An Introduction to the ipaddress Module

Overview:

This document provides an introduction to the ipaddress module used in the Python language for manipulation of IPv4 and IPv6 addresses. At a high level, IPv4 and IPv6 addresses are used for like purposes and functions. However, since there are major differences in the address structure for each protocol, this tutorial has separated into separate sections, one each for IPv4 and IPv6.

In today's Internet, the IPv4 protocol controls the majority of IP processing and will remain so for the near future. The enhancements in scale and functionality that come with IPv6 are necessary for the future of the Internet and adoption is progressing. The adoption rate, however, remains slow to this date.

IPv4 and the ipaddress Module:

The following is a brief discussion of the engineering of an IPv4 address. Only topics pertinent to the ipaddress module are included.

A IPv4 address is composed of 32 bits, organized into four eight bit groupings referred to as "octets". The word "octet" is used to identify an eight-bit structure in place of the more common term "byte", but they carry the same definition. The four octets are referred to as octet1, octet2, octet3, and octet4. This is a "dotted decimal" format where each eight-bit octet can have a decimal value based on eight bits from zero to 255.

IPv4 example: 192.168.100.10

Every packet in an IPv4 network contains a separate source and destination address. As a unique entity, a IPv4 address should be sufficient to route an IPv4 data packet from the source address of the packet to the destination address of the packet on a IPv4 enabled network, such as the Internet.

IPv4 addresses are assigned to hosts (any IP capable end-user device) and to interfaces on routers. IPv4 domains are composed of networks, or like groupings of nodes within a defined area, normally a Local Area Network (LAN or VLAN). A portion of every IPv4 address is used to define membership in a specific network, with the balance of the address used to identify the individual node within the network.

Remembering that an IPv4 address is 32 bits long, the reference to the number of bits used to identify the network is referenced with a "CIDR notation" that discloses the number of leading bits that belong to network identification. The balance of the 32 bits are used to identify specific host/node/interface addresses within the network.

IPv4 example: **192.168.100.10/24**

The '/24' is CIDR notation to indicate that leading 24 of the 32 bits are used to identify the network portion of the address. Remembering that each octet is 8 bits long, this means that the first three octets (3 X 8 = 24) identify the network (**192.168.100**.x) and the remaining eight bits of the address identify the node (x.x.x.**10**).

Our example shows a typical network address structure where the eight bits in octet4 are used for node/host identification. Eight bits provide 256 values from zero to 255, but the first value in any network is reserved for network identification (in our example: 192.168.100.**0**) and the last is reserved as a broadcast address (in our example: 192.168.100.**255**). Neither can be used for host identification.

CIDR notation can be anything from /8 bits through to /30 bits, with an occasional /32 bits (/31 is invalid), but /24 is often used. For example, your home network, or your school or company network is most likely identified with a /24 CIDR.

In the Internet Service Provider (ISP) environment, CIDR notations of less than /24 are used to consolidate network routing; this is referred to as route summarization. In an enterprise/business/home environment, CIDR notations greater than /24 are used for special purposes. For example, a /30 is often used for point-to-point network links as they only require two IPv4 addresses within the network; this is referred to as subnetting.

CIDR notation was not part of the original IPv4 definition, but was added to help scale the protocol to handle the exponential increase in traffic and routing on the Internet. The legacy method of defining the network and node portions of an IPv4 address was to use a "network mask" which is still extensively used for node assignment and router interface provisioning. The network mask is a 32-bit entity separate from the IPv4 address. Each bit in the mask (and therefore each octet) corresponds to it's like position in the IPv4 address. If a bit in the mask is "one" then that bit in the IPv4 address is part of the network.

IPv4 example: **192.168.100.10**
IPv4 mask: **255.255.255.0**

This example corresponds to a /24 CIDR. The first octet is 255, which means that all eight bits are "one". This is also true for octet2 and octet3 which gives a total of 24 bits of network addressing.

The subnet mask is often specified at octet boundaries: 255.255.0.0 (or /16) and 255.255.255.0 (or /24). In parallel with CIDR notation, other subnet masks are used to accommodate specific network requirements. For example, a 255.255.255.252 (or /30) mask is often used for point-to-point links.

The ipaddress module is designed around CIDR notation, which is recommended because of its brevity and ease of use. The ipaddress module also includes methods to revert to a network mask if required.

The original definition of IPv4 addresses includes a “class” that is defined by address ranges in the first octet. The ipaddress module does not recognize IPv4 classes and is therefore not included in this tutorial.

Additionally, the IPv4 addressing protocol defines ranges of IPv4 addresses for private networks, referred to a “private” addresses as opposed to IPv4 addresses used to route over the Internet which are referred to a “public” addresses. The ipaddress module briefly mentions private vs. public addressing and we will discuss the difference at that time.

Creating IPv4 Address/Interface/Network Objects:

The ipaddress module includes three specific **ip address object types**:

1) a “host” or an individual ***address object*** that does not include CIDR notation,
2) an individual ***interface address object*** that includes CIDR notation, and
3) and a ***network address object*** that refers to the range of IP addresses for the entire network.

The major difference between a “host” and an “interface” is that a host or “ip_address” object does not include CIDR notation, whereas an “ip_interface” object includes the CIDR notation:

- The ***ip_address object*** is most useful when working with IP packets that do not need nor use CIDR notation.
- The ***ip_interface object*** is most useful when working with node and interface identification for connection to an IP network which must include network/subnet identification.
- The ***ip_network object*** includes all addresses within a network and is most useful for network identification.

IPv4 Host Address Object:

The **`ipaddress.ip_address()`** factory function is used to create an ip_address object. This automatically determines whether to create an IPv4 or IPv6 address based on the passed-in value (IPv6 addressing will be discussed at a latter point in this tutorial). As noted above, this object represents an IP Address as found in a packet traversing a network where CIDR is not required.

In many cases, the value used to create an ip_address object will be a string in the IPv4 dotted decimal format as per this example:

```
>>> myIP = ipaddress.ip_address ( "192.168.100.10" )
>>> myIP
IPv4Address('192.168.100.10')
```

Alternatively, the IPv4 address may be entered in binary, as a decimal value of the full 32 bit binary value, or in hexadecimal format as per this example:

```
>>> # All 32 binary bits can be used to create an IPv4 address:
>>> ipaddress.ip_address ( 0b11000000101010000110010000001010 )
IPv4Address('192.168.100.10')
>>> # The decimal value of the 32 bit binary number can also be used:
>>> ipaddress.ip_address (3232261130)
IPv4Address('192.168.100.10')
>>> # As can the hexadecimal value of the 32 bits:
>>> ipaddress.ip_address ( 0xC0A8640A )
IPv4Address('192.168.100.10')
```

The first example uses the full 32 bit address, and the second example is the decimal value of the 32 bit address. Both are unwieldy, error-prone and of limited value. The third example uses a hexadecimal value which can be useful as most packet formats from parsing or sniffing are represented in hexadecimal format.

IPv4 Interface Address Object:

The **`ipaddress.ip_interface()`** factory function is used to create an ip_interface object, which automatically determines whether to create an IPv4 or IPv6 address based on the passed-in value (IPv6 addressing will be discussed at a latter point in this tutorial). As previously discussed, the ip_interface object represents the ip address found on a host or network interface where the CIDR (or mask) is required for proper handling of the packet.

```
>>> # An ip_interface object is used to represent IP addressing
>>> # for a host or router interface, including the CIDR:
>>> myIP = ipaddress.ip_interface ( '192.168.100.10/24' )
>>> myIP
IPv4Interface('192.168.100.10/24')
>>> # This method translates the CIDR into a mask as would normally
>>> # be used on a host or router interface
>>> myIP.netmask
IPv4Address('255.255.255.0')
```

One can use the same options in the creation of an ip_interface option as with an ip_address option (binary, decimal value, hexadecimal). However, the only way to effectively create an ip_interface with the proper CIDR notation or mask is with a dotted decimal IPv4 address string.

IPv4 Network Address Object:

The **`ipaddress.ip_network()`** factory function is used to create an ip_network object, which automatically determines whether to create an IPv4 or IPv6 address based on the passed-in value (IPv6 addressing will be discussed at a latter point in this tutorial).

An IP network is defined as a range of consecutive IP address that define a network or subnet. Example:

- 192.168.100.0/24 is the 192.168.100.0 network where the /24 specifies that the first three octets comprise the network identification.
- The 4th octet is used for assignment to individual hosts and router interfaces.
- The address range is 192.168.100.1 through to .254.
- 192.168.100.0 is used to define the network/subnet and 192.168.100.255 is the broadcast address for this network. Neither can be used for assignment to a host or router interface.

The creation of a ip_network object follows the same syntax as the creation of a ip_interface object:

```
>>> # Creates a ip_network object.  The IPv4 address and CIDR must be
>>> # a valid network address, the first address in an address range:
>>> myIPnet = ipaddress.ip_network ( '192.168.100.0/24' )
>>> myIPnet
IPv4Network('192.168.100.0/24')
```

In the above example, the network address used must be a valid network address, which is the first address in the range of IPv4 addresses that constitute the network. If this is not the case, Python will throw a traceback exception:

```
>>> # Python will throw an exception if the address used is not
>>> # a valid network address.  In the following, ".10" is a host address
>>> # not a valid network address ident cation, which is ".0":
>>> myIPnet = ipaddress.ip_network ( '192.168.100.10/24' )
Traceback (most recent call last):
  File "<pyshell#26>", line 1, in <module>
    myIPnet = ipaddress.ip_network ( '192.168.100.10/24' )
  File "C:\Program Files (x86)\lib\ipaddress.py", line 74, in ip_network
    return IPv4Network(address, strict)
  File "C:\Program Files (x86)\lib\ipaddress.py", line 1536, in __init__
    raise ValueError('%s has host bits set' % self)
ValueError: 192.168.100.10/24 has host bits set
```

When working with host or router interfaces, it is often necessary to determine the network address. This can be calculated, but takes several steps which can be accomplished in a single step using the "strict=False" option ("strict =True" is default).

```
>>> # If the network address needs to be calculated,
>>> # use the strict=False option.  This will calculate and populate
>>> # the ip_network object with the network rather than the
>>> # interface address:
>>> myIP = ipaddress.ip_interface ( '192.168.100.10/24' )
>>> myIP
IPv4Interface('192.168.100.10/24')
>>> testIPnet = ipaddress.ip_network ( myIP, strict=False )
>>> testIPnet
IPv4Network('192.168.100.0/24')
```

In the above example, the ip_interface address is known (192.168.100.10) but not the ip_network the interface belongs to. Using the "strict=False" option, the ip_network address (192.168.100.0/24) is calculated and populated in the ip_network object.

IPv6 and the ipaddress Module:

There are many differences between IPv4 and IPv6, but the notable difference is in the addressing structure. Most importantly, IPv6 addresses are 128 bits long, which is a significant increase over the 32 bits in a IPv4 address. The additional length provides an exponential increase in the number of networks and host that can be supported.

IPv6 Address example: **2001:db8:abcd:100::1/64**

Where the IPv4 address uses a dotted decimal format, the IPv4 protocol uses hexadecimal notation. Each position in an IPv4 address represents four bits with a value from "0" to "f", organized as follows:

- The 128 bits are divided into 8 groupings of 16 bits each separated by colons. A group is referred to as a "quartet" or "hextet" each with four hexadecimal characters (4 hex characters times 4 bits = 16 bits). In the above example, the first quartet is 2001.

- Leading zeros in any quartet are suppressed/condensed. In the above example, the second quartet is "db8", which is actually "0db8" with the leading zero suppressed. The last quartet is "1", which is actually "0001" with three leading zeros suppressed.

- If a quartet contains all zeros, it is suppressed to a single zero. For example: a quartet with ":0000:" would be compressed to ":0:".

- If an address contains a contiguous string of quartets that are all zeros, the contiguous string of zeros is condensed and represented with double colons. In the above example, the double colon represents three all zero quartets, or ":0000:0000:0000:" condensed to "::". Since the example address has five quartets with values, the number of condensed quartets must be three (eight total minus five populated).

All IPv6 address structures used CIDR notation to determine how many of the leading bits are used for network identification with the balance used for host/interface identification. Given 128 bits, many options are available. However, as a best practice, a "/64" CIDR notation is most often used with other variations reserved for unique descriptions or specific implantations. A /64 CIDR indicates that the first four quartets (4 * 16 bits each = 64 bits) are used for network identification and the remaining 4 quartets are used for host/node/interface identification. This combines maximum utility of the address space with a simple to use and easy to understand implementation.

IPv6 Address Types:

There are several types of IPv6 address structures, each with its own specific coding. A full discussion of IPv6 address types is beyond the scope of this document. However, the following provides background on the most important address types as used in the ipaddress module. This includes:

IPv6 Address	Description	First Octet Range	Most Often Use:
Loopback Address	Used for testing and verification of IPv6 processing on a device. A successful IPv6 ping to ::1 insures the IPv6 driver is activated and processing normally.	0000 through 00FF	::1
Global unicast addresses	A unicast address that uniquely identifies an interface on an IPv6-enabled device. A Global unicast address has a unique network identification within the global IPv6 Internet.	2000 through 3FFF	first quartet = 2001
Link-local unicast addresses	Used for IPv6 communication within a local network.	FE80 through FEBF	first quartet = FE80
Unique local unicast addresses	Used within a private network where interface to the IPv6 Internet is not required. These addresses are not routable on the Internet and roughly correspond to private IPv4 addressing. IPv4 private addressing is normally paired with NAT/PAT for access to the Internet. NAT/PAT is not recommended for IPv6.	FC00 through FDFF	first quartet = FC00
Multicast addresses	An IPv6 multicast address is used to send a single IPv6 packet to multiple destinations. Used for communication to groups of devices	FF00 through FFFF	first quartet = FF00

Anycast addresses	An IPv6 anycast address is any IPv6 unicast address that can be assigned to multiple devices.	Assigning a unicast address to more than one interface makes a unicast address an anycast address

IPv6 Loopback Addresses:

A loopback address is used for testing and verification of the IPv6 stack within a device. A successful "ping" using a loopback address verifies that the IPv6 stack is operational.

IPv6 Global Addresses:

Each separate company/enterprise network will have one global IPv6 address that identifies that organization on the global IPv6 Internet. Global address are assigned by the organization's Internet Service Provider (ISP) based on blocks of addresses allocated to the ISP by the Internet authorities.

In our example, 2001:db8:abcd::/48 is the global identification, or the IPv6 Internet ID that will uniquely identify this organization on the IPv6 Internet. In our discussion, we will used this address as a constant to parallel the address that would be assigned to an individual company/enterprise:

- The first quartet is "2001:" with is a block assigned to North American Regional Internet Authority.
- The second quartet is ":db8:" which is reserved for testing/training/documentation and is not routable in the IPv6 Internet. This would normally be a code to indicate the ISP within the Regional Authority.
- The third quartet is ":abcd:" which is arbitrarily assigned value. This would normally be the ISP's customer identification.
- Taken all together, the 2001:db8:abcd::/48 address is the IPv6 sample address used in this tutorial. A /48 CIDR specifies that the first 48 bits are defined as the global address. While the format is complex and more detailed than with IPv4 addresses, it's important to note this becomes a constant within an organization that does not change in use or over time.

The above is an intentionally simplified as the detailed definition on a bit-by-bit basis is beyond the scope of this document.

As a best practice, the fourth quartet, is reserved for subnet identification within the organization. Remembering that each quarter is four hexadecimal digits long, sixteen bits provides unique subnet identification for 65,536 subnets (2 raised to the 16^{th} power), which should be sufficient subnet identification for even the very largest organization. In our example :100: is used as the subnet number, which is an arbitrary number used for illustration.

Given the above, the network number for our example is the first four quartets, or 2001:db8:abcd:100::/64 which makes up 64 bits of the global address, hence the /64 CIDR notation. It is recommended as a best

practice to always use /64 as the CIDR. In special circumstances other CIDR values can be used, but they should be reserved for special circumstances where other solutions are not possible.

IPv6 Link-local addresses

The IPv6 protocol includes an addressing structure that is used exclusively for communication with the boundaries of a network/subnet. This is used for overhead functions in place of where IPv4 would use broadcast packets (ARP, for example).

IPv6 differs from IPv4 in that multiple IP addresses are allowed on a host/interface. Generally, there will be a link-local address in IPv6 address in addition to the global IPv6 address.

The link-local IPv6 address is a simple to use and understand structure where the first octet is equal to “fe80:”. As a best practice, the router interface which serves as the default gateway to the Internet will have an address of “fe80::1 link-local”. As a best-practice, an individual host/interface on a network/subnet will use “fe80::1” as the default-gateway.

In additional to a simplified default-gateway, link-local addresses are used for the overhead protocols used within a network/subnet. This includes the automatic assignment of IPv6 addresses (SLAAC) which is beyond the scope of this discussion.

IPv6 Multicast Addresses:

A multicast address is used for special purpose packets destined for a multiple nodes/hosts. A discussion of purpose and use is beyond the scope of this discussion. However, it may be important to identify a multicast packet. They are identified with “ff02:” as the first quartet. The host ID is important: a “ff02::1” indicates an “all-nodes” multicast and a “ff02::2” indicates an all-routers multicast.

IPv6 Unique Local Addresses:

The IPv6 protocol provides a simplified addressing structure called “unique local” for use within enterprise/company, but that is not routable to the Internet. This is analogous to IPv4 private addressing. As with IPv4, if a packet needs go to the Internet, the address must be translated for routing outside the enterprise/company network. IPv4 uses Network Address Translation (NAT) for this translation, and IPv6 provides similar functionality.

The rationality for unique-local addressing is simplicity. The somewhat complex format for the global address is replaced with the first quartet = “FC00”. A unique-local network address assignment could be fc00:0:0:100::1/64 which will be functionally equivalent to global addressing within the confines of an enterprise/company network, but with the advantage of ease of use.

Anycast Addresses:

A discussion of IPv6 anycast addressing is an advanced topic beyond the scope of this document.

IPv6 Host/Interface Identification:

The balance of an IPv6 address (the remaining four quartets or 64 bits for a /64 CIDR), is used for host and/or interface identification. There are two standards used to populate the host ID: **static assignment** and auto **EUI-64 assignment**.

Static assignment is used for network infrastructure hardware. For example, the router that is the default gateway for subnet 100 will have an ID of 2001:db8:abcd:100::1/64 where the subnet = 100 and the host ID = 1. On point-to-point links, one side would be ::1 and the opposite side will be ::2.

The EUI-64 is a protocol is used for automatic assignment of a unique host ID via automatic provisioning of host addresses in a IPv6 network. The full definition of EUI-64 provisioning is beyond the scope of this tutorial, but the foundation of the address is helpful. A link-local address is a derivative of the MAC address found on an interface. Using the MAC address as the base guarantees that a unique logical host address will be assigned. However, the full address will look complicated and is potentially confusing.

Example of a host IPv6 address with EUI-64 encoding for the host ID:

2001:DB8:ABCD:100:240:BFF:FE30:7086/64

IPv6 Network ID - **2001:DB8:ABCD:100**:240:BFF:FE30:7086**/64**

MAC Address - **0040.0B30.7086**

IPv6 Host ID - **2001:DB8:ABCD:100:240:BFF:FE30:7086**/64

The IPv6 Host address is a derivative of the MAC address using the EUI-64 algorithm.

The above IPv6 address illustrates how the EUI-64 protocol populates the last 64 bits of the IPv6 address using the host MAC address as a base. A MAC address is 48 bits long and is expanded to 64 bits by splitting the address in half and inserting hex “fffe” in the middle of the split. This creates a 64 bit address (48 bits + 16 bits). Additionally, the seventh bit of the MAC address is flipped, which generally means that the 5th quartet will be changed. In the above example: the fifth quartet which started as “0040” is changed to “0240” where the “2” bit (seventh bit from the left) is turned on.

Generally, we need not be concerned over EUI-64 host/interface addressing. It is, however, handy to recognize that the “fffe” in the middle of the host portion of an IPv6 address means that the address was generated via EUI-64.

Other IPv6 Considerations:

There are other IPv6 address structures that are not used or specifically recognized by the ipaddress module and are therefore not covered in this tutorial.

Because of the128 bit length and it's specialized formats, IPv6 addresses can seem intimidating. However, once the differences are absorbed, the two protocols operate and act in a similar, but not identical, manner with the IPv6 protocol generally simpler and easier to understand and use.

Creating IPv6 Address/Interface/Network Objects:

As with IPv4, the ipaddress module uses the same three basic factory functions already described for IPv4. includes include:

1) a "host" or an ***individual address object*** that does not include CIDR notation,
2) an ***interface address*** object that includes CIDR notation, and
3) and a ***network address object*** that refers to the range of IP addresses for the entire network.

Since the detail is covered in the section on IPv4, a brief overview is only necessary.

IPv6 Host Address Object:

The **`ipaddress.ip_address()`** factory function is used to create an ip_address object. This automatically knows to use the IPv6 address format based on the passed-in value. Note that the CIDR notation is not used with the ipaddress function.

In the majority of cases, the value used to create an ip_address object for IPv6 will be a string in the IPv6 quartet/hextet format as per this example:

```
>>> # Create an IPv6 Address Object for a Global Address:
>>> myIPv6addr = ipaddress.ip_address ( "2001:db8:abcd:100::1" )
>>> myIPv6addr
IPv6Address('2001:db8:abcd:100::1')
>>> # Create an IPv6 Address Object for a link-local address:
>>> myIPv6 = ipaddress.ip_address ("fe80::1")
>>> myIPv6
IPv6Address('fe80::1')
```

As with IPv4, it is possible to create an IPv6 address object using the full binary, decimal, or hexadecimal value. This is unwieldy with 32 bits for an IPv4 address and is even more awkward for a 128 bit IPv6 address. As a practical matter, it is anticipated that the string representation of the eight quartets will be the norm.

IPv6 Interface Address Object:

The **`ipaddress.ip_interface()`** factory function is used to create an ip_interface object, which automatically create an IPv6 address based on the passed-in value. Note that the CIDR notation must be included in the function.

```
>>> # Creates an IP Interface Object for a Global Address:
>>> myIPv6int = ipaddress.ip_interface ( "2001:db8:abcd:100::1/64" )
>>> myIPv6int
IPv6Interface('2001:db8:abcd:100::1/64')
>>> # Creates an IP Interface Object for a Link-local Address:
>>> myIPv6 = ipaddress.ip_interface ("fe80::1/64")
>>> myIPv6
IPv6Interface('fe80::1/64')
```

IPv6 Network Address Object:

The **ipaddress.ip_network()** factory function is used to create an ip_network object for IPv6 based on the passed-in value.

As with IPv4, an IPv6 network is defined as a range of consecutive IP address that can be assigned to specific host or router interfaces. Example: using our example of 2001:db8:abcd:100::/64, the /64 CIDR specifies that the four quartets make up the full network identification. Remember that the first three quartets are global ID assigned by the IPS and the fourth quartet identifies the internal subnet number. The balance of the 64 bits are used for host identification with a range from "0000:0000:0000:0001" though to "ffff:ffff:ffff:fffe". As with IPv4 addressing, the first and last address in a IPv6 subnet cannot be used for host addressing. Given a /64 CIDR, this means that there are 2 to the 64th power (minus 2) possible host addresses, which is means there are 18,446,744,073,709,551,614 mathematically possible host addresses per network/subnet.

```
>>> # Creates an IP Network Object for a Global Address:
>>> myIPv6net = ipaddress.ip_network ( "2001:db8:abcd:100::/64" )
>>> myIPv6net
IPv6Network('2001:db8:abcd:100::/64')
>>> # Creates an IP Network Object for a Link-local Address:
>>> myIPv6 = ipaddress.ip_network ("fe80::/64")
>>> myIPv6
IPv6Network('fe80::/64')
```

The above global address is broken down as follows:

- Global Identifier assigned by ISP: **2001:db8:abcd::/48**
- Subnet identification: **2001:db8:abcd:100::/64**
- First usable address in the subnet: **2001:db8:abcd:100::1/64**
- Last usable address in the subnet: **2001:db8:abcd:100:ffff:ffff:ffff:fffe/64**