was working correctly.

  There was an issue with constructed data with elements that were arrays, the elements should have had Python list semantics rather than BACnet array semantics, so there is some additional checking for this in the decoding. r282
- A branch was created for dealing with unicode strings rather than the default string encoding. No final decision has been made on this issue, I need more experience. r283 r284 r285 r286 r287 r289 r290 r291 r292
- Delete an unecessary import (a.k.a., "flake"). r288
- Handle the various combinations of present/missing values for the object identifier and object list keyword arguments to the device object better. r293
- The Random Analog Value Object sample code used the object identifier keyword argument in a non-standard way, and I thought this fixed it, but it seems to have re-introduced some debugging code as well. This needs investigation. r294
- For sequences that specify "any atomic value" which is application encoded, the constructed data decoder presents those values as instances of one of the subclasses of Atomic rather that presenting them as Any which needs more work decoding for the BACpypes developer. r295
- This patch takes advantage of the r295 and applies it to the Schedule Object and the TimeValue, used in SpecialEvent, used in the exception Schedule. r296
- In the Read Property sample code, if the value has a debug_contents API then it is called and this gives a little bit more detailed output. r297
- New Schedule Object sample code. r298
- The fileIdentifier parameter of the Atomic Read/Write File services is application encoded, not context encoded. r299

- Bill Roberts submitted some patches to clean up element encoding errors, thank you Bill! r300
- Bump the version number and release. Notes to be committed later. r301

6.1.9 Version 0.10.3

This version contains some enhancements and bug fixes.

- Sangeeth Saravanaraj submitted an enchancement that allows the ConsoleCmd class to accept stdin and stdout parameters and replaces the print statements with self.stdout.write calls. Thank you! r276
- This is a new filter that looks for Who-Is and I-Am messages related to a specific device instance number in a pcap file. r277
- This minor enhancement allows longs in the object type for an object identifier `__init__` parameter rather than just ints. r278
- Application service access point encode and decoding errors bail out of the effort rather than raising an error. There is a very long running application that I have that would decode an APDU incorrectly every once in a great while, but it was very difficult to track down. I think this was actually field device that was adding additional cruft on the end of a packet and BACpypes would raise an error. I need the stack to toss these errant PDUs out as if they never happened. It would be nice if there was a logging hook that developers could use to track when this happens. r279
- This is a pair of sample applications for proprietary object types and proprietary properties to demonstrate how to extend the core types. r280
- Bump the version number and update these release notes. r281

6.1.10 Version 0.10.2

This version contains bug fixes.

- The invokeID for outbound client requests must be unique per server, but can be the same value for different servers. I had solved this problem once before in the sample HTTP server code, but didn't migrate the code into the core library. At some point there was some other code that couldn't generate more than 255 requests, so this never got tested. Other BACneteers are more aggressive! r272
- The segment count of a confirmed ack is at least one, even if there is no PDU data. This was solved on the client side (in the client segmentation state machine for seeing if requests needed to be segmented on the way out) but not on the server side. This fixes that bug. r273
- The ReadPropertyMultipleServer code would see that an object didn't exist and build an error response, which was oblitered by the default code at the bottom of the loop so it was never returned. Now if any of the read access specifications refers to an object that doesn't exist the request will correctly return an error. r274
- Bump the version number and update these release notes. r275

6.1.11 Version 0.10.1

This version contains more contributions that should have been included in the previous release, but I updated the library in a different order than the mailing list. Sigh.

- The library did not return the correct error for writing to immutable properties. r269
- The lowerCamelCase for CharacterStringValue objects was incorrect and didn't match the enumeration value. r270

- Bump the version number and update these release notes. r271

6.1.12 Version 0.10

This version contains updates courtesy of contributions from other BACpypes users, of whom I am grateful!

- The consolelogging module ConfigArgumentParser inherits from the built-in ArgumentParser class, but the parse_args didn't have the same function signature. r264
- The MultipleReadProperty new sample application has a list of points and it shows how to put those points into a queue so each one of them can be read sequentially. r265
- The Read Access and Stream Access choices in the atomic file services were backwards, stream access is choice zero (0) and record access is one (1). r266
- In the process of confirming that the file access services were in fact wrong, I decided to update the sample applications and give them better names. r267
- Bump the version number and update these release notes. r268

6.1.13 Version 0.9.5

I have been working more on converting PDU's into JSON content that can be archived and searched in MongoDB.

- Simple bug, while I was updated in the `__init__` calling chain I got the class name wrong. r260
- When there is network layer traffic on a port that is not the "local port" it still needs to be processed by the local `NetworkServiceElement`. And trying to debug this problem, there was no debugger for the NSE! r261
- As I have been shuffling around JSON-like content in various applications it became harder and harder to manage if the result of calling `dict_content` was going to return PCI layer information (the NPCI, APCI, or BVLCI), or the "data" portion of the packet. I also took the opportunity to use simpler names. r262
- Bump the version number and update these release notes. r263

6.1.14 Version 0.9.4

This revision is an annouced release. The combination of r258 and r256 makes this important to get out to the community sooner rather than later.

- The `TimeSynchronizationRequest` application layer PDUs have their `time` parameter application encoded, not context encoded. r258
- Bump the version number and update these release notes. r259

6.1.15 Version 0.9.3

This release just has some minor bug fixes, but in order to get a large collection of applications running quickly it was simpler to make minor release and install it on other machines. The version was release to PyPI but never annouced.

Revisions r255 through r257.

- A simple copy/paste error from some other sample code. r255
- When shuffling data around to other applications and databases (like MongoDB) there are problems with raw string data, a.k.a., octet strings, or in Python3 terms byte strings. This is a simple mechanism to make hex strings out of the data portion of tag data. This is subject to change to some other format as we get more experience with data in other applications. r256

- Remove the "flakes" (modules that were imported but not used). r257

6.1.16 Version 0.9.2

Apart from the usual bug fixes and small new features, this release changes almost all of the `__init__` functions to use `super()` rather than calling the parent class initializer.

New School Initialization

For example, while the old code did this:

```
class Foo(Bar):

    def __init__(self):
        Bar.__init__(self)
        self.foo = 12
```

New the code does this:

```
class Foo(Bar):

    def __init__(self, *args, **kwargs):
        super(Foo, self).__init__(*args, **kwargs)
        self.foo = 12
```

If you draw an inheritance tree starting with `PDUData` at the top and ending with something like `ReadPropertyRequest` at the bottom, you will see lots of branching and merging. Calling the parent class directly may lead to the same base class being "initialized" more than once which was causing all kinds of havoc.

Simply replacing the one with the new wasn't quite good enough however, because it could lead to a situation where a keyword arguement needed to be "consumed" if it existed because it didn't make sense for the parent class or any of its parents. In many cases this works:

```
class Foo(Bar):

    def __init__(self, foo_arg=None, *args, **kwargs):
        super(Foo, self).__init__(*args, **kwargs)
        self.foo = 12
```

When the parent class initializer gets called the `foo_arg` will be a regular parameter and won't be in the `kwargs` that get passed up the inheritance tree. However, with `Sequence` and `Choice` there is no knowledge of what the keyword parameters are going to be without going through the associated element lists. So those two classes go to great lengths to divide the kwargs into "mine" and "other".

New User Data PDU Attribute

I have been working on a fairly complicated application that is a combination of being a BBMD on multiple networks and router between them. The twist is that there are rules that govern what segments of the networks can see each other. To manage this, there needed to be a way to attach an object at the bottom of the stack when a PDU is received and make sure that context information is maintained all the way up through the stack to the application layer and then back down again.

To accomplish this there is a `pduUserData` attribute you can set and as long as the stack is dealing with that PDU or the derived encoded/decoded PDUs, that reference is maintained.

Revisions r246 through r254.

- The sample HTTP server was using the old syle argument parser and the old version didn't have the options leading to confusion. r246
- Set the 'reuse' flag for broadcast sockets. A BACneteer has a workstation with two physical adapters connected to the same LAN with different IP addresses assigned for each one. Two BACpypes applications were attempting to bind to the same broadcast address, this allows that scenerio to work. r247
- Fix the help string and add a little more error checking to the ReadPropertyMultiple.py sample application. r248
- Add the –color option to debugging. This wraps the output of the LoggingFormatter with ANSI CSI escape codes so the output from different log handlers is output in different colors. When debugging is turned on for many modules it helps! r249
- The WriteProperty method now has a ''direct" parameter, this fixes the function signatures of the sample applications to include it. r250
- Change the `__init__` functions to use `super()`, see explanation above. r251
- Bump the minor version number. r252
- Update the getting started document to include the new color debugging option. There should be more explanation of what that means exactly, along with a link to the Wikipedia color code tables. r253
- Update these release notes. r254

6.1.17 Version 0.9.1

Most of this release is just documentation, but it includes some new functionality for translating PDUs into dictionaries. The new `dict_contents` functions will most likely have some bugs, so consider that API unstable.

Revisions r238 through r245.

- For some new users of BACpypes, particularly those that were also new to BACnet, it can be a struggle getting something to work. This is the start of a new documentation section to speed that process along. r238 r239 r240
- For multithreaded applications it is sometimes handly to override the default spin value, which is the maximum amount of time that the application should be stuck in the asyncore.loop() function. The developer could import the core module and change the CORE value before calling run(), but that seems excessively hackish. r241
- Apparently there should not be a dependancy on `setuptools` for developers that want to install the library without it. In revision r227 I changed the setup.py file, but that broke the release script. I'm not completely sure this is correct, but it seems to work. r242
- This revision includes a new dict_contents() function that encodes PDU content into a dict-like object (a real `dict` by default, but the developer can provide any other class that supports `__setitem__`). This is the first step in a long road to translate PDU data into JSON, then into BSON to be streamed into a MongoDB database for analysis applications. r243
- Bump the version number before releasing it. r244
- Update these release notes. r245

6.1.18 Version 0.9

There are a number of significant changes in BACpypes in this release, some of which may break existing code so it is getting a minor release number. While this project is getting inexorably closer to a 1.0 release, we're not there yet.

The biggest change is the addition of a set of derived classes of `Property` that match the names of the way properties are described in the standard; `OptionalProperty`, `ReadableProperty`, and `WritableProperty`.

This takes over from the awkward and difficult-to-maintain combinations of `optional` and `mutable` constructor parameters. I went through the standard again and matched the class name with the object definition and it is much cleaner.

This change was brought about by working on the BACowl project where I wanted the generated ontology to more closely match the content of the standard. This is the first instance where I've used the ontology design to change application code.

Revisions r227 through r234.

- At some point `setuptools` was replaced with `distutils` and this needed to change while I was getting the code working on Windows. r227
- Added the new property classes and renamed the existing `Property` class instances. There are object types that are not complete (not every object type has every property defined) and these will be cleaned up and added in a minor release in the near future. r228
- The UDP module had some print statements and a traceback call that sent content to stdout, errors should go to stderr. r229
- With the new property classes there needed to be a simpler and cleaner way managing the __init__ keyword parameters for a `LocalDeviceObject`. During testing I had created objects with no name or object identifier and it seemed like some error checking was warrented, so that was added to `add_object` and `delete_object`. r230
- This commit is the first pass at changing the way object classes are registered. There is now a new `vendor_id` parameter so that derived classes of a standard object can be registered. For example, if vendor Snork has a custom SnorkAnalogInputObject class (derived from `AnalogInputObject` of course) then both classes can be registered.

  The `get_object_class` has a cooresponding `vendor_id` parameter, so if a client application is looking for the appropriate class, pass the `vendorIdentifier` property value from the deivce object of the server and if there isn't a specific one defined, the standard class will be returned.

  The new and improved registration function would be a lot nicer as a decorator, but optional named parameters make and interesting twist. So depending on the combination of parameters it returns a decorator, which is an interesting twist on recursion.

  At some point there will be a tutorial covering just this functionality, and before this project hits version 1.0, there will be a similar mechanism for vendor defined enumerations, especially `PropertyIdentifier`, and this will also follow the BACowl ontology conventions.

  This commit also includes a few minor changes like changing the name `klass` to the not-so-cute `cls`, `property` to `propid` because the former is a reserved word, and the dictionary of registered objects from `object_types` to `registered_object_types`. r231
- Simple wrapping of the command line argument interpretation for a sample application. r232
- The `CommandableMixin` isn't appropriate for `BinaryValueObject` type, so I replaced it with a `DateValueObject`. r233
- I managed to install Sphinx on my Windows laptop and this just added a build script to make it easier to put in these release notes. r235
- This adds the relaease notes page and a link to it for documentation, committed so I could continue working on it from a variety of different places. I usually wouldn't make a commit just for this unless I was working in a branch, but because I'm working in the trunk rather than using a service like DropBox I decided to let myself get away with it. r234 r236
- Committed the final version of these notes and bumped the minor version number. r237

6.1.19 Version 0.8

Placeholder for 0.8 release notes.

Revisions r224 through r226.

- Placeholder for comments about revision 224. r224
- Placeholder for comments about revision 225. r225
- Bump the minor version number. r226

6.1.20 Version 0.7.5

Placeholder for 0.8 release notes.

Revisions r217 through r223.

- Placeholder for comments about revision 217. r217
- Placeholder for comments about revision 218. r218
- Placeholder for comments about revision 219. r219
- Placeholder for comments about revision 220. r220
- Placeholder for comments about revision 221. r221
- Placeholder for comments about revision 222. r222
- Bump the patch version number. r223

6.1.21 Version 0.7.4

Lost to the sands of time.

CHAPTER 7

Modules

Tip: Documentation intended for BACpypes developers.

7.1 BACpypes Modules

7.1.1 Core

Core

All applications have to have some kind of outer blcok.

Globals

`core.`**`running`**
: This is a boolean that the application is running. It can be turned off by an application, but the *`stop()`* function is usually used.

`core.`**`taskManager`**
: This is a reference to the `TaskManager` instance that is used to schedule some operation. There is only one task manager instance in an application.

`core.`**`deferredFns`**
: This is a list of function calls to make after all of the asyncore.loop processing has completed. This is a list of (fn, args, kwargs) tuples that are appended to the list by the *`deferred()`* function.

`core.`**`sleeptime`**
: This value is used to "sleep" the main thread for a certian amount of before continuing on to the asyncore loop. It is used to be friendly to other threads that may be starved for processing time. See *`enable_sleeping()`*.

Functions

core.**run**(*spin=SPIN*, *sigterm=stop*, *sigusr1=print_stack*)

Parameters

- **spin** – the amount of time to wait if no tasks are scheduled
- **sigterm** – a function to call when SIGTERM is signaled, defaults to stop
- **sigusr1** – a function to call when SIGUSR1 is signaled, defaults to print_stack

This function is called by a BACpypes application after all of its initialization is complete.

The spin parameter is the maximum amount of time to wait in the sockets asyncore loop() function that waits for network activity. Setting this to a large value allows the application to consume very few system resources while there is no network activity. If the application uses threads, setting this to a large value will starve the child threads for time.

The sigterm parameter is a function to be installed as a signal handler for SIGTERM events. For historical reasons this defaults to the stop() function so that Ctrl-C in interactive applications will exit the application rather than raise a KeyboardInterrupt exception.

The sigusr1 parameter is a function to be installed as a signal handler for SIGUSR1 events. For historical reasons this defaults to the print_stack() function so if an application seems to be stuck on waiting for an event or in a long running loop the developer can trigger a "stack dump".

The sigterm and sigusr1 parameters must be None when the run() function is called from a non-main thread.

core.**stop**(**args*)

Parameters **args** – optional signal handler arguments

This function is called to stop a BACpypes application. It resets the *running* boolean value. This function also installed as a signal handler responding to the TERM signal so you can stop a background (deamon) process:

```
$ kill -TERM 12345
```

core.**print_stack**(*sig*, *frame*)

Parameters

- **sig** – signal
- **frame** – stack trace frame

core.**deferred**(*fn*, **args*, ***kwargs*)

Parameters

- **fn** – function to call
- **args** – regular arguments to pass to fn
- **kwargs** – keyword arguments to pass to fn

This function is called to postpone a function call until after the asyncore.loop processing has completed. See *run()*.

core.**enable_sleeping**([*stime*])

Parameters **stime** – amount of time to sleep, defaults to one millisecond

BACpypes applications are generally written as a single threaded application, the stack is not thread safe. However, applications may use threads at the application layer and above for other types of work. This function

allows the main thread to sleep for some small amount of time so that it does not starve child threads of processing time.

When sleeping is enabled, and it only needs to be enabled for multithreaded applications, it will put a damper on the throughput of the application.

Comm

All applications have to have some kind of outer blcok.

Globals

comm.**client_map**
This is . . .

comm.**server_map**
This is . . .

comm.**service_map**
This is . . .

comm.**element_map**
This is . . .

Functions

comm.**bind**(**args*)

Parameters **args** – a list of clients and servers to bind together in a stack

Protocol Data Units

A Protocol Data Unit (PDU) is the name for a collection of information that is passed between two entities. It is composed of Protcol Control Information (PCI) - information about addressing, processing instructions - and data. The set of classes in this module are not specific to BACnet.

class comm.**PCI**

pduSouce
The source of a PDU. The datatype and composition of the address is dependent on the client/server relationship and protocol context. The source may be *None*, in which case it has no source or the source is implicit.

pduDestination
The destination of a PDU. The datatype and composition of the address is dependent on the client/server relationship and protocol context. The destination may be *None*, in which case it has no destination or the destination is implicit.

__init__(*[source=addr][,destination=addr]*)

Parameters

- **source** (*addr*) – the initial source value
- **destination** (*addr*) – the initial destination value

Protocol Control Information is generally the context information and/or other types of processing instructions.

class comm.**PDUData**

The PDUData class has functions for extracting information from the front of the data octet string, or append information to the end. These are helper functions but may not be applicable for higher layer protocols which may be passing significantly more complex data.

pduData

This attribute typically holds a simple octet string, but for higher layers of a protocol stack it may contain more abstract pieces or components.

get()

Extract a single octet from the front of the data. If the octet string is empty this will raise a DecodingError.

get_data(*len*)

Parameters **len** (*integer*) – the number of octets to extract.

Extract a number of octets from the front of the data. If there are not at least *len* octets this will raise a DecodingError exception.

get_short()

Extract a short integer (two octets) from the front of the data.

get_long()

Extract a long integer (four octets) from the front of the data.

put(*ch*)

Parameters **ch** (*octet*) – the octet to append to the end

put_data(*data*)

Parameters **data** (*string*) – the octet string to append to the end

put_short(*n*)

Parameters **integer** (*short*) – two octets to append to the end

put_long(*n*)

Parameters **integer** (*long*) – four octets to append to the end

class comm.**PDU**(*PCI*, *PDUData*)

The PDU class combines the PCI and PDUData classes together into one object.

Protocol Stack Classes

class comm.**Client**

class comm.**Server**

class comm.**Debug**

class comm.**Echo**

Application Classes

class comm.**ServiceAccessPoint**

class comm.**ApplicationServiceElement**

class comm.**NullServiceElement**

class comm.**DebugServiceElement**

BACnet Protocol Data Units

This is a long line of text.

Addressing

class pdu.**Address**
This is a long line of text.

addrType
This is a long line of text.

addrNet
This is a long line of text.

addrLen
This is a long line of text.

addrAddr
This is a long line of text.

decode_address(*addr*)

Parameters **addr** (*string*) – address specification to interpret

This is a long line of text.

__str__()

__repr__()
This method overrides the built-in function to provide a little bit better string, using *__str__* for help.

__hash__()
This method is used to allow addresses to be used as keys in dictionaries which require keys to be hashable.

Note: Once an address is used in a dictionary is should be considered immutable.

__eq__(*arg*)
__ne__(*arg*)

Parameters **arg** – another address, or something that can be interpreted as an address

This is a long line of text.

class pdu.**LocalStation**(*Address*)
This is a long line of text.

class pdu.**RemoteStation**(*Address*)
This is a long line of text.

class pdu.**LocalBroadcast**(*Address*)
This is a long line of text.

class pdu.**RemoteBroadcast**(*Address*)
This is a long line of text.

class pdu.**GlobalBroadcast**(*Address*)
This is a long line of text.

Extended PCI

This is a long line of text.

class pdu.**PCI**(*_PCI*)
This is a long line of text.

pduExpectingReply
This is a long line of text.

pduNetworkPriority
This is a long line of text.

class pdu.**PDU**(*PCI*, *PDUData*)
This is a long line of text.

Debugging

All applications use some kind of debugging.

Globals

debugging.**_root**
This is a long line of text.

Functions

debugging.**ModuleLogger**(*globs*)

Parameters **globs** – dictionary of module globals

This function, posing as an instance creator, returns a . . .

Function Decorators

debugging.**function_debugging**()
This function decorates a function with instances of buggers that are named by the function name combined with the module name. It is used like this:

```
@function_debugging
def some_function(arg):
    if _debug: some_function._debug("some_function %r", arg)
    # rest of code
```

This results in a bugger called **module.some_function** that can be accessed by that name when attaching log handlers.

Note: This should really be called **debug_function** or something like that.

Classes

class debugging.**DebugContents**
This is a long line of text.

_debug_contents
This is a long line of text.

debug_contents(*indent=1*, *file=sys.stdout*, *_ids=None*)

Parameters

- **indent** – function to call
- **file** – regular arguments to pass to fn
- **_ids** – keyword arguments to pass to fn

This is a long line of text.

class debugging.**LoggingFormatter**(*logging.Formatter*)
This is a long line of text.

__init__()
This is a long line of text.

format(*record*)

Parameters **record** (*logging.LogRecord*) – record to format

This function converts the record into a string. It uses the regular formatting function that it overrides, then if any of the parameters inherit from *DebugContents* (or duck typed by providing a **debug_contents** function) the message is extended with the deconstruction of those parameters.

class debugging.**Logging**
This is a long line of text.

Note: Now that Python supports class decorators, this should really be a class decorator called **debug_class** or something like that.

Console Logging

This module provides a function that is typically used to attach a log handler to a **_debug** logger that has been created by the methods in *debugging*.

Functions

consolelogging.**ConsoleLogHandler**(*loggerRef=''*, *level=logging.DEBUG*)

Parameters

- **loggerRef** (*string*) – function to call
- **level** – logging level

This is a long line of text.

Console Command

Python has a cmd module that makes it easy to embed a command line interpreter in an application. BACpypes extends this interpreter with some commands to assist debugging and runs the interpreter in a separate thread so it does not interfere with the BACpypes *core.run()* functionality.

Functions

consolecmd.**console_interrupt**(**args*)

Parameters **args** –

Classes

class consolecmd.**ConsoleCmd**(*cmd.Cmd*, *Thread*)

__init__(*prompt="> "*, *allow_exec=False*)

Parameters

- **prompt** (*string*) – prompt for commands
- **allow_exec** (*boolean*) – allow non-commands to be executed

run()
Begin execution of the application's main event loop. Place this after the the initialization statements.

do_something(*args*)

Parameters **args** – commands

Template of a function implementing a console command.

Commands

help
List an application's console commands:

```
> help
Documented commands (type help <topic>):
========================================
EOF  buggers  bugin  bugout  exit  gc  help  nothing  shell
```

gc
Print out garbage collection information:

```
> gc
Module                          Type                          Count  dCount   dRef
bacpypes.object                 OptionalProperty                787       0      0
bacpypes.constructeddata        Element                         651       0      0
bacpypes.object                 ReadableProperty                362       0      0
bacpypes.object                 WritableProperty                 44       0      0
__future__                      _Feature                          7       0      0
Queue                           Queue                             2       0      0
bacpypes.pdu                    Address                           2       0      0
```

(continues on next page)

(continued from previous page)

```
bacpypes.udp                    UDPActor                          2       1       4
bacpypes.bvllservice            UDPMultiplexer                    1       0       0
bacpypes.app                    DeviceInfoCache                   1       0       0

Module                          Type                          Count  dCount    dRef
bacpypes.udp                    UDPActor                          2       1       4
```

bugin <name>
Attach a debugger.:

```
> bugin bacpypes.task.OneShotTask
handler to bacpypes.task.OneShotTask added
```

bugout <name>
Detach a debugger.:

```
> bugout bacpypes.task.OneShotTask
handler to bacpypes.task.OneShotTask removed
```

buggers
Get a list of the available buggers.:

```
> buggers
no handlers
__main__
bacpypes
bacpypes.apdu
bacpypes.apdu.APCI
...
bacpypes.vlan.Network
bacpypes.vlan.Node
```

exit
Exit a BACpypes Console application.:

```
> exit
Exiting...
```

Errors

This module defines the exception class for errors it detects in the configuration of the stack or in encoding or decoding PDUs. All of these exceptions are derived from ValueError (in Python's built-in exceptions module).

Classes

class errors.**ConfigurationError**
This error is raised when there are required components that are missing or defined incorrectly. Many components, such as instances of *comm.Client* and *comm.Server*, are required to be bound together in specific ways.

class errors.**EncodingError**
This error is raised while PDU data is being encoded, which typically means while some structured data is being turned into an octet stream or some other simpler structure. There may be limitations of the values being encoded.

class errors.**DecodingError**
This error is raised while PDU data is being decoded, which typically means some unstructured data like an octet stream is being turned into structured data. There may be values in the PDU being decoded that are not appropriate, or not enough data such as a truncated packet.

Singleton

Singleton classes are a design pattern which returns the same object for every 'create an instance' call. In the case of BACpypes there can only be one instance of a *task.TaskManager* and all of the tasks are scheduled through it. The design pattern "hides" all of the implementation details of the task manager behind its interface.

There are occasions when the task manager needs to provide additional functionality, or a derived class would like a change to intercept the methods. In this case the developer can create a subclass of `TaskManager`, then create an instance of it. Every subsequent call to get a task manager will return this special instance.

Classes

class singleton.**Singleton**
By inheriting from this class, all calls to build an object will return the same object.

class singleton.**SingletonLogging**
This special class binds together the metaclasses from both this singleton module and from the *debugging.Logging*. Python classes cannot inherit from two separate metaclasses at the same time, but this class takes advantage of Pythons ability to have multiple inheritance of metaclasses.

Task

A **task** is something that needs to be done. Tasks come in a variety of flavors:

- *OneShotTask* - do something once
- *OneShotDeleteTask* - do something once, then delete the task object
- *RecurringTask* - do something at regular intervals

Every derived class of one of these classes must provide a *process_task* method which will be called at the next opportunity available to the application. All task processing is expected to be cooperative, which means that it must be written so that it is cognizant that other tasks may also be waiting for a chance to be processed.

Tasks are *installed* when they should be scheduled for processing, may be *suspended* or removed from scheduling, and then may be *resumed* or re-installed.

Singleton Task Manager

All operations involving tasks are directed to a single instance of *TaskManager* or an instance of a derived class. If the developer creates a derived class of *TaskManager* and an instance of it *before* the *core.run()* function is called, that instance will be used to schedule tasks and return the next task to process.

Globals

task.**_task_manager**
This is a long line of text.

task.**_unscheduled_tasks**
This is a long line of text.

Functions

task.**OneShotFunction**(*fn*, **args*, ***kwargs*)

Parameters

- **fn** – function to schedule
- **args** – function to schedule
- **kwargs** – function to schedule

This is a long line of text.

task.**FunctionTask**(*fn*, **args*, ***kwargs*)

Parameters **fn** – function to update

This is a long line of text.

task.**RecurringFunctionTask**(*interval*, *fn*, **args*, ***kwargs*)

Parameters **fn** – function to update

This is a long line of text.

Function Decorators

task.**recurring_function**(*interval*)

Parameters **interval** – interval to call the function

This function will return a decorator which will wrap a function in a task object that will be called at regular intervals and can also be called as a function. For example:

```
@recurring_function(5000)
def my_ping(arg=None):
    print "my_ping", arg
```

The my_ping object is a task that can be installed, suspended, and resumed like any other task. This is installed to run every 5s and will print:

```
my_ping None
```

And can also be called as a regular function with parameters, so calling my_ping("hello") will print:

```
my_ping hello
```

Classes

class task.**_Task**
This is a long line of text.

install_task(*when=None*)

Parameters **when** (*float*) – time task should be processed

This is a long line of text.

process_task()

Parameters **when** (*float*) – time task should be processed

This is a long line of text.

suspend_task()

Parameters **when** (*float*) – time task should be processed

This is a long line of text.

resume_task()

Parameters **when** (*float*) – time task should be processed

This is a long line of text.

class task.**OneShotTask**
This is a long line of text.

class task.**OneShotDeleteTask**
This is a long line of text.

class task.**RecurringTask**
This is a long line of text.

class task.**TaskManager**
This is a long line of text.

install_task(*task*)

Parameters **task** – task to be installed

This is a long line of text.

suspend_task(*task*)

Parameters **task** – task to be suspended

This is a long line of text.

resume_task(*task*)

Parameters **task** – task to be resumed

This is a long line of text.

get_next_task()
This is a long line of text.

process_task()
This is a long line of text.

Event

At the heart of *core.run()* is a call to the **select** function of the built in select module. That function is provided a list of file descriptors and will exit when there is activity on one of them.

In a multi-threaded application, if the main thread is waiting for IO activity then child threads need a mechanism to "wake up" the main thread. This may be because the child thread has detected some timeout.

An instance of this class is used by the *task.TaskManager* to wake up the main thread when tasks are scheduled by child threads. If the child thread is requesting "as soon as possible" execution of the task, then scheduling the task wakes up the main thread, which causes it to be processed.

Note: This is not available on Windows platforms, which may suffer from a small preformance hit. This can be mitigated somewhat by changing the **SPIN** value in the **core** module.

Classes

class event.**WaitableEvent**

The methods in this class provide the same interface as **asyncore.file_dispatcher** and the ones that are typically used in multi-threaded applications the way **Threading.Event** objects are used.

These methods use an internal pipe to provide a "read" and "write" file descriptors. There are no direct references to this pipe, only through the file descriptors that are linked to it.

__init__()

The internal file descriptors which are understood by the **asyncore.loop** call in *core.run()* are created by calling **os.pipe()**, then initialization continues to the usual **asyncore.file_dispatcher** initializer.

__del__()

When an instance of this class is deleted, the file references to the "read" and "write" sides of the pipe are closed. The OS will then delete the pipe.

readable()

This method returns `True` so it will always be included in the list of file-like objects when waiting for IO activity.

writable()

This method returns `False` becuase there is never any pending write activity like there would be for a actual file or socket.

handle_read()

This method performs no activity. If an instance of this event is "set" then the only way to clear it is by calling *clear()* which will read the pending character out of the pipe.

handle_write()

This function is never called because *writable()* always returns `False`.

handle_close()

This method is called when a close is requested, so this in turn passes it to the **asyncore.file_dispatcher.close** function.

wait(*timeout=None*)

Parameters **timeout** (*float*) – maximum time to wait for the event to be set

Similar to the way the **asyncore.loop** function will wait for activity on a file descriptor, **select.select** is used by this method to wait for some activity on the "read" side of its internal pipe.

The *set()* function will write to the "write" side of the pipe, so the "read" side will have activity and the select function will exit.

This function returns `True` if the "event" is "set".

isSet()

This method calls *wait()* with a zero timeout which essentially probes the pipe to see if there is data waiting, which in turn implies the "event" is "set".

set()
Setting the event involves writing a single character to the internal pipe, but only if there is no data in the pipe.

clear()
Clearing the event involves reading the character that was written to the intrenal pipe, provided one is there. If there is no data in the pipe then the **os.read** function would stall the thread.

7.1.2 UDP Communications

UDP

User Datagram Protocol is wonderful...

Classes

class udp.**UDPDirector**(*asyncore.dispatcher*, *Server*, *ServiceAccessPoint*, *Logging*)
This is a long line of text.

__init__(*self*, *address*, *timeout=0*, *actorClass=UDPActor*, *sid=None*, *sapID=None*)

Parameters

- **address** – the initial source value
- **timeout** – the initial source value
- **actorClass** – the initial source value
- **sid** – the initial source value
- **sapID** – the initial source value

This is a long line of text.

AddActor(*actor*)

Parameters **actor** – the initial source value

This is a long line of text.

RemoveActor(*actor*)

Parameters **actor** – the initial source value

This is a long line of text.

GetActor(*address*)

Parameters **address** – the initial source value

This is a long line of text.

handle_connect()
This is a long line of text.

readable()
This is a long line of text.

handle_read()
This is a long line of text.

writable()
This is a long line of text.

handle_write()
This is a long line of text.

handle_close()
This is a long line of text.

indication(*pdu*)
This is a long line of text.

_response(*pdu*)

class udp.**UDPActor**(*Logging*)
This is a long line of text.

director
This is a long line of text.

peer
This is a long line of text.

timeout
This is a long line of text.

timer
This is a long line of text.

__init__(*director*, *peer*)

Parameters

- **director** – the initial source value
- **peer** – the initial destination value

This is a long line of text.

IdleTimeout()
This is a long line of text.

indication(*pdu*)

Parameters **pdu** – the initial source value

This is a long line of text.

response(*pdu*)

Parameters **pdu** – the initial source value

This is a long line of text.

class udp.**UDPPickleActor**(*UDPActor*, *Logging*)

indication(*pdu*)

Parameters **pdu** – the initial source value

This is a long line of text.

response(*pdu*)

Parameters **pdu** – the initial source value

This is a long line of text.

BACnet Virtual Link Layer

BACnet virtual link layer...

PDU Base Types

This is a long line of text.

class bvll.**BVLCI**(*PCI*, *DebugContents*, *Logging*)

bvlciType

bvlciFunction

bvlciLength

This is a long line of text.

class bvll.**BVLPDU**(*BVLCI*, *PDUData*)
This is a long line of text.

PDU Types

This is a long line of text.

class bvll.**Result**(*BVLCI*)

Broadcast Distribution Table

This is a long line of text.

class bvll.**ReadBroadcastDistributionTable**(*BVLCI*)
This is a long line of text.

class bvll.**ReadBroadcastDistributionTableAck**(*BVLCI*)
This is a long line of text.

class bvll.**WriteBroadcastDistributionTable**(*BVLCI*)
This is a long line of text.

Foreign Devices

This is a long line of text.

class bvll.**FDTEntry**(*DebugContents*)
This is a long line of text.

class bvll.**RegisterForeignDevice**(*BVLCI*)
This is a long line of text.

class bvll.**ReadForeignDeviceTable**(*BVLCI*)
This is a long line of text.

class bvll.**ReadForeignDeviceTableAck**(*BVLCI*)
This is a long line of text.

class bvll.**DeleteForeignDeviceTableEntry**(*BVLCI*)
This is a long line of text.

Message Broadcasting

This is a long line of text.

class bvll.**OriginalUnicastNPDU**(*BVLPDU*)
This is a long line of text.

class bvll.**OriginalBroadcastNPDU**(*BVLPDU*)
This is a long line of text.

class bvll.**DistributeBroadcastToNetwork**(*BVLPDU*)
This is a long line of text.

class bvll.**ForwardedNPDU**(*BVLPDU*)
This is a long line of text.

BACnet Virtual Link Layer Service

BACnet virtual link layer...

UDP Multiplexing

class bvll.**UDPMultiplexer**

__init__(*addr=None*, *noBroadcast=False*)

Parameters

- **addr** – address to bind
- **noBroadcast** – option for separate broadcast socket

This is a long line of text.

indication(*server*, *pdu*)

Parameters

- **server** – multiplexer reference
- **pdu** – message to process

This is a long line of text.

confirmation(*client*, *pdu*)

Parameters

- **server** – multiplexer reference
- **pdu** – message to process

This is a long line of text.

class bvll.**_MultiplexClient**

multiplexer
This is a long line of text.

__init__(*mux*)

Parameters **mux** – multiplexer reference

This is a long line of text.

confirmation(*pdu*)

Parameters **pdu** – message to process

This is a long line of text.

class bvll.**_MultiplexServer**

multiplexer
This is a long line of text.

__init__(*mux*)

Parameters **mux** – multiplexer reference

This is a long line of text.

confirmation(*pdu*)

Parameters **pdu** – message to process

This is a long line of text.

Annex H - Tunneling

class bvll.**BTR**

__init__()
This is a long line of text.

indication(*pdu*)

Parameters **pdu** – message to process

This is a long line of text.

confirmation(*pdu*)

Parameters **pdu** – message to process

This is a long line of text.

add_peer(*peerAddr*[, *networks*])

Parameters

- **peerAddr** – peer address
- **networks** – list of networks reachable by peer

This is a long line of text.

delete_peer(*peerAddr*)

Parameters **peerAddr** – peer address

This is a long line of text.

Annex J - B/IP

Service Access Point Types

class bvll.**BIPSAP**(*ServiceAccessPoint*)

__init__()
This is a long line of text.

sap_indication(*pdu*)

Parameters **pdu** – message to process

This is a long line of text.

sap_confirmation(*pdu*)

Parameters **pdu** – message to process

This is a long line of text.

class bvll.**BIPSimple**(*BIPSAP*, *Client*, *Server*)

indication(*pdu*)

Parameters **pdu** – message to process

This is a long line of text.

confirmation(*pdu*)

Parameters **pdu** – message to process

This is a long line of text.

class bvll.**BIPForeign**(*BIPSAP*, *Client*, *Server*, *OneShotTask*)

indication(*pdu*)

Parameters **pdu** – message to process

This is a long line of text.

confirmation(*pdu*)

Parameters **pdu** – message to process

This is a long line of text.

register(*addr*, *ttl*)

Parameters

- **addr** – message to process
- **ttl** – time-to-live

This is a long line of text.

unregister()
This is a long line of text.

process_task()
This is a long line of text.

class bvll.**BIPBBMD**(*BIPSAP*, *Client*, *Server*, *RecurringTask*)

__init__(*addr*)

Parameters **addr** – address of itself

This is a long line of text.

indication(*pdu*)

Parameters **pdu** – message to process

This is a long line of text.

confirmation(*pdu*)

Parameters **pdu** – message to process

This is a long line of text.

RegisterForeignDevice(*addr*, *ttl*)

Parameters

- **addr** – address of foreign device
- **ttl** – time-to-live

This is a long line of text.

DeleteForeignDeviceTableEntry(*addr*)

Parameters **addr** – address of foreign device to delete

This is a long line of text.

process_task()
This is a long line of text.

add_peer(*addr*)

Parameters **addr** – address of peer to add

This is a long line of text.

delete_peer(*addr*)

Parameters **addr** – addess of peer to delete

This is a long line of text.

Service Element

class bvll.**BVLLServiceElement**(*ApplicationServiceElement*)

indication(*pdu*)

Parameters **pdu** – message to process

This is a long line of text.

confirmation(*pdu*)

Parameters **pdu** – message to process

This is a long line of text.

7.1.3 TCP Communications

TCP

Transmission Control Protocol is wonderful...

Client Classes

class tcp.**TCPClientDirector**(*Server*, *ServiceAccessPoint*)

This is a long line of text.

__init__(*address*, *timeout=0*, *actorClass=UDPActor*)

Parameters

- **address** – the initial source value
- **timeout** – the initial source value
- **actorClass** – the initial source value

This is a long line of text.

AddActor(*actor*)

Parameters **actor** – the initial source value

This is a long line of text.

RemoveActor(*actor*)

Parameters **actor** – the initial source value

This is a long line of text.

GetActor(*address*)

Parameters **address** – the initial source value

This is a long line of text.

connect(*address*, *reconnect=0*)

Parameters

- **address** – address to establish a connection
- **reconnect** – timer value

disconnect(*address*)

Parameters **address** – address to disconnect

indication(*pdu*)

This is a long line of text.

class tcp.**TCPClient**(*asyncore.dispatcher*)

__init__(*peer*)

Parameters **peer** – This is a long line of text.

This is a long line of text.

handle_connect()
This is a long line of text.

handle_expt()
This is a long line of text.

readable()
This is a long line of text.

handle_read()
This is a long line of text.

writable()
This is a long line of text.

handle_write()
This is a long line of text.

handle_close()
This is a long line of text.

indication(*pdu*)

Parameters **pdu** – data to send

This is a long line of text.

class tcp.**TCPClientActor**(*Logging*)
This is a long line of text.

director
This is a long line of text.

peer
This is a long line of text.

timeout
This is a long line of text.

timer
This is a long line of text.

__init__(*director*, *peer*)

Parameters

- **director** – the initial source value
- **peer** – the initial destination value

This is a long line of text.

handle_close()
This is a long line of text.

IdleTimeout()
This is a long line of text.

indication(*pdu*)

Parameters **pdu** – the initial source value

This is a long line of text.

response(*pdu*)

Parameters **pdu** – the initial source value

This is a long line of text.

Flush()
This is a long line of text.

class tcp.**TCPPickleClientActor**(*PickleActorMixIn*, *TCPClientActor*)
This is a long line of text.

Server Classes

class tcp.**TCPServerDirector**(*asyncore.dispatcher*, *Server*, *ServiceAccessPoint*)

__init__(*address*, *listeners=5*, *timeout=0*, *reuse=False*, *actorClass=TCPServerActor*)

Parameters

- **address** – socket for connection
- **listeners** – socket for connection
- **timeout** – socket for connection
- **reuse** – socket for connection
- **actorClass** – socket for connection

This is a long line of text.

handle_accept()
This is a long line of text.

handle_close()
This is a long line of text.

AddActor(*actor*)

Parameters **actor** – the initial source value

This is a long line of text.

RemoveActor(*actor*)

Parameters **actor** – the initial source value

This is a long line of text.

GetActor(*address*)

Parameters **address** – the initial source value

This is a long line of text.

indication(*pdu*)
This is a long line of text.

class tcp.**TCPServer**(*asyncore.dispatcher*)

__init__(*sock*, *peer*)

Parameters

- **sock** – socket for connection
- **peer** – This is a long line of text.

This is a long line of text.

handle_connect()
This is a long line of text.

readable()
This is a long line of text.

handle_read()
This is a long line of text.

writable()
This is a long line of text.

handle_write()
This is a long line of text.

handle_close()
This is a long line of text.

indication(*pdu*)

Parameters **pdu** – data to send

This is a long line of text.

class tcp.**TCPServerActor**(*TCPServer*)
This is a long line of text.

director
This is a long line of text.

peer
This is a long line of text.

timeout
This is a long line of text.

timer
This is a long line of text.

__init__(*director*, *sock*, *peer*)

Parameters

- **director** – the initial source value
- **sock** – socket for connection
- **peer** – the initial destination value

This is a long line of text.

handle_close()
This is a long line of text.

IdleTimeout()
This is a long line of text.

indication(*pdu*)

Parameters **pdu** – the initial source value

This is a long line of text.

response(*pdu*)

Parameters **pdu** – the initial source value

This is a long line of text.

Flush()
This is a long line of text.

class tcp.**TCPPickleServerActor**(*PickleActorMixIn*, *TCPServerActor*)
This is a long line of text.

Streaming Packets

class tcp.**StreamToPacket**(*Client*, *Server*)

Packetize(*pdu*, *streamBuffer*)
This is a long line of text.

indication(*pdu*)
This is a long line of text.

confirmation(*pdu*)
This is a long line of text.

class tcp.**StreamToPacketSAP**(*ApplicationServiceElement*, *ServiceAccessPoint*)

Stream Pickling

class tcp.**PickleActorMixIn**

indication(*pdu*)

Parameters **pdu** – the initial source value

This is a long line of text.

response(*pdu*)

Parameters **pdu** – the initial source value

This is a long line of text.

BACnet Streaming Link Layer

BACnet streaming link layer...

PDU Base Types

class bsll.**BSLCI**(*PCI*)

bslciType

bslciFunction

bslciLength

This is a long line of text.

class bsll.**BSLPDU**(*BVSCI*, *PDUData*)
This is a long line of text.

Service Requests

class bsll.**Result**(*BVLCI*)

bslciResultCode

This is a long line of text.

class bsll.**ServiceRequest**(*BSLCI*)

class bsll.**AccessRequest**(*BSLCI*)

class bsll.**AccessChallenge**(*BSLCI*)

class bsll.**AccessResponse**(*BSLCI*)

Device-To-Device Stream

class bsll.**DeviceToDeviceAPDU**(*BSLPDU*)

Router-To-Router Stream

class bsll.**RouterToRouterNPDU**(*BSLPDU*)

Proxy-To-Server Stream

class bsll.**ProxyToServerUnicastNPDU**(*BSLPDU*)

class bsll.**ProxyToServerBroadcastNPDU**(*BSLPDU*)

class bsll.**ServerToProxyUnicastNPDU**(*BSLPDU*)

class bsll.**ServerToProxyBroadcastNPDU**(*BSLPDU*)

LAN Emulation Stream

ClientToLESUnicastNPDU(BSLPDU

class bsll.**ClientToLESBroadcastNPDU**(*BSLPDU*)

class bsll.**LESToClientUnicastNPDU**(*BSLPDU*)

class bsll.**LESToClientBroadcastNPDU**(*BSLPDU*)

BACnet Streaming Link Layer Service

BACnet streaming link layer. . .

Streaming Packets

bsllservice.**_Packetize**(*data*)

Parameters **data** – octet stream to slice into packets

This is a long line of text.

class bsllservice.**_StreamToPacket**(*StreamToPacket*)
This is a long line of text.

User Information

This is a long line of text.

class bsllservice.**UserInformation**

__init__(***kwargs*)

Parameters

- **username** (*string*) – the user name
- **password** (*string*) – the user password
- **allServices** (*boolean*) –
- **deviceToDeviceService** (*boolean*) –
- **routerToRouterService** (*boolean*) –
- **proxyService** (*boolean*) –
- **laneService** (*boolean*) –
- **proxyNetwork** (*boolean*) –

This is a long line of text.

Connection State

Every thing is connected and every connection has a state.

- NOT_AUTHENTICATED - no authentication attempted
- REQUESTED - access request sent to the server (client only)
- CHALLENGED - access challenge sent to the client (server only)
- AUTHENTICATED - authentication successful

This is a long line of text.

class bsllservice.**ConnectionState**
This is a long line of text.

address
This is a long line of text.

service
This is a long line of text.

connected
This is a long line of text.

accessState
This is a long line of text.

challenge
This is a long line of text.

userinfo
This is a long line of text.

proxyAdapter
This is a long line of text.

Service Adapter

This is a long line of text.

class bsllservice.**ServiceAdapter**
This is a long line of text.

__init__(*mux*)
This is a long line of text.

authentication_required(*addr*)
This is a long line of text.

get_default_user_info(*addr*)
This is a long line of text.

get_user_info(*username*)
This is a long line of text.

add_connection(*conn*)
This is a long line of text.

remove_connection(*conn*)
This is a long line of text.

service_request(*pdu*)
This is a long line of text.

service_confirmation(*conn*, *pdu*)
This is a long line of text.

class bsllservice.**NetworkServiceAdapter**(*ServiceAdapter*, *NetworkAdapter*)
This is a long line of text.

TCP Multiplexing

This is a long line of text.

class bsllservice.**TCPServerMultiplexer**(*Client*)
This is a long line of text.

__init__(*addr=None*)

Parameters **addr** – address to bind

This is a long line of text.

request(*pdu*)

Parameters **pdu** – message to process

This is a long line of text.

indication(*server*, *pdu*)

Parameters

- **server** – multiplexer reference
- **pdu** – message to process

This is a long line of text.

confirmation(*pdu*)

Parameters **pdu** – message to process

This is a long line of text.

do_AccessRequest(*conn*, *bslpdu*)

Parameters

- **conn** – message to process
- **bslpdu** – message to process

This is a long line of text.

do_AccessResponse(*conn*, *bslpdu*)

Parameters

- **conn** – message to process
- **bslpdu** – message to process

This is a long line of text.

class bsllservice.**TCPClientMultiplexer**(*Client*)

This is a long line of text.

__init__()

This is a long line of text.

request(*pdu*)

Parameters **pdu** – message to process

This is a long line of text.

indication(*server*, *pdu*)

Parameters

- **server** – multiplexer reference
- **pdu** – message to process

This is a long line of text.

confirmation(*pdu*)

Parameters **pdu** – message to process

This is a long line of text.

do_AccessChallenge(*conn*, *bslpdu*)

Parameters

- **conn** – message to process
- **bslpdu** – message to process

This is a long line of text.

class bsllservice.**TCPMultiplexerASE**(*ApplicationServiceElement*)
This is a long line of text.

__init__(*self*, *mux*)
This is a long line of text.

indication(**args*, ***kwargs*)

Parameters

- **addPeer** – peer address to add
- **delPeer** – peer address to delete

This is a long line of text.

Device-to-Device Service

This is a long line of text.

class bsllservice.**DeviceToDeviceServerService**(*NetworkServiceAdapter*)
This is a long line of text.

process_npdu(*npdu*)
This is a long line of text.

service_confirmation(*conn*, *pdu*)
This is a long line of text.

class bsllservice.**DeviceToDeviceClientService**(*NetworkServiceAdapter*)
This is a long line of text.

process_npdu(*npdu*)
This is a long line of text.

connect(*addr*)
This is a long line of text.

connect_ack(*conn*, *pdu*)
This is a long line of text.

service_confirmation(*conn*, *pdu*)
This is a long line of text.

Router-to-Router Service

This is a long line of text.

class bsllservice.**RouterToRouterService**(*NetworkServiceAdapter*)
This is a long line of text.

process_npdu(*npdu*)
This is a long line of text.

connect(*addr*)
This is a long line of text.

connect_ack(*conn*, *pdu*)
This is a long line of text.

add_connection(*conn*)
This is a long line of text.

remove_connection(*conn*)
This is a long line of text.

service_confirmation(*conn*, *pdu*)
This is a long line of text.

Proxy Service

This is a long line of text.

class bsllservice.**ProxyServiceNetworkAdapter**(*NetworkAdapter*)
This is a long line of text.

process_npdu(*npdu*)
This is a long line of text.

service_confirmation(*conn*, *pdu*)
This is a long line of text.

class bsllservice.**ProxyServerService**(*ServiceAdapter*)
This is a long line of text.

add_connection(*conn*)
This is a long line of text.

remove_connection(*conn*)
This is a long line of text.

service_confirmation(*conn*, *bslpdu*)
This is a long line of text.

class bsllservice.**ProxyClientService**(*ServiceAdapter*)
This is a long line of text.

__init__(*self*, *mux*, *addr=None*, *userinfo=None*)

Parameters

- **mux** –
- **addr** –
- **userinfo** –

This is a long line of text.

get_default_user_info(*addr*)
This is a long line of text.

connect(*addr=None*, *userinfo=None*)
This is a long line of text.

connect_ack(*conn*, *bslpdu*)
This is a long line of text.

service_confirmation(*conn*, *bslpdu*)
This is a long line of text.

confirmation(*pdu*)
This is a long line of text.

LAN Emulation Service

To be developed.

7.1.4 Network Layer

Network Layer Protocol Data Units

This is a long line of text.

PDU Base Types

class npdu.**NPCI**(*PCI*)
This is a long line of text.

npduVersion
This is a long line of text.

npduControl
This is a long line of text.

npduDADR
This is a long line of text.

npduSADR
This is a long line of text.

npduHopCount
This is a long line of text.

npduNetMessage
This is a long line of text.

npduVendorID
This is a long line of text.

update(*npci*)
This is a long line of text.

encode(*pdu*)
decode(*pdu*)

Parameters **pdu** – pdu.PDUData buffer

This is a long line of text.

class npdu.**NPDU**(*NPCI*, *PDUData*)
This is a long line of text.

encode(*pdu*)
decode(*pdu*)

Parameters **pdu** – pdu.PDUData buffer

This is a long line of text.

Service Requests

class npdu.**WhoIsRouterToNetwork**(*NPCI*)
This is a long line of text.

encode(*npdu*)
decode(*npdu*)

Parameters **pdu** – *NPDU* buffer

This is a long line of text.

class npdu.**IAmRouterToNetwork**(*NPCI*)
This is a long line of text.

encode(*npdu*)
decode(*npdu*)

Parameters **pdu** – *NPDU* buffer

This is a long line of text.

class npdu.**ICouldBeRouterToNetwork**(*NPCI*)
This is a long line of text.

encode(*npdu*)
decode(*npdu*)

Parameters **pdu** – *NPDU* buffer

This is a long line of text.

class npdu.**RejectMessageToNetwork**(*NPCI*)
This is a long line of text.

encode(*npdu*)
decode(*npdu*)

Parameters **pdu** – *NPDU* buffer

This is a long line of text.

class npdu.**RouterBusyToNetwork**(*NPCI*)
This is a long line of text.

encode(*npdu*)
decode(*npdu*)

Parameters **pdu** – *NPDU* buffer

This is a long line of text.

class npdu.**RouterAvailableToNetwork**(*NPCI*)
This is a long line of text.

encode(*npdu*)
decode(*npdu*)

Parameters **pdu** – *NPDU* buffer

This is a long line of text.

class npdu.**RoutingTableEntry**
This is a long line of text.

rtDNET
This is a long line of text.

rtPortID
This is a long line of text.

rtPortInfo
This is a long line of text.

class npdu.**InitializeRoutingTable**(*NPCI*)
This is a long line of text.

encode(*npdu*)
decode(*npdu*)

Parameters **pdu** – *NPDU* buffer

This is a long line of text.

class npdu.**InitializeRoutingTableAck**(*NPCI*)
This is a long line of text.

encode(*npdu*)
decode(*npdu*)

Parameters **pdu** – *NPDU* buffer

This is a long line of text.

class npdu.**EstablishConnectionToNetwork**(*NPCI*)
This is a long line of text.

encode(*npdu*)
decode(*npdu*)

Parameters **pdu** – *NPDU* buffer

This is a long line of text.

class npdu.**DisconnectConnectionToNetwork**(*NPCI*)
This is a long line of text.

encode(*npdu*)
decode(*npdu*)

Parameters **pdu** – *NPDU* buffer

This is a long line of text.

Network Layer Service

BACnet network layer. . .

Connection State

Every thing is connected and every connection has a state.

- ROUTER_AVAILABLE - normal
- ROUTER_BUSY - router is busy

- ROUTER_DISCONNECTED - could make a connection, but hasn't
- ROUTER_UNREACHABLE - cannot route

This is a long line of text.

Reference Structures

This is a long line of text.

class netservice.**NetworkReference**
This is a long line of text.

network
This is a long line of text.

router
This is a long line of text.

status
This is a long line of text.

class netservice.**RouterReference**
This is a long line of text.

adapter
This is a long line of text.

address
This is a long line of text.

networks
This is a long line of text.

status
This is a long line of text.

Network Service

This is a long line of text.

class netservice.**NetworkServiceElement**(*ApplicationServiceElement*)
This is a long line of text.

indication(*adapter*, *npdu*)

Parameters

- **adapter** –
- **npdu** –

This is a long line of text.

confirmation(*adapter*, *npdu*)

Parameters

- **adapter** –
- **npdu** –

This is a long line of text.

WhoIsRouterToNetwork (*adapter*, *npdu*)
This is a long line of text.

IAmRouterToNetwork (*adapter*, *npdu*)
This is a long line of text.

ICouldBeRouterToNetwork (*adapter*, *npdu*)
This is a long line of text.

RejectMessageToNetwork (*adapter*, *npdu*)
This is a long line of text.

RouterBusyToNetwork (*adapter*, *npdu*)
This is a long line of text.

RouterAvailableToNetwork (*adapter*, *npdu*)
This is a long line of text.

InitializeRoutingTable (*adapter*, *npdu*)
This is a long line of text.

InitializeRoutingTableAck (*adapter*, *npdu*)
This is a long line of text.

EstablishConnectionToNetwork (*adapter*, *npdu*)
This is a long line of text.

DisconnectConnectionToNetwork (*adapter*, *npdu*)
This is a long line of text.

Virtual LAN

This is a long line of text.

class vlan.**Network**
This is a long line of text.

nodes
This is a long line of text.

dropPercent
This is a long line of text.

addrLen
This is a long line of text.

addrAddr
This is a long line of text.

__init__ (*addr*, *dropPercent=0.0*)

Parameters **dropPercent** (*float*) – percentage of packets to drop

This is a long line of text.

add_node (*node*)

Parameters **node** (Node) – node to add to the network

This is a long line of text.

remove_node (*node*)

Parameters **node** (Node) – node to remove from the network

This is a long line of text.

process_pdu(*pdu*)

Parameters **pdu** – pdu to send on the network

This is a long line of text.

__len__()
Simple mechanism to return the number of nodes on the network.

class vlan.**Node**
This is a long line of text.

__init__(*addr*, *lan=None*, *promiscuous=False*, *spoofing=False*)

Parameters

- **addr** (Address) – address for the node
- **lan** (Network) – network reference
- **promiscuous** (*boolean*) – receive all packets
- **spoofing** (*boolean*) – send with mocked source address

This is a long line of text.

bind(*lan*)

Parameters **lan** (Network) – network reference

This is a long line of text.

indication(*pdu*)

Parameters **pdu** – pdu to send on the network

This is a long line of text.

7.1.5 Application Layer

Primative Data

This is a long line of text.

Tags

This is a long line of text.

class primitivedata.**Tag**
This is a long line of text.

tagClass
This is a long line of text.

tagNumber
This is a long line of text.

tagLVT
This is a long line of text.

tagData
This is a long line of text.

_app_tag_name
This is a long line of text.

_app_tag_class
This is a long line of text.

__init__(**args*)
This is a long line of text.

set(*tclass*, *tnum*, *tlvt=0*, *tdata=''*)
This is a long line of text.

set_app_data(*tnum*, *tdata*)
This is a long line of text.

encode(*pdu*)
decode(*pdu*)
This is a long line of text.

app_to_context(*context*)
context_to_app(*dataType*)
This is a long line of text.

app_to_object()
This is a long line of text.

__repr__()
This is a long line of text.

__eq__(*tag*)
__ne__(*tag*)
This is a long line of text.

debug_contents(*indent=1*, *file=sys.stdout*, *_ids=None*)
This is a long line of text.

class primitivedata.**ApplicationTag**(*Tag*)
This is a long line of text.

class primitivedata.**ContextTag**(*Tag*)
This is a long line of text.

class primitivedata.**OpeningTag**(*Tag*)
This is a long line of text.

class primitivedata.**ClosingTag**(*Tag*)
This is a long line of text.

class primitivedata.**TagList**
This is a long line of text.

Atomic Data Types

This is a long line of text.

class primitivedata.**Atomic**
This is a long line of text.

__cmp__(*other*)

Parameters **other** – reference to some other atomic data type object

This is a long line of text.

class primitivedata.**Null**(*Atomic*)
This is a long line of text.

encode(*tag*)
decode(*tag*)

Parameters **tag** – *Tag* reference

This is a long line of text.

class primitivedata.**Boolean**(*Atomic*)
This is a long line of text.

encode(*tag*)
decode(*tag*)

Parameters **tag** – *Tag* reference

This is a long line of text.

class primitivedata.**Unsigned**(*Atomic*)
This is a long line of text.

encode(*tag*)
decode(*tag*)

Parameters **tag** – *Tag* reference

This is a long line of text.

class primitivedata.**Integer**(*Atomic*)
This is a long line of text.

encode(*tag*)
decode(*tag*)

Parameters **tag** – *Tag* reference

This is a long line of text.

class primitivedata.**Real**(*Atomic*)
This is a long line of text.

encode(*tag*)
decode(*tag*)

Parameters **tag** – *Tag* reference

This is a long line of text.

class primitivedata.**Double**(*Atomic*)
This is a long line of text.

encode(*tag*)
decode(*tag*)

Parameters **tag** – *Tag* reference

This is a long line of text.

class primitivedata.**OctetString**(*Atomic*)
This is a long line of text.

encode(*tag*)
decode(*tag*)

Parameters **tag** – *Tag* reference

This is a long line of text.

class primitivedata.**CharacterString**(*Atomic*)
This is a long line of text.

encode(*tag*)
decode(*tag*)

Parameters **tag** – *Tag* reference

This is a long line of text.

class primitivedata.**BitString**(*Atomic*)
This is a long line of text.

encode(*tag*)
decode(*tag*)

Parameters **tag** – *Tag* reference

This is a long line of text.

__getitem__(*bit*)
This is a long line of text.

__setitem__(*bit*, *value*)
This is a long line of text.

class primitivedata.**Enumerated**(*Atomic*)
This is a long line of text.

enumerations
This is a long line of text.

_xlate_table
This is a long line of text.

__getitem__(*item*)
This is a long line of text.

get_long()
This is a long line of text.

keylist()
This is a long line of text.

__cmp__(*other*)
This is a long line of text.

encode(*tag*)
decode(*tag*)

Parameters **tag** – *Tag* reference

This is a long line of text.

class primitivedata.**Date**(*Atomic*)
This is a long line of text.

__init__(*arg=None*, *year=255*, *month=255*, *day=255*, *dayOfWeek=255*)

Parameters

- **arg** –
- **year** –
- **month** –
- **day** –
- **dayOfWeek** –

This is a long line of text.

now()
This is a long line of text.

CalcDayOfWeek()
This is a long line of text.

encode(*tag*)
decode(*tag*)

Parameters **tag** – *Tag* reference

This is a long line of text.

class primitivedata.**Time**(*Atomic*)
This is a long line of text.

__init__(*arg=None*, *hour=255*, *minute=255*, *second=255*, *hundredth=255*)

Parameters

- **arg** –
- **hour** –
- **minute** –
- **second** –
- **hundredth** –

This is a long line of text.

now()
This is a long line of text.

encode(*tag*)
decode(*tag*)

Parameters **tag** – *Tag* reference

This is a long line of text.

class primitivedata.**ObjectType**(*Enumerated*)
This is a long line of text.

class primitivedata.**ObjectIdentifier**(*Atomic*)
This is a long line of text.

objectTypeClass
This is a long line of text.

__init__(**args*)
This is a long line of text.

set_tuple(*objType*, *objInstance*)
get_tuple()

Parameters

- **objType** – *ObjectType* object type
- **objInstance** (*int*) – object instance

This is a long line of text.

set_long(*value*)
get_long()
This is a long line of text.

encode(*tag*)
decode(*tag*)

Parameters **tag** – *Tag* reference

This is a long line of text.

__hash__()
This is a long line of text.

__cmp__(*other*)
This is a long line of text.

Constructed Data

This is a long line of text.

Elements

This is a long line of text.

class constructeddata.**Element**

name
This is a long line of text.

klass
This is a long line of text.

context
This is a long line of text.

optional
This is a long line of text.

Sequences

This is a long line of text.

class constructeddata.**Sequence**

sequenceElements
This is a long line of text.

encode(*taglist*)
decode(*taglist*)

Parameters **taglist** – list of *primitivedata.Tag* objects

This is a long line of text.

debug_contents(*indent=1*, *file=sys.stdout*, *_ids=None*)
This is a long line of text.

class constructeddata.**SequenceOf**(*klass*)

append(*value*)
This is a long line of text.

__getitem__(*item*)

Parameters **item** – item number

This is a long line of text.

__len__()
This is a long line of text.

encode(*taglist*)
decode(*taglist*)

Parameters **taglist** – list of *primitivedata.Tag* objects

This is a long line of text.

debug_contents(*indent=1*, *file=sys.stdout*, *_ids=None*)
This is a long line of text.

Arrays

This is a long line of text.

class constructeddata.**Array**
This is a long line of text.

class constructeddata.**ArrayOf**(*klass*)
This is a long line of text.

append(*value*)
This is a long line of text.

__len__()
This is a long line of text.

__getitem__(*item*)

Parameters **item** – item number

This is a long line of text.

__setitem__(*item*, *value*)

Parameters

- **item** – item number
- **value** – new value for item

This is a long line of text.

__delitem__(*item*)

Parameters **item** – item number

This is a long line of text.

index(*value*)

Parameters **value** – new value for item

This is a long line of text.

encode(*taglist*)
decode(*taglist*)

Parameters **taglist** – list of *primitivedata.Tag* objects

This is a long line of text.

encode_item(*item*, *taglist*)
decode_item(*item*, *taglist*)

Parameters

- **item** – item number
- **taglist** – list of *primitivedata.Tag* objects

This is a long line of text.

debug_contents(*indent=1*, *file=sys.stdout*, *_ids=None*)
This is a long line of text.

Choice

This is a long line of text.

class constructeddata.**Choice**
This is a long line of text.

__init__(*self*, ***kwargs*)

Parameters **kwargs** – expected value to set choice

This is a long line of text.

encode(*taglist*)
decode(*taglist*)

Parameters **taglist** – list of *primitivedata.Tag* objects

This is a long line of text.

debug_contents(*indent=1*, *file=sys.stdout*, *_ids=None*)
This is a long line of text.

Any

This is a long line of text.

class constructeddata.**Any**
This is a long line of text.

tagList
This is a long line of text.

__init__(*self*, **args*)

Parameters **args** – initial values to cast in

This is a long line of text.

encode(*taglist*)
decode(*taglist*)

Parameters **taglist** – list of *primitivedata.Tag* objects

This is a long line of text.

cast_in(*element*)

Parameters **element** – value to cast in

This is a long line of text.

cast_out(*klass*)

Parameters **klass** – class reference to decode value

This is a long line of text.

debug_contents(*indent=1*, *file=sys.stdout*, *_ids=None*)
This is a long line of text.

Base Types

This is a long line of text.

Array

class basetypes.**ArrayOfObjectIdentifier**
This is a long line of text.

Bit Strings

class basetypes.**BACnetDaysOfWeek**(*BitString*)
This is a long line of text.

class basetypes.**BACnetEventTransitionBits**(*BitString*)
This is a long line of text.

class basetypes.**BACnetLimitEnable**(*BitString*)
This is a long line of text.

class basetypes.**BACnetObjectTypesSupported**(*BitString*)
This is a long line of text.

class basetypes.**BACnetResultFlags**(*BitString*)
This is a long line of text.

class basetypes.**BACnetServicesSupported**(*BitString*)
This is a long line of text.

class basetypes.**BACnetStatusFlags**(*BitString*)
This is a long line of text.

Enumerations

class basetypes.**BACnetAccumulatorStatus**(*Enumerated*)
This is a long line of text.

class basetypes.**BACnetAction**(*Enumerated*)
This is a long line of text.

class basetypes.**BACnetBinaryPV**(*Enumerated*)
This is a long line of text.

class basetypes.**BACnetDeviceStatus**(*Enumerated*)
This is a long line of text.

class basetypes.**BACnetEngineeringUnits**(*Enumerated*)
This is a long line of text.

class basetypes.**BACnetEventState**(*Enumerated*)
This is a long line of text.

class basetypes.**BACnetEventType**(*Enumerated*)
This is a long line of text.

class basetypes.**BACnetFileAccessMethod**(*Enumerated*)
This is a long line of text.

class basetypes.**BACnetLifeSafetyMode**(*Enumerated*)
This is a long line of text.

class basetypes.**BACnetProgramError**(*Enumerated*)
This is a long line of text.

class basetypes.**BACnetProgramRequest**(*Enumerated*)
This is a long line of text.

class basetypes.**BACnetProgramState**(*Enumerated*)
This is a long line of text.

class basetypes.**BACnetPropertyIdentifier**(*Enumerated*)
This is a long line of text.

class basetypes.**BACnetNotifyType**(*Enumerated*)
This is a long line of text.

class basetypes.**BACnetPolarity**(*Enumerated*)
This is a long line of text.

class basetypes.**BACnetPrescale**(*Sequence*)
This is a long line of text.

class basetypes.**BACnetReliability**(*Enumerated*)
This is a long line of text.

class basetypes.**BACnetSegmentation**(*Enumerated*)
This is a long line of text.

class basetypes.**BACnetVTClass**(*Enumerated*)
This is a long line of text.

class basetypes.**BACnetNodeType**(*Enumerated*)
This is a long line of text.

Structures

class basetypes.**BACnetActionCommand**(*Sequence*)
This is a long line of text.

class basetypes.**BACnetActionList**(*Sequence*)
This is a long line of text.

class basetypes.**BACnetAddress**(*Sequence*)
This is a long line of text.

class basetypes.**BACnetAddressBinding**(*Sequence*)
This is a long line of text.

class basetypes.**BACnetDateRange**(*Sequence*)
This is a long line of text.

class basetypes.**BACnetWeekNDay**(*OctetString*)
This is a long line of text.

class basetypes.**BACnetCalendarEntry**(*Choice*)
This is a long line of text.

class basetypes.**BACnetScale**(*Choice*)
This is a long line of text.

class basetypes.**BACnetTimeValue**(*Sequence*)
This is a long line of text.

class basetypes.**BACnetDailySchedule**(*Sequence*)
This is a long line of text.

class basetypes.**BACnetDateTime**(*Sequence*)
This is a long line of text.

class basetypes.**BACnetRecipient**(*Choice*)
This is a long line of text.

class basetypes.**BACnetDestination**(*Sequence*)
This is a long line of text.

class basetypes.**BACnetPropertyStates**(*Choice*)
This is a long line of text.

class basetypes.**NotificationChangeOfBitstring**(*Sequence*)
This is a long line of text.

class basetypes.**NotificationChangeOfState**(*Sequence*)
This is a long line of text.

class basetypes.**NotificationChangeOfValueNewValue**(*Choice*)
This is a long line of text.

class basetypes.**NotificationChangeOfValue**(*Sequence*)
This is a long line of text.

class basetypes.**NotificationCommandFailure**(*Sequence*)
This is a long line of text.

class basetypes.**NotificationFloatingLimit**(*Sequence*)
This is a long line of text.

class basetypes.**NotificationOutOfRange**(*Sequence*)
This is a long line of text.

class basetypes.**NotificationComplexEventType**(*Any*)
This is a long line of text.

class basetypes.**NotificationChangeOfLifeSafety**(*Any*)
This is a long line of text.

class basetypes.**NotificationExtended**(*Any*)
This is a long line of text.

class basetypes.**NotificationBufferReady**(*Any*)
This is a long line of text.

class basetypes.**NotificationUnsignedRange**(*Any*)
This is a long line of text.

class basetypes.**BACnetNotificationParameters**(*Choice*)
This is a long line of text.

class basetypes.**BACnetObjectPropertyReference**(*Sequence*)
This is a long line of text.

class basetypes.**BACnetObjectPropertyValue**(*Sequence*)
This is a long line of text.

class basetypes.**BACnetObjectType**(*ObjectType*)
This is a long line of text.

class basetypes.**BACnetPriorityValue**(*Choice*)
This is a long line of text.

class basetypes.**BACnetPriorityArray**
Implemented as ArrayOf(BACnetPriorityValue)

class basetypes.**BACnetPropertyReference**(*Sequence*)
This is a long line of text.

class basetypes.**BACnetPropertyValue**(*Sequence*)
This is a long line of text.

class basetypes.**BACnetRecipientProcess**(*Sequence*)
This is a long line of text.

class basetypes.**BACnetSessionKey**(*Sequence*)
This is a long line of text.

class basetypes.**BACnetSetpointReference**(*Sequence*)
This is a long line of text.

class basetypes.**BACnetSpecialEvent**(*Sequence*)
This is a long line of text.

class basetypes.**BACnetTimeStamp**(*Choice*)
This is a long line of text.

class basetypes.**BACnetVTSession**(*Sequence*)
This is a long line of text.

class basetypes.**BACnetDeviceObjectReference**(*Sequence*)
This is a long line of text.

Application Layer PDUs

This is a long line of text.

Globals

apdu.**apdu_types**
This is a long line of text.

apdu.**confirmed_request_types**
This is a long line of text.

apdu.**complex_ack_types**
This is a long line of text.

apdu.**unconfirmed_request_types**
This is a long line of text.

apdu.**error_types**
This is a long line of text.

Functions

apdu.**register_apdu_type**(*klass*)
This is a long line of text.

apdu.**register_confirmed_request_type**(*klass*)
This is a long line of text.

apdu.**register_complex_ack_type**(*klass*)
This is a long line of text.

apdu.**register_unconfirmed_request_type**(*klass*)
This is a long line of text.

apdu.**register_error_type**(*klass*)
This is a long line of text.

apdu.**encode_max_apdu_segments**(*arg*)

apdu.**decode_max_apdu_segments**(*arg*)
This is a long line of text.

apdu.**encode_max_apdu_response**(*arg*)

apdu.**decode_max_apdu_response**(*arg*)
This is a long line of text.

PDU Base Types

This is a long line of text.

class apdu.**APCI**(*PCI*)

apduType

apduSeg

apduMor

apduSA

apduSrv

apduNak

apduSeq

apduWin

apduMaxSegs

apduMaxResp

apduService

apduInvokeID

apduAbortRejectReason

This is a long line of text.

update(*apci*)

Parameters **apci** – source data to copy

This is a long line of text.

encode(*pdu*)
decode(*pdu*)

Parameters **pdu** – pdu.PDUData buffer

This is a long line of text.

class apdu.**APDU**(*APCI*, *PDUData*)

This is a long line of text.

encode(*pdu*)
decode(*pdu*)

Parameters **pdu** – pdu.PDUData buffer

This is a long line of text.

class apdu.**_APDU**(*APDU*)

This is a long line of text.

encode(*pdu*)
decode(*pdu*)

Parameters **pdu** – pdu.PDUData buffer

This is a long line of text.

set_context(*context*)

Parameters **context** – *APDU* reference

Basic Classes

This is a long line of text.

class apdu.**ConfirmedRequestPDU**(*_APDU*)
This is a long line of text.

class apdu.**ConfirmedRequestPDU**(*_APDU*)
This is a long line of text.

class apdu.**UnconfirmedRequestPDU**(*_APDU*)
This is a long line of text.

class apdu.**SimpleAckPDU**(*_APDU*)
This is a long line of text.

class apdu.**ComplexAckPDU**(*_APDU*)
This is a long line of text.

class apdu.**SegmentAckPDU**(*_APDU*)
This is a long line of text.

class apdu.**ErrorPDU**(*_APDU*)
This is a long line of text.

class apdu.**RejectPDU**(*_APDU*)
This is a long line of text.

class apdu.**SimpleAckPDU**(*_APDU*)
This is a long line of text.

Sequence Classes

This is a long line of text.

class apdu.**APCISequence**(*APCI*, *Sequence*)
This is a long line of text.

class apdu.**ConfirmedRequestSequence**(*APCISequence*, *ConfirmedRequestPDU*)
This is a long line of text.

class apdu.**ComplexAckSequence**(*APCISequence*, *ComplexAckPDU*)
This is a long line of text.

class apdu.**UnconfirmedRequestSequence**(*APCISequence*, *UnconfirmedRequestPDU*)
This is a long line of text.

class apdu.**ErrorSequence**(*APCISequence*, *ErrorPDU*)
This is a long line of text.

Errors

This is a long line of text.

class apdu.**ErrorClass**(*Enumerated*)
This is a long line of text.

class apdu.**ErrorCode**(*Enumerated*)
This is a long line of text.

class apdu.**ErrorType**(*Sequence*)
This is a long line of text.

class apdu.**Error**(*ErrorSequence*, *ErrorType*)
This is a long line of text.

Who-Is/I-Am

This is a long line of text.

class apdu.**WhoIsRequest**(*UnconfirmedRequestSequence*)
This is a long line of text.

class apdu.**IAmRequest**(*UnconfirmedRequestSequence*)
This is a long line of text.

Who-Has/I-Have

This is a long line of text.

class apdu.**WhoHasRequest**(*UnconfirmedRequestSequence*)
This is a long line of text.

class apdu.**WhoHasLimits**(*Sequence*)
This is a long line of text.

class apdu.**WhoHasObject**(*Choice*)
This is a long line of text.

This is a long line of text.

class apdu.**IHaveRequest**(*UnconfirmedRequestSequence*)
This is a long line of text.

Read-Property

This is a long line of text.

class apdu.**ReadPropertyRequest**(*ConfirmedRequestSequence*)
This is a long line of text.

class apdu.**ReadPropertyACK**(*ComplexAckSequence*)
This is a long line of text.

Write-Property

This is a long line of text.

class apdu.**WritePropertyRequest**(*ConfirmedRequestSequence*)
This is a long line of text.

Read-Property-Multiple

This is a long line of text.

class apdu.**ReadPropertyMultipleRequest**(*ConfirmedRequestSequence*)
This is a long line of text.

class apdu.**ReadAccessSpecification**(*Sequence*)
This is a long line of text.

class apdu.**ReadPropertyMultipleACK**(*ComplexAckSequence*)
This is a long line of text.

class apdu.**ReadAccessResult**(*Sequence*)
This is a long line of text.

class apdu.**ReadAccessResultElement**(*Sequence*)
This is a long line of text.

class apdu.**ReadAccessResultElementChoice**(*Choice*)
This is a long line of text.

Write-Property-Multiple

This is a long line of text.

class apdu.**WritePropertyMultipleRequest**(*ConfirmedRequestSequence*)
This is a long line of text.

class apdu.**WriteAccessSpecification**(*Sequence*)
This is a long line of text.

class apdu.**WritePropertyMultipleError**(*ErrorSequence*)
This is a long line of text.

Read-Range

This is a long line of text.

class apdu.**ReadRangeRequest**(*ConfirmedRequestSequence*)
This is a long line of text.

class apdu.**Range**(*Choice*)
This is a long line of text.

class apdu.**RangeByPosition**(*Sequence*)
This is a long line of text.

class apdu.**RangeBySequenceNumber**(*Sequence*)
This is a long line of text.

class apdu.**RangeByTime**(*Sequence*)
This is a long line of text.

class apdu.**ReadRangeACK**(*ComplexAckSequence*)
This is a long line of text.

Event-Notification

This is a long line of text.

class apdu.**ConfirmedEventNotificationRequest**(*ConfirmedRequestSequence*)
This is a long line of text.

class apdu.**UnconfirmedEventNotificationRequest**(*Sequence*)
This is a long line of text.

Change-Of-Value-Notification

This is a long line of text.

class apdu.**UnconfirmedCOVNotificationRequest**(*UnconfirmedRequestSequence*)
This is a long line of text.

Other Errors

This is a long line of text.

class apdu.**ChangeListError**(*ErrorSequence*)
This is a long line of text.

class apdu.**CreateObjectError**(*ErrorSequence*)
This is a long line of text.

class apdu.**ConfirmedPrivateTransferError**(*ErrorSequence*)
This is a long line of text.

class apdu.**VTCloseError**(*ErrorSequence*)
This is a long line of text.

Objects

BACnet virtual link layer...

Globals

This is a long line of text.

object.**map_name_re**
This is a long line of text.

object.**object_types**
This is a long line of text.

Functions

This is a long line of text.

object.**map_name**(*name*)

Parameters **name** (*string*) – something

This is a long line of text.

object.**register_object_type**(*klass*)

Parameters **klass** – class to register

This is a long line of text.

object.**get_object_class**(*objectType*)

Parameters **objectType** – something

Returns something

This is a long line of text.

object.**get_datatype**(*objectType*, *property*)

Parameters

- **objectType** – something
- **property** – something

Returns datatype class

This is a long line of text.

Properties

This is a long line of text.

class object.**Property**

This is a long line of text.

identifier

This is a long line of text.

datatype

This is a long line of text.

optional

This is a long line of text.

mutable

This is a long line of text.

default

This is a long line of text.

ReadProperty(*obj*, *arrayIndex=None*)

Parameters

- **obj** – object reference
- **arrayIndex** – optional array index

This is a long line of text.

WriteProperty(*obj*, *value*, *arrayIndex=None*, *priority=None*)

Parameters

- **obj** – object reference
- **value** – new property value

- **arrayIndex** – optional array index
- **priority** – optional priority

This is a long line of text.

class object.**ObjectIdentifierProperty**

WriteProperty(*obj*, *value*, *arrayIndex=None*, *priority=None*)

Parameters

- **obj** – object reference
- **value** – new property value
- **arrayIndex** – optional array index
- **priority** – optional priority

This is a long line of text.

class object.**CurrentDateProperty**

ReadProperty(*obj*, *arrayIndex=None*)

Parameters

- **obj** – object reference
- **arrayIndex** – optional array index

This is a long line of text.

WriteProperty(*obj*, *value*, *arrayIndex=None*, *priority=None*)
This method is to override the *Property.WriteProperty()* so instances of this class will raise an expection and be considered unwriteable.

class object.**CurrentTimeProperty**

ReadProperty(*obj*, *arrayIndex=None*)

Parameters

- **obj** – object reference
- **arrayIndex** – optional array index

This is a long line of text.

WriteProperty(*obj*, *value*, *arrayIndex=None*, *priority=None*)
This method is to override the *Property.WriteProperty()* so instances of this class will raise an expection and be considered unwriteable.

Objects

This is a long line of text.

Standard Object Types

This is a long line of text.

class object.**AccumulatorObject**(*Object*)

class object.**BACnetAccumulatorRecord**(*Sequence*)

class object.**AnalogInputObject**(*Object*)

class object.**AnalogOutputObject**(*Object*)

class object.**AnalogValueObject**(*Object*)

class object.**AveragingObject**(*Object*)

class object.**BinaryInputObject**(*Object*)

class object.**BinaryOutputObject**(*Object*)

class object.**BinaryValueObject**(*Object*)

class object.**CalendarObject**(*Object*)

class object.**CommandObject**(*Object*)

class object.**DeviceObject**(*Object*)

class object.**EventEnrollmentObject**(*Object*)

class object.**FileObject**(*Object*)

class object.**GroupObject**(*Object*)

class object.**LifeSafetyPointObject**(*Object*)

class object.**LifeSafetyZoneObject**(*Object*)

class object.**LoopObject**(*Object*)

class object.**MultiStateInputObject**(*Object*)

class object.**MultiStateOutputObject**(*Object*)

class object.**MultiStateValueObject**(*Object*)

class object.**NotificationClassObject**(*Object*)

class object.**ProgramObject**(*Object*)

class object.**PulseConverterObject**(*Object*)

class object.**ScheduleObject**(*Object*)

class object.**StructuredViewObject**(*Object*)

class object.**TrendLogObject**(*Object*)

Extended Object Types

class object.**LocalDeviceObject**(*DeviceObject*)

Application

This is a long line of text.

Device Information

The device information objects and associated cache are used to assist with the following:

- Device-address-binding, the close associate between the device identifier for a device and its network address
- Construction of confirmed services to determine if a device can accept segmented requests and/or responses and the maximum size of an APDU
- The vendor of the device to know what additional vendor specific objects, properties, and other datatypes are available

class app.**DeviceInfo**

This is a long line of text.

deviceIdentifier
The device instance number associated with the device.

address
The *pdu.LocalStation* or *pdu.RemoteStation* associated with the device.

maxApduLengthAccepted
The maximum APDU length acccepted, which has the same value as the property of the *object.DeviceObject* of the device. This is typically initialized with the parameter with the same name from the *apdu.IAmRequest*.

segmentationSupported
The enumeration value basetypes.Segmentation that describes the segmentation supported by the device; sending, receiving, both, or no segmentation supported.

vendorID
The vendor identifier of the device.

maxNpduLength
The maximum length of an NPDU permitted by the links used by the local, remote, and intervening networks.

maxSegmentsAccepted
The maximum number of segments of an APDU that this device will accept.

__init__()
Initialize a *DeviceInfo* object using the default values that are typical for BACnet devices.

class app.**DeviceInfoCache**

An instance of this class is used to manage the cache of device information on behalf of the application. The information may come from interrogating the device as it presents itself on the network or from a database, or some combination of the two.

The default implementation is to only use information from the network and provide some reasonable defaults when information isn't available. The *Application* is provided a reference to an instance of this class or a derived class, and multiple application instances may share a cache, if that's appropriate.

cache
This is a private dictionary for use by the class or derived class methods. The default implementation uses a mix of device identifiers, addresses, or both to reference *DeviceInfo* objects.

has_device_info(*key*)

Parameters **key** – a device object identifier, a *pdu.LocalStation* or a RemoteStation address.

Return true if there is a *DeviceInfo* instance in the cache.

add_device_info(*apdu*)

Parameters **apdu** (IAmRequest) – an IAmRequest

This function is called by an application when it receives an *apdu.IAmRequest* and it wants to cache the information. For example the application had issued a *apdu.WhoIsRequest* for a device and this is the corresponding *apdu.IAmRequest*.

get_device_info(*key*)

Parameters **key** – a device object identifier, a *pdu.LocalStation* or a RemoteStation address.

Return the *DeviceInfo* instance in the cache associated with the key, or *None* if it does not exist.

update_device_info(*info*)

Parameters **info** (DeviceInfo) – the updated device information

This function is called by the application service layer when the device information has changed as a result of comparing it with incoming requests. This function is overriden when the application has additional work, such as updating a database.

release_device_info(*info*)

Parameters **info** (DeviceInfo) – device information no longer being used

This function is called by the application service layer when there are no more confirmed requests associated with the device and the *DeviceInfo* can be removed from the cache. This function is overridden by a derived class to change the cache behaviour, for example perhaps the objects are removed from the cache until some timer expires.

Base Class

This is a long line of text.

class app.**Application**(*ApplicationServiceElement*)

This is a long line of text.

__init__(*localDevice*, *localAddress*)

Parameters

- **localDevice** (DeviceObject) – the local device object
- **localAddress** (Address) – the local address
- **actorClass** – the initial source value

This is a long line of text.

snork(*address=None*, *segmentationSupported='no-segmentation'*, *maxApduLengthAccepted=1024*, *maxSegmentsAccepted=None*)

Parameters

- **localAddress** (Address) – the local address
- **segmentationSupported** – enumeration *basetypes.BACnetSegmentation*
- **maxApduLengthAccepted** – maximum APDU length
- **maxSegmentsAccepted** – segmentation parameter

This is a long line of text.

add_object(*obj*)

Parameters **obj** – the initial source value

This is a long line of text.

delete_object(*obj*)

Parameters **obj** – the initial source value

This is a long line of text.

get_object_id(*objid*)

Parameters **obj** – the initial source value

This is a long line of text.

get_object_name(*objname*)

Parameters **objname** – address to establish a connection

iter_objects()

Parameters **address** – address to disconnect

indication(*apdu*)

Parameters **apdu** – application layer PDU

This is a long line of text.

do_WhoIsRequest(*apdu*)

Parameters **apdu** – Who-Is request, *apdu.WhoIsRequest*

This is a long line of text.

do_IAmRequest(*apdu*)

Parameters **apdu** – I-Am request, *apdu.IAmRequest*

This is a long line of text.

do_ReadPropertyRequest(*apdu*)

Parameters **apdu** – Read-Property request, *apdu.ReadPropertyRequest*

This is a long line of text.

do_WritePropertyRequest(*apdu*)

Parameters **apdu** – Write-Property request, *apdu.WritePropertyRequest*

This is a long line of text.

BACnet/IP Applications

This is a long line of text.

class app.**BIPSimpleApplication**(*Application*)

__init__(*localDevice*, *localAddress*)

Parameters

- **localDevice** – This is a long line of text.

- **localAddress** – This is a long line of text.

This is a long line of text.

class app.**BIPForeignApplication**(*Application*)

__init__(*localDevice*, *localAddress*, *bbmdAddress*, *bbmdTTL*)

Parameters

- **localDevice** – This is a long line of text.
- **localAddress** – This is a long line of text.
- **bbmdAddress** – This is a long line of text.
- **bbmdTTL** – This is a long line of text.

This is a long line of text.

BACnet/IP Network Application

This is a long line of text.

class app.**BIPNetworkApplication**(*NetworkServiceElement*)

__init__(*localAddress*)

Parameters **localAddress** – This is a long line of text.

This is a long line of text.

Application Service

This is a long line of text.

Segmentation State Machine

This is a long line of text.

class appservice.**SSM**(*OneShotTask*)

This is a long line of text.

remoteDevice

This is a long line of text.

invokeID

This is a long line of text.

state

This is a long line of text.

segmentAPDU

This is a long line of text.

segmentSize

This is a long line of text.

segmentCount
This is a long line of text.

maxSegmentsAccepted
This is a long line of text.

retryCount
This is a long line of text.

segmentRetryCount
This is a long line of text.

sentAllSegments
This is a long line of text.

lastSequenceNumber
This is a long line of text.

initialSequenceNumber
This is a long line of text.

actualWindowSize
This is a long line of text.

proposedWindowSize
This is a long line of text.

__init__(*sap*)

Parameters **sap** – service access point reference

This is a long line of text.

start_timer(*msecs*)

Parameters **msecs** – milliseconds

This is a long line of text.

stop_timer()
This is a long line of text.

restart_timer(*msecs*)

Parameters **msecs** – milliseconds

This is a long line of text.

set_state(*newState*, *timer=0*)

Parameters

- **newState** – new state
- **timer** – timer value

set_segmentation_context(*apdu*)

Parameters **apdu** – application PDU

get_segment(*indx*)

Parameters **apdu** – application layer PDU

This is a long line of text.

append_segment(*apdu*)

Parameters **apdu** – application PDU

This is a long line of text.

in_window(*seqA*, *seqB*)

Parameters

- **seqA** (*int*) – latest sequence number
- **seqB** (*int*) – initial sequence number

This is a long line of text.

FillWindow(*self*, *seqNum*)

Parameters **seqNum** (*int*) – initial sequence number

This is a long line of text.

Client Segmentation State Machine

This is a long line of text.

Server Segmentation State Machine

This is a long line of text.

Application Stack

This is a long line of text.

class appservice.**StateMachineAccessPoint**(*DeviceInfo*, *Client*, *ServiceAccessPoint*)

This is a long line of text.

class appservice.**ApplicationServiceAccessPoint**(*ApplicationServiceElement*, *ServiceAccessPoint*)

This is a long line of text.

7.1.6 Services

Service Modules

Device Services

class WhoIsIAmServices(*Capability*)

This class provides the capability to initiate and respond to device-address-binding PDUs.

do_WhoIsRequest(*apdu*)

Parameters **apdu** (WhoIsRequest) – Who-Is Request from the network

See Clause 16.10.1 for the parameters to this service.

do_IAmRequest(*apdu*)

Parameters **apdu** (IAmRequest) – I-Am Request from the network

See Clause 16.10.3 for the parameters to this service.

who_is (*self*, *low_limit=None*, *high_limit=None*, *address=None*)

Parameters

- **low_limit** (Unsigned) – optional low limit
- **high_limit** (Unsigned) – optional high limit
- **address** (Address) – optional destination, defaults to a global broadcast

This is a utility function that makes it simpler to generate a *WhoIsRequest*.

i_am (*self*, *address=None*)

Parameters **address** (Address) – optional destination, defaults to a global broadcast

This is a utility function that makes it simpler to generate an *IAmRequest* with the contents of the local device object.

class WhoHasIHaveServices (*Capability*)

This class provides the capability to initiate and respond to device and object binding PDU's.

do_WhoHasRequest (*apdu*)

Parameters **apdu** (WhoHasRequest) – Who-Has Request from the network

See Clause 16.9.1 for the parameters to this service.

do_IHaveRequest (*apdu*)

Parameters **apdu** (IHaveRequest) – I-Have Request from the network

See Clause 16.9.3 for the parameters to this service.

who_has (*thing*, *address=None*)

Parameters

- **thing** – object identifier or object name
- **address** (Address) – optional destination, defaults to a global broadcast

Not implemented.

i_have (*thing*, *address=None*)

Parameters

- **thing** – object identifier or object name
- **address** (Address) – optional destination, defaults to a global broadcast

This is a utility function that makes it simpler to generate an *IHaveRequest* given an object.

Support Classes

There are a few support classes in this module that make it simpler to build the most common BACnet devices.

class CurrentDateProperty (*Property*)

This class is a specialized readonly property that always returns the current date as provided by the operating system.

ReadProperty (*self*, *obj*, *arrayIndex=None*)

Returns the current date as a 4-item tuple consistent with the Python implementation of the Date primitive value.

WriteProperty(*self*, *obj*, *value*, *arrayIndex=None*, *priority=None*)
Object instances of this class are readonly, so this method raises a *writeAccessDenied* error.

class CurrentTimeProperty(*Property*)
This class is a specialized readonly property that always returns the current local time as provided by the operating system.

ReadProperty(*self*, *obj*, *arrayIndex=None*)
Returns the current date as a 4-item tuple consistent with the Python implementation of the `Time` primitive value.

WriteProperty(*self*, *obj*, *value*, *arrayIndex=None*, *priority=None*)
Object instances of this class are readonly, so this method raises a *writeAccessDenied* error.

class LocalDeviceObject(*DeviceObject*)
The *LocalDeviceObject* is an implementation of a `DeviceObject` that provides default implementations for common properties and behaviors of a BACnet device. It has default values for communications properties, returning the local date and time, and the *objectList* property for presenting a list of the objects in the device.

Object Services

class ReadWritePropertyServices(*Capability*)
This class provides the capability to respond to ReadProperty and WriteProperty service, used by a client BACnet-user to request the value of one property of one BACnet Object.

do_ReadPropertyRequest(*apdu*)

Parameters **apdu** (`ReadPropertyRequest`) – request from the network

See Clause 15.5 for the parameters to this service.

do_WritePropertyRequest(*apdu*)

Parameters **apdu** (`WritePropertyRequest`) – request from the network

See Clause 15.9 for the parameters to this service.

class ReadWritePropertyMultipleServices(*Capability*)
This class provides the capability to respond to ReadPropertyMultiple and WritePropertyMultiple service, used by a client BACnet-user to request the values of one or more specified properties of one or more BACnet Objects.

do_ReadPropertyMultipleRequest(*apdu*)

Parameters **apdu** (`ReadPropertyRequest`) – request from the network

See Clause 15.7 for the parameters to this service.

do_WritePropertyMultipleRequest(*apdu*)

Parameters **apdu** (`WritePropertyMultipleRequest`) – request from the network

Not implemented.

Support Functions

read_property_to_any(obj, propertyIdentifier, propertyArrayIndex=None):

Parameters

- **obj** – object
- **propertyIdentifier** – property identifier
- **propertyArrayIndex** – optional array index

Called by *read_property_to_result_element* to build an appropriate *Any* result object from the supplied object given the property identifier and optional array index.

read_property_to_result_element(obj, propertyIdentifier, propertyArrayIndex=None):

Parameters

- **obj** – object
- **propertyIdentifier** – property identifier
- **propertyArrayIndex** – optional array index

Called by *do_ReadPropertyMultipleRequest* to build the result element components of a *ReadPropertyMultipleACK*.

File Services

class FileServices(*Capability*)

This class provides the capability to read from and write to file objects.

do_AtomicReadFileRequest(*apdu*)

Parameters **apdu** (*AtomicReadFileRequest*) – request from the network

This method looks for a local file object by the object identifier and and passes the request parameters to the implementation of the record or stream support class instances.

do_AtomicWriteFileRequest(*apdu*)

Parameters **apdu** (*AtomicWriteFileRequest*) – request from the network

This method looks for a local file object by the object identifier and and passes the request parameters to the implementation of the record or stream support class instances.

Support Classes

class LocalRecordAccessFileObject(*FileObject*)

This abstract class provides a simplified API for implementing a local record access file. A derived class must provide implementations of these methods for the object to be used by the *FileServices*.

__len__()

Return the length of the file in records.

read_record(*start_record*, *record_count*)

Parameters

- **start_record** (*int*) – starting record
- **record_count** (*int*) – number of records

Return a tuple (eof, record_data) where the *record_data* is an array of octet strings.

write_record(*start_record*, *record_count*, *record_data*)

Parameters

- **start_record** (*int*) – starting record
- **record_count** (*int*) – number of records
- **record_data** – array of octet strings

Update the file with the new records.

class LocalStreamAccessFileObject(*FileObject*)
This abstract class provides a simplified API for implementing a local stream access file. A derived class must provide implementations of these methods for the object to be used by the *FileServices*.

__len__()
Return the length of the file in octets.

read_stream(*start_position*, *octet_count*)

Parameters

- **start_position** (*int*) – starting position
- **octet_count** (*int*) – number of octets

Return a tuple (eof, record_data) where the *record_data* is an array of octet strings.

write_stream(*start_position*, *data*)

Parameters

- **start_position** (*int*) – starting position
- **data** – octet string

Update the file with the new records.

class FileServicesClient(*Capability*)
This class adds a set of functions to the application that provides a simplified client API for reading and writing to files. It is not currently implemented.

Change Detection and Reporting

Detect

This is a long line of text.

Classes

class detect.**DetectionMonitor**

algorithm

parameter

obj

prop

filter

__init__(*algorithm*, *parameter*, *obj*, *prop*, *filter=None*)
This is a long line of text.

property_change(*old_value*, *new_value*)
This is a long line of text.

class detect.**DetectionAlgorithm**

_monitors
This private attribute is a list of *DetectionMonitor* objects that associate this algorithm instance with objects and properties.

_triggered
This private attribute is *True* when there is a change in a parameter which causes the algorithm to schedule itself to execute. More than one parameter may change between the times that the algorithm can execute.

__init__()
Initialize a detection algorithm, which simply initializes the instance attributes.

bind(***kwargs*)

Parameters **kwargs** (*tuple*) – parameter to property mapping

Create a *DetectionMonitor* instance for each of the keyword arguments and point it back to this algorithm instance. The algorithm parameter matches the keyword parameter name and the parameter value is an (object, property_name) tuple.

unbind()
Delete the *DetectionMonitor* objects associated with this algorithm and remove them from the property changed call list(s).

execute()
This function is provided by a derived class which checks to see if something should happen when its parameters have changed. For example, maybe a change-of-value or event notification should be generated.

_execute()
This method is a special wrapper around the *execute()* function that sets the internal trigger flag. When the flag is set then the *execute()* function is already scheduled to run (via *deferred()*) and doesn't need to be scheduled again.

Decorators

detect.**monitor_filter**(*parameter*)

Parameters **parameter** (*string*) – name of parameter to filter

This decorator is used with class methods of an algorithm to determine if the new value for a propert of an object is significant enough to consider the associated parameter value changed. For example:

```
class SomeAlgorithm(DetectionAlgorithm):

    @monitor_filter('pValue')
    def value_changed(self, old_value, new_value):
        return new_value > old_value + 10
```

Assume that an instance of this algorithm is bound to the *presentValue* of an *AnalogValueObject*:

```
some_algorithm = SomeAlgorithm()
some_algorithm.bind(pValue = (avo, 'presentValue'))
```

The algorithm parameter *pValue* will only be considered changed when the present value of the analog value object has increased by more than 10 at once. If it slowly climbs by something less than 10, or declines at all, the algorithm will not execute.

Change of Value (COV) Services

class ChangeOfValueServices(*Capability*)
This class provides the capability of managing COV subscriptions and initiating COV notifications.

do_SubscribeCOVRequest(apdu):

Parameters **apdu** (*SubscribeCOVRequest*) – request from the network

This method processes the request by looking up the referenced object and attaching a COV detection algorithm object. Any changes the to referenced object properties (such as *presentValue* to *statusFlags*) will trigger the algorithm to run and initiate COV notifications as necessary.

add_subscription(*cov*)
This method adds a subscription to the internal dictionary of subscriptions indexed by the object reference. There can be multiple COV subscriptions for the same object.

cancel_subscription(*cov*)
This method removes a subscription from the internal dictionary of subscriptions. If all of the subscriptinos have been removed, for example they have all expired, then the detection "hook" into the object is removed.

cov_notification(*cov*, *request*)
This method is used to wrap a COV notification request in an IOCB wrapper, submitting it as an IO request. The following confirmation function will be called when it is complete.

cov_confirmation(*iocb*)
This method looks at the response that was given to the COV notification and dispatchs one of the following functions.

cov_ack(*cov*, *request*, *response*)
This method is called when the client has responded with a simple acknowledgement.

cov_error(*cov*, *request*, *response*)
This method is called when the client has responded with an error. Depending on the error, the COV subscription might be canceled.

cov_reject(*cov*, *request*, *response*)
This method is called when the client has responded with a reject. Depending on the error, the COV subscription might be canceled.

cov_abort(*cov*, *request*, *response*)
This method is called when the client has responded with an abort. Depending on the error, the COV subscription might be canceled.

Support Classes

class ActiveCOVSubscriptions(*Property*)
An instance of this property is added to the local device object. When the property is read it will return a list of COVSubscription objects.

class SubscriptionList

append(*cov*)

Parameters **cov** (Subscription) – additional subscription

remove(*cov*)

Parameters **cov** (Subscription) – subscription to remove

find(*client_addr*, *proc_id*, *obj_id*)

Parameters

- **client_addr** (Address) – client address
- **proc_id** (*int*) – client process identifier
- **obj_id** (ObjectIdentifier) – object identifier

This method finds a matching Subscription object where all three parameters match. It is used when a subscription request arrives it is used to determine if it should be renewed or canceled.

class Subscription(*OneShotTask*)

Instances of this class are active subscriptions with a lifetime. When the subscription is created it "installs" itself as a task for the end of its lifetime and when the process_task function is called the subscription is canceled.

__init__(*obj_ref*, *client_addr*, *proc_id*, *obj_id*, *confirmed*, *lifetime*)

Parameters

- **obj_ref** – reference to the object being monitored
- **client_addr** – address of the client
- **proc_id** – process id of the client
- **obj_id** – object identifier
- **confirmed** – issue confirmed notifications
- **lifetime** – subscription lifetime

cancel_subscription()

This method is called to cancel a subscription, it is called by process_task.

renew_subscription(*lifetime*)

Parameters **lifetime** (*int*) – seconds until expiration

This method is called to renew a subscription.

process_task()

Call when the lifetime of the subscription has run out.

class COVDetection(*DetectionAlgorithm*)

This is a base class for a series of COV detection algorithms. The derived classes provide a list of the properties that are being monitored for changes and a list of properties that are reported.

execute()

This method overrides the execute function of the detection algorithm.

send_cov_notifications()

This method sends out notifications to all of the subscriptions that are associated with the algorithm.

class GenericCriteria(*COVDetection*)

This is the simplest detection algorithm that monitors the present value and status flags of an object.

class COVIncrementCriteria(*COVDetection*)

This detection algorithm is used for those objects that have a COV increment property, such as Analog Value Objects, where the change in the present value needs to exceed some delta value.

class AccessDoorCriteria(*COVDetection*)
This detection algorithm is used for Access Door Objects.

class AccessPointCriteria(*COVDetection*)
This detection algorithm is used for Access Point Objects.

class CredentialDataInputCriteria(*COVDetection*)
This detection algorithm is used for Credential Data Input Objects.

class LoadControlCriteria(*COVDetection*)
This detection algorithm is used for Load Control Objects.

class PulseConverterCriteria(*COVDetection*)
This detection algorithm is used for Pulse Converter Objects.

7.1.7 Analysis

Analysis of PCAP Files

This is a long line of text.

Functions

analysis.**strftimestamp**(*ts*)

Parameters **ts** – timestamp

This is a long line of text.

Decoders

This is a long line of text.

analysis.**decode_ethernet**(*s*)

Parameters **s** – packet string

This is a long line of text.

analysis.**decode_vlan**(*s*)

Parameters **s** – packet string

This is a long line of text.

analysis.**decode_ip**(*s*)

Parameters **s** – packet string

This is a long line of text.

analysis.**decode_udp**(*s*)

Parameters **s** – packet string

This is a long line of text.

analysis.**decode_udp**(*s*)

Parameters **s** – packet string

This is a long line of text.

analysis.**decode_packet**(*s*)

Parameters **s** – packet string

This is a long line of text.

analysis.**decode_file**(*fname*)

Parameters **name** – pcap file name

This is a long line of text.

Tracing

This is a long line of text.

class analysis.**Tracer**

currentState
This is a long line of text.

__init__(*initialState=None*)

Parameters **initialState** – initial state function

This is a long line of text.

Start(*pkt*)

Parameters **pkt** – packet

This is a long line of text.

Next(*pkt*)

Parameters **pkt** – packet

This is a long line of text.

analysis.**trace**(*fname*, *tracers*)

Parameters

- **fname** – pcap file name
- **tracers** – list of tracer classes

This is a long line of text.

7.1.8 Other

Capability

Something here.

Classes

class capability.**Capability**

_zIndex
Capability functions are ordered by this attribute.

class capability.**Collector**

capabilities
A list of Capability derived classes that are in the inheritance graph.

__init__()
At initialization time the collector searches through the inheritance graph and builds the list of Capability derived classes and then calls the *__init__()* method for each of them.

capability_functions(*fn*)

Parameters **fn** (*string*) – name of a capability function

A generator that yields all of the functions of the Capability classes with the given name, ordered by z-index.

add_capability(*cls*)

Parameters **cls** (*class*) – add a Capability derived class

Add a Capability derived class to the method resolution order of the object. This will give the object a new value for its __class__ attribute. The *__init__()* method will also be called with the object instance.

This new capability will only be given to the object, no other objects with the same type will be given the new capability.

_search_capability(*base*)
This private method returns a flatten list of all of the Capability derived classes, including other Collector classes that might be in the inheritance graph using recursion.

Functions

capability.**compose_capability**(*base*, **classes*)

Parameters

- **base** (Collector) – Collector derived class
- **classes** (Capability) – Capability derived classes

Create a new class composed of the base collector and the provided capability classes.

capability.**add_capability**(*base*, **classes*)

Parameters

- **base** (Collector) – Collector derived class
- **classes** (Capability) – Capability derived classes

Add a capability derived class to a collector base.

Note: Objects that were created *before* the additional capabilities were added will have the new capability, but the *__init__()* functions of the classes will not be called.

Objects created *after* the additional capabilities were added will have the additional capabilities with the *__init__()* functions called.

Command Logging

The follow set of classes are used to provide access to the defined loggers as a client or a service. For example, instances of these classes can be stacked on top of a UDP or TCP director to provide debugging to remote devices or to BACpypes applications running as a daemon where there is no interactive command capability.

class commandlogging.**CommandLoggingHandler**(*logging.Handler*)
This is a long line of text.

__init__(*self*, *commander*, *destination*, *loggerName*)

Parameters

- **commander** – record to format
- **destination** – record to format
- **loggerName** – record to format

This is a long line of text.

emit(*self*, *record*)

Parameters **commander** – record to format

This is a long line of text.

class commandlogging.**CommandLogging**(*Logging*)
This is a long line of text.

handlers
This is a long line of text.

process_command(*self*, *cmd*, *addr*)

Parameters

- **cmd** – command message to be processed
- **addr** – address of source of request/response

This is a long line of text.

emit(*self*, *msg*, *addr*)

Parameters

- **msg** – message to send
- **addr** – address to send request/response

This is a long line of text.

class commandlogging.**CommandLoggingServer**(*CommandLogging*, *Server*, *Logging*)
This is a long line of text.

indication(*pdu*)

Parameters **pdu** – command message to be processed

This is a long line of text.

emit(*self*, *msg*, *addr*)

Parameters

- **msg** – message to send
- **addr** – address to send response

This is a long line of text.

class commandlogging.**CommandLoggingClient**(*CommandLogging*, *Client*, *Logging*)

This is a long line of text.

confirmation(*pdu*)

Parameters **pdu** – command message to be processed

This is a long line of text.

emit(*self*, *msg*, *addr*)

Parameters

- **msg** – message to send
- **addr** – address to send request

This is a long line of text.

IO Control Block

The IO Control Block (IOCB) is a data structure that is used to store parameters for some kind of processing and then used to retrieve the results of that processing at a later time. An IO Controller (IOController) is the executor of that processing.

They are modeled after the VAX/VMS IO subsystem API in which a single function could take a wide variety of combinations of parameters and the application did not necessarily wait for the operation to complete, but could be notified when it was by an event flag or semaphore. It could also provide a callback function to be called when processing was complete.

For example, given a simple function call:

```
result = some_function(arg1, arg2, kwarg1=1)
```

The IOCB would contain the arguments and keyword arguments, the some_function() would be the controller, and the result would alo be stored in the IOCB when the function is complete.

If the IOController encountered an error during processing, some value specifying the error is also stored in the IOCB.

Classes

There are two fundamental classes in this module, the *IOCB* for bundling request parameters together and processing the result, and *IOController* for executing requests.

The *IOQueue* is an object that manages a queue of IOCB requests when some functionality needs to be processed one at a time, and an *IOQController* which has the same signature as an IOController but takes advantage of a queue.

The *IOGroup* is used to bundle a collection of requests together that may be processed by separate controllers at different times but has *wait()* and *add_callback()* functions and can be otherwise treated as an IOCB.

class iocb.**IOCB**

The IOCB contains a unique identifier, references to the arguments and keyword arguments used when it was constructed, and placeholders for processing results or errors.

ioID
Every IOCB has a unique identifier that persists for the lifetime of the block. Similar to the Invoke ID for confirmed services, it can be used to synchronize communications and related functions.

The default identifier value is a thread safe monotonically increasing value.

args, kwargs
These are copies of the arguments and keyword arguments passed during the construction of the IOCB.

ioState
The ioState of an IOCB is the state of processing for the block.

- *idle* - an IOCB is idle when it is first constructed and before it has been given to a controller.
- *pending* - the IOCB has been given to a controller but the processing of the request has not started.
- *active* - the IOCB is being processed by the controller.
- *completed* - the processing of the IOCB has completed and the positive results have been stored in *ioResponse*.
- *aborted* - the processing of the IOCB has encountered an error of some kind and the error condition has been stored in *ioError*.

ioResponse
The result that some controller is providing to the application that created the IOCB.

ioError
The error condition that the controller is providing when the processing resulted in an error.

__init__(**args*, ***kwargs*)

Parameters

- **args** – arbitrary arguments
- **kwargs** – arbitrary keyword arguments

Create an IOCB and store the arguments and keyword arguments in it. The IOCB will be given a unique identifier and start in the *idle* state.

complete(*msg*)

Parameters **msg** – positive result of request

abort(*msg*)

Parameters **msg** – negative results of request

trigger()
This method is called by complete() or abort() after the positive or negative result has been stored in the IOCB.

wait(**args*)

Parameters **args** – arbitrary arguments

Block until the IO operation is complete and the positive or negative result has been placed in the ICOB. The arguments are passed to the *wait()* function of the ioComplete event.

add_callback(*fn*, **args*, ***kwargs*)

Parameters

- **fn** – the function to call when the IOCB is triggered
- **args** – additional arguments passed to the function

- **kwargs** – additional keyword arguments passed to the function

Add the function *fn* to a list of functions to call when the IOCB is triggered because it is complete or aborted. When the function is called the first parameter will be the IOCB that was triggered.

An IOCB can have any number of callback functions added to it and they will be called in the order they were added to the IOCB.

If the IOCB is has already been triggered then the callback function will be called immediately. Callback functions are typically added to an IOCB before it is given to a controller.

set_timeout(*delay*, *err=TimeoutError*)

Parameters

- **delay** (*seconds*) – the time limit for processing the IOCB
- **err** – the error to use when the IOCB is aborted

Set a time limit on the amount of time an IOCB can take to be completed, and if the time is exceeded then the IOCB is aborted.

class iocb.**IOController**

An IOController is an API for processing an IOCB. It has one method *process_io()* provided by a derived class which will be called for each IOCB that is requested of it. It calls one of its *complete_io()* or *abort_io()* functions as necessary to satisfy the request.

This class does not restrict a controller from processing more than one IOCB simultaneously.

request_io(*iocb*)

Parameters **iocb** – the IOCB to be processed

This method is called by the application requesting the service of a controller.

process_io(*iocb*)

Parameters **iocb** – the IOCB to be processed

The implementation of *process_io()* should be written using "functional programming" principles by not modifying the arguments or keyword arguments in the IOCB, and without side effects that would require the application using the controller to submit IOCBs in a particular order. There may be occasions following a "remote procedure call" model where the application making the request is not in the same process, or even on the same machine, as the controller providing the functionality.

active_io(*iocb*)

Parameters **iocb** – the IOCB being processed

This method is called by the derived class when it would like to signal to other types of applications that the IOCB is being processed.

complete_io(*iocb*, *msg*)

Parameters

- **iocb** – the IOCB to be processed
- **msg** – the message to be returned

This method is called by the derived class when the IO processing is complete. The *msg*, which may be None, is put in the *ioResponse* attribute of the IOCB which is then triggered.

IOController derived classes should call this function rather than the *complete()* function of the IOCB.

abort_io(*iocb*, *msg*)

Parameters

- **iocb** – the IOCB to be processed
- **msg** – the error to be returned

This method is called by the derived class when the IO processing has encountered an error. The *msg* is put in the *ioError* attribute of the IOCB which is then triggered.

IOController derived classes should call this function rather than the *abort()* function of the IOCB.

abort(*err*)

Parameters **msg** – the error to be returned

This method is called to abort all of the IOCBs associated with the controller. There is no default implementation of this method.

class iocb.**IOQueue**

An IOQueue is simply a first-in-first-out priority queue of IOCBs, but the IOCBs are modified to know that they can been queued. If an IOCB is aborted before being retrieved from the queue, it will ask the queue to remove it.

put(*iocb*)

Parameters **iocb** – add an IOCB to the queue

get(*block=1*, *delay=None*)

Parameters

- **block** – wait for an IOCB to be available in the queue
- **delay** – maximum time to wait for an IOCB

The *get()* request returns the next IOCB in the queue and waits for one if there are none available. If *block* is false and the queue is empty, it will return None.

remove(*iocb*)

Parameters **iocb** – an IOCB to remove from the queue

Removes an IOCB from the queue. If the IOCB is not in the queue, no action is performed.

abort(*err*)

Parameters **msg** – the error to be returned

This method is called to abort all of the IOCBs in the queue.

class iocb.**IOQController**

An *IOQController* has an identical interface as the *IOContoller*, but provides additional hooks to make sure that only one IOCB is being processed at a time.

request_io(*iocb*)

Parameters **iocb** – the IOCB to be processed

This method is called by the application requesting the service of a controller. If the controller is already busy processing a request, this IOCB is queued until the current processing is complete.

process_io(*iocb*)

Parameters **iocb** – the IOCB to be processed

Provided by a derived class, this is identical to *IOController.process_io*.

active_io(*iocb*)

Parameters **iocb** – the IOCB to be processed

Called by a derived class, this is identical to *IOController.active_io*.

complete_io(*iocb*, *msg*)

Parameters **iocb** – the IOCB to be processed

Called by a derived class, this is identical to *IOController.complete_io*.

abort_io(*iocb*, *msg*)

Parameters **iocb** – the IOCB to be processed

Called by a derived class, this is identical to *IOController.abort_io*.

abort(*err*)

Parameters **msg** – the error to be returned

This method is called to abort all of the IOCBs associated with the controller. All of the pending IOCBs will be aborted with this error.

class iocb.**IOGroup**(*IOCB*)

An *IOGroup* is like a set that is an IOCB. The group will complete when all of the IOCBs that have been added to the group are complete.

add(*iocb*)

Parameters **iocb** – an IOCB to include in the group

Adds an IOCB to the group.

abort(*err*)

Parameters **err** – the error to be returned

This method is call to abort all of the IOCBs that are members of the group.

group_callback(*iocb*)

: param iocb: the member IOCB that has completed

This method is added as a callback to all of the IOCBs that are added to the group and it is called when each one completes. Its purpose is to check to see if all of the IOCBs have completed and if they have, trigger the group as completed.

class iocb.**IOChainMixIn**

The IOChainMixIn class adds an additional API to things that act like an IOCB and can be mixed into the inheritance chain for translating requests from one form to another.

__init__(*iocb*)

Parameters **iocb** – the IOCB to chain from

Create an object that is chained from some request.

encode()

This method is called to transform the arguments and keyword arguments into something suitable for the other controller. It is typically overridden by a derived class to perform this function.

decode()

This method is called to transform the result or error returned by the other controller into something suitable to return. It is typically overridden by a derived class to perform this function.

chain_callback(*iocb*)

Parameters **iocb** – the IOCB that has completed, which is itself

When a chained IOCB has completed, the results are translated or decoded for the next higher level of the application. The *iocb* parameter is redundant because the IOCB becomes its own controller, but the callback API requires the parameter.

abort_io(*iocb*, *err*)

Parameters

- **iocb** – the IOCB that is being aborted
- **err** – the error to be used as the abort reason

Call this method to abort the IOCB, which will in turn cascade the abort operation to the chained IOCBs. This has the same function signature that is used by an IOController because this instance becomes its own controller.

class iocb.**IOChain**(*IOCB*, *IOChainMixIn*)

An IOChain is a class that is an IOCB that includes the IOChain API. Chains are used by controllers when they need the services of some other controller and results need to be processed further.

Controllers that operate this way are similar to an adapter, they take arguments in one form, encode them in some way in an IOCB, pass it to the other controller, then decode the results.

class iocb.**ClientController**(*Client*, *IOQController*)

An instance of this class is a controller that sits at the top of a protocol stack as a client. The IOCBs to be processed contain a single PDU parameter that is sent down the stack. Any PDU coming back up the stack is assumed to complete the current request.

This class is used for protocol stacks with a strict master/slave architecture.

This class inherits from *IOQController* so if there is already an active request then subsequent requests are queued.

class iocb.**_SieveQueue**(*IOQController*)

This is a special purpose controller used by the *SieveClientController* to serialize requests for the same source/destination address.

class iocb.**SieveClientController**(*Client*, *IOController*)

Similar to the *ClientController*, this class is a controller that also sits at the top of a protocol stack as a client. The IOCBs to be processed contain a single PDU parameter with a *pduDestination* address. Unlike the *ClientController*, this class creates individual queues for each destination address so it can process multiple requests simultaneously while maintaining a strict master/slave relationship with each address.

When an upstream PDU is received, the *pduSource* address is used to associate this response with the correct request.

Functions

iocb.**register_controller**(*controller*)

Parameters **controller** – controller to register

The module keeps a dictionary of "registered" controllers so that other parts of the application can find the controller instance. For example, if an HTTP controller provided a GET service and it was registered then other parts of the application could take advantage of the service the controller provides.

CHAPTER 8

Indices and tables

- genindex
- modindex
- search

Python Module Index

Index

Symbols

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

R

S

T

U

V

W