

---

# **Ironic Python Agent Documentation**

*Release 10.0.1.dev1*

**OpenStack Foundation**

**Nov 25, 2024**



# CONTENTS

<b>1</b>	<b>Overview</b>	<b>1</b>
<b>2</b>	<b>Contents</b>	<b>3</b>
2.1	Installing Ironic Python Agent . . . . .	3
2.1.1	Image Builders . . . . .	3
2.1.2	IPA Flags . . . . .	3
2.1.3	IPA and TLS . . . . .	3
2.1.4	Hardware Managers . . . . .	5
2.2	Ironic Python Agent Administration . . . . .	5
2.2.1	How it works . . . . .	5
2.2.2	Built-in hardware managers . . . . .	9
2.2.3	Custom hardware managers . . . . .	13
2.2.4	Rescue mode . . . . .	14
2.2.5	Troubleshooting Ironic-Python-Agent (IPA) . . . . .	14
2.3	Contributing to Ironic Python Agent . . . . .	19
2.3.1	Hardware Managers . . . . .	19
2.3.2	Emitting metrics from Ironic-Python-Agent (IPA) . . . . .	24
2.3.3	Rescue Mode . . . . .	25
2.3.4	Generated Developer Documentation . . . . .	26
<b>3</b>	<b>Indices and tables</b>	<b>89</b>
	<b>Python Module Index</b>	<b>91</b>
	<b>Index</b>	<b>93</b>



## OVERVIEW

Ironic Python Agent (often abbreviated as IPA) is an agent for controlling and deploying [Ironic](#) controlled baremetal nodes. Typically run in a ramdisk, the agent exposes a REST API for provisioning servers.



## CONTENTS

## 2.1 Installing Ironic Python Agent

### 2.1.1 Image Builders

Unlike most other python software, you must build or download an IPA ramdisk image before use. This is because its not installed in an operating system, but instead is run from within a ramdisk.

Two kinds of images are published on every commit from every branch of IPA:

- **DIB** images are suitable for production usage and can be downloaded from <https://tarballs.openstack.org/ironic-python-agent/dib/files/>.
- **TinyIPA** images are suitable for CI and testing environments and can be downloaded from <https://tarballs.openstack.org/ironic-python-agent/tinyipa/files/>.

If you need to build your own image, use the tools from the [ironic-python-agent-builder](#) project.

### 2.1.2 IPA Flags

You can pass a variety of flags to IPA on start up to change its behavior.

- `--debug`: Enables debug logging.

### 2.1.3 IPA and TLS

#### Client Configuration

During its operation IPA makes HTTP requests to a number of other services, currently including

- `ironic` for lookup/heartbeats
- `ironic-inspector` to publish results of introspection
- HTTP image storage to fetch the user image to be written to the nodes disk (Object storage service or other service storing user images when `ironic` is running in a standalone mode)

When these services are configured to require TLS-encrypted connections, IPA can be configured to either properly use such secure connections or ignore verifying such TLS connections.

Configuration mostly happens in the IPA config file (default is `/etc/ironic_python_agent/ironic_python_agent.conf`, can also be any file placed in `/etc/ironic-python-agent.d`) or command line arguments passed to `ironic-python-agent`, and it is possible to provide some options via kernel command line arguments instead.

Available options in the `[DEFAULT]` config file section are:

### **insecure**

Whether to verify server TLS certificates. When not specified explicitly, defaults to the value of `ipa-insecure` kernel command line argument (converted to boolean). The default for this kernel command line argument is taken to be `False`. Overriding it to `True` by adding `ipa-insecure=1` to the value of `[pxe]pxe_append_params` in ironic configuration file will allow running the same IPA-based deploy ramdisk in a CI-like environment when services are using secure HTTPS endpoints with self-signed certificates without adding a custom CA file to the deploy ramdisk (see below).

### **cafile**

Path to the PEM encoded Certificate Authority file. When not specified, available system-wide list of CAs will be used to verify server certificates. Thus in order to use IPA with HTTPS endpoints of other services in a secure fashion (with `insecure` option being `False`, see above), operators should either ensure that certificates of those services are verifiable by root CAs present in the deploy ramdisk, or add a custom CA file to the ramdisk and set this IPA option to point to this file at ramdisk build time.

### **certfile**

Path to PEM encoded client certificate cert file. This option must be used when services are configured to require client certificates on SSL-secured connections. This cert file must be added to the deploy ramdisk and path to it specified for IPA via this option at ramdisk build time. This option has an effect only when the `keyfile` option is also set.

### **keyfile**

Path to PEM encoded client certificate key file. This option must be used when services are configured to require client certificates on SSL-secured connections. This key file must be added to the deploy ramdisk and path to it specified for IPA via this option at ramdisk build time. This option has an effect only when the `certfile` option is also set.

Currently a single set of `cafile/certfile/keyfile` options is used for all HTTP requests to the other services.

## **Server Configuration**

Starting with the Victoria release, the API provided by `ironic-python-agent` can also be secured via TLS. There are two options to do that:

### **Automatic TLS**

This option is enabled by default if no other options are enabled. If ironic supports API version 1.68, a new self-signed TLS certificate will be generated in runtime and sent to ironic on heartbeat.

No special configuration is required on the ironic side.

### **Manual TLS**

If you need to provide your own TLS certificate, you can configure it when building an image. Set the following options in the `ironic-python-agent` configuration file:

```
[DEFAULT]
listen_tls = True
advertise_protocol = https
# Disable automatic TLS.
enable_auto_tls = False

[ssl]
# Certificate and private key file paths (on the ramdisk).
cert_file = /path/to/certificate
```

(continues on next page)



(continued from previous page)

```
# The private key must not be password-protected!
key_file = /path/to/private/key
# Optionally, authenticate connecting clients (i.e. ironic conductors).
#ca_file = /path/to/ca
```

If using DIB to build the ramdisk, use the `ironic-python-agent-tls` element to automate these steps.

On the ironic side you have two options:

- If the certificate can pass host validation, i.e. contains the correct host name or IP address of the agent, add its path to each node with:

```
baremetal node set <node> --driver-info agent_verify_ca=/path/to/ca/
↪or/certificate
```

- Usually, the IP address of the agent is not known in advance, so you need to disable host validation instead:

```
baremetal node set <node> --driver-info agent_verify_ca=False
```

### 2.1.4 Hardware Managers

Hardware managers are how IPA supports multiple different hardware platforms in the same agent. Any action performed on hardware can be overridden by deploying your own hardware manager.

Custom hardware managers allow you to include hardware-specific tools, files and cleaning steps in the Ironic Python Agent. For example, you could include a BIOS flashing utility and BIOS file in a custom ramdisk. Your custom hardware manager could expose a cleaning step that calls the flashing utility and flashes the packaged BIOS version (or even download it from a tested web server).

Operators wishing to build their own hardware managers should reference the documentation available at *Hardware Managers*.

## 2.2 Ironic Python Agent Administration

### 2.2.1 How it works

#### Integration with Ironic

For information on how to install and configure Ironic drivers, including drivers for IPA, see the [Ironic drivers documentation](#).

#### Lookup

On startup, the agent performs a lookup in Ironic to determine its node UUID by sending a hardware profile to the Ironic lookup endpoint: `/v1/lookup`.

#### Heartbeat

After successfully looking up its node, the agent heartbeats via `/v1/heartbeat/{node_ident}` every N seconds, where N is the Ironic conductors `agent.heartbeat_timeout` value multiplied by a number between .3 and .6.

For example, if your conductors `ironic.conf` contains:

```
[agent]
heartbeat_timeout = 60
```

IPA will heartbeat between every 20 and 36 seconds. This is to ensure jitter for any agents reconnecting after a network or API disruption.

After the agent heartbeats, the conductor performs any actions needed against the node, including querying status of an already run command. For example, initiating in-band cleaning tasks or deploying an image to the node.

### Inspection

IPA can conduct hardware inspection on start up and post data to the [Ironic Inspector](#) via the `/v1/continue` endpoint.

Edit your default PXE/iPXE configuration or IPA options baked in the image, and set `ipa-inspection-callback-url` to the full endpoint of Ironic Inspector, for example:

```
ipa-inspection-callback-url=http://IP:5050/v1/continue
```

Make sure your DHCP environment is set to boot IPA by default.

If you use the new built-in [Ironic in-band inspection](#), it is enough to only set a list of collectors (see [inspection data](#)), for example:

```
ipa-inspection-collectors=default,logs
```

Then the correct callback URL will be determined from the Ironic URL in `ipa-api-url`.

### Instance agent

For the cases where the infrastructure operator and cloud user are the same, an additional tool exists that can be installed alongside the agent inside a running instance. This is the `ironic-collect-introspection-data` command which allows for a node in ACTIVE state to publish updated introspection data to `ironic-inspector`. This ability requires `ironic-inspector` to be configured with `[processing]permit_active_introspection` set to True. For example:

```
ironic-collect-introspection-data --inspection_callback_url http://IP:5050/v1/
↳continue
```

Alternatively, this command may also be used with multicast DNS functionality to identify the Ironic Inspector service endpoint. For example:

```
ironic-collect-introspection-data --inspection_callback_url mdns
```

An additional daemon mode may be useful for some operators who wish to receive regular updates, in the form of the `[DEFAULT]introspection_daemon` boolean configuration option. For example:

```
ironic-collect-introspection-data --inspection_callback_url mdns --
↳introspection_daemon
```

The above command will attempt to connect to introspection and will then enter a loop to publish every 300 seconds. This can be tuned with the `[DEFAULT]introspection_daemon_post_interval` configuration option.

## Inspection Data

As part of the inspection process, data is collected on the machine and sent back to [Ironic Inspector](#) for storage. It can be accessed via the [introspection data API](#).

The exact format of the data depends on the enabled *collectors*, which can be configured using the `ipa-inspection-collectors` kernel parameter. Each collector appends information to the resulting JSON object. The in-tree collectors are:

### default

The default enabled collectors. Collects the following keys:

- `inventory` - *Hardware Inventory*.
- `root_disk` - The default root device for this machine, which will be used for deployment if root device hints are not provided.
- `configuration` - Inspection configuration, an object with two keys:
  - `collectors` - List of enabled collectors.
  - `managers` - List of enabled *Hardware Managers*: items with keys `name` and `version`.
- `boot_interface` - Deprecated, use the `inventory.boot.pxe_interface` field.

### logs

Collect system logs. To yield useful results it must always go last in the list of collectors. Provides one key:

- `logs` - base64 encoded tarball with various logs.

### pci-devices

Collects the list of PCI devices. Provides one key:

- `pci_devices` - list of objects with keys `vendor_id` and `product_id`.

### extra-hardware

Collects a vast list of facts about the systems, using the [hardware](#) library, which is a required dependency for this collector. Adds one key:

- `data` - raw data from the `hardware-collect` utility. Is a list of lists with 4 items each. It is recommended to use this collector together with the `extra_hardware` processing hook on the Ironic Inspector side to convert it to a nested dictionary in the `extra` key.

If `ipa-inspection-benchmarks` is set, the corresponding benchmarks are executed and their result is also provided.

### dmi-decode

Collects information from `dmidecode`. Provides one key:

- `dmi` DMI information in three keys: `bios`, `cpu` and `memory`.

### numa-topology

Collects [NUMA](#) topology information. Provides one key:

- `numa_topology` with three nested keys:
  - `ram` - list of objects with keys `numa_node` (node ID) and `size_kb`.
  - `cpus` - list of objects with keys `cpu` (CPU ID), `numa_node` (node ID) and `thread_siblings` (list of sibling threads).

- `nics` - list of objects with keys `name` (NIC name) and `numa_node` (node ID).

### **lldp**

Collects information about the network connectivity using **LLDP**. Provides one key:

- `lldp_raw` - mapping of interface names to lists of raw type-length-value (TLV) records.

### **usb-devices**

Collects USB devices information. Adds one key:

- `usb_devices` - list of objects with keys `product`, `vendor` and `handle`

## **Hardware Inventory**

IPA collects various hardware information using its *Hardware Managers*, and sends it to Ironic on lookup and to Ironic Inspector on *Inspection*.

The exact format of the inventory depends on the hardware manager used. Here is the basic format expected to be provided by all hardware managers. The inventory is a dictionary (JSON object), containing at least the following fields:

### **cpu**

CPU information: `model_name`, `frequency`, `count`, `architecture`, `flags` and `socket_count`.

### **memory**

RAM information: `total` (total size in bytes), `physical_mb` (physically installed memory size in MiB, optional).

#### **Note**

The difference is that the latter includes the memory region reserved by the kernel and is always slightly bigger. It also matches what the Nova flavor would contain for this node and thus is used by the inspection process instead of `total`.

### **bmc\_address**

IPv4 address of the nodes BMC (aka IPMI v4 address), optional.

### **bmc\_v6address**

IPv6 address of the nodes BMC (aka IPMI v6 address), optional.

### **disks**

list of disk block devices with fields: `name`, `model`, `size` (in bytes), `rotational` (boolean), `wwn`, `serial`, `uuid`, `vendor`, `wwn_with_extension`, `wwn_vendor_extension`, `hctl` and `by_path` (the full disk path, in the form `/dev/disk/by-path/<rest-of-path>`).

### **interfaces**

list of network interfaces with fields: `name`, `mac_address`, `ipv4_address`, `lldp`, `vendor`, `product`, and optionally `biosdevname` (BIOS given NIC name) and `speed_mbps` (maximum supported speed).

#### **Note**

For backward compatibility, interfaces may contain `lldp` fields. They are deprecated, consumers should rely on the `lldp` inspection collector instead.

### **system\_vendor**

system vendor information from SMBIOS as reported by dmidecode: `product_name`, `serial_number` and `manufacturer`, as well as a firmware structure with fields `vendor`, `version` and `build_date`.

### **boot**

boot information with fields: `current_boot_mode` (boot mode used for the current boot - BIOS or UEFI) and `pxe_interface` (interface used for PXE booting, if any).

### **hostname**

hostname for the system

#### **Note**

This is most likely to be set by the DHCP server. Could be localhost if the DHCP server does not set it.

## **Image Checksums**

As part of the process of downloading images to be written to disk as part of image deployment, a series of fields are utilized to determine if the image which has been downloaded matches what the user stated as the expected image checksum utilizing the `instance_info/image_checksum` value.

OpenStack, as a whole, has replaced the legacy `checksum` field with `os_hash_value` and `os_hash_algo` fields, which allows for an image checksum and value to be asserted. An advantage of this is a variety of algorithms are available, if a user/operator is so-inclined.

For the purposes of Ironic, we continue to support the pass-through checksum field as we support the checksum being retrieved via a URL.

We also support determining the checksum by length.

The field can be utilized to designate:

- A URL to retrieve a checksum from.
- MD5 (Disabled by default, see `[DEFAULT]md5_enabled` in the agent configuration file.)
- SHA-2 based SHA256
- SHA-2 based SHA512

SHA-3 based checksums are not supported for auto-determination as they can have a variable length checksum result. At of when this documentation was added, SHA-2 based checksum algorithms have not been withdrawn from approval. If you need to force use of SHA-3 based checksums, you *must* utilize the `os_hash_algo` setting along with the `os_hash_value` setting.

## **2.2.2 Built-in hardware managers**

### **GenericHardwareManager**

This is the default hardware manager for `ironic-python-agent`. It provides support for *Hardware Inventory* and the default `deploy`, `clean`, and `service` steps.

## Deploy steps

### **deploy.write\_image(node, ports, image\_info, configdrive=None)**

A deploy step backing the `write_image` deploy step of the [direct deploy interface](#). Should not be used explicitly, but can be overridden to provide a custom way of writing an image.

### **deploy.erase\_devices\_metadata(node, ports)**

Erases partition tables from all recognized disk devices. Can be used with software RAID since it requires empty holder disks.

### **raid.apply\_configuration(node, ports, raid\_config, delete\_existing=True)**

Apply a software RAID configuration. It belongs to the `raid` interface and must be used through the [ironic RAID feature](#).

## Injecting files

### **deploy.inject\_files(node, ports, files, verify\_ca=True)**

This optional deploy step (introduced in the Wallaby release series) allows injecting arbitrary files into the node. The list of files is built from the optional `inject_files` property of the node concatenated with the explicit `files` argument. Each item in the list is a dictionary with the following fields:

#### **path (required)**

An absolute path to the file on the target partition. All missing directories will be created.

#### **partition**

Specifies the target partition in one of 3 ways:

- A number is treated as a partition index (starting with 1) on the root device.
- A path is treated as a block device path (e.g. `/dev/sda1` or `/dev/disk/by-partlabel/<something>`).
- If missing, the agent will try to find a partition containing the first component of the path on the root device. E.g. for `/etc/sysctl.d/my.conf`, look for a partition containing `/etc`.

#### **deleted**

If `True`, the file is deleted, not created. Incompatible with `content`.

#### **content**

Data to write. Incompatible with `deleted`. Can take two forms:

- A URL of the content. Can use Python-style formatting to build a node specific URL, e.g. `http://server/{node[uuid]}/{ports[0][address]}`.
- Base64 encoded binary contents.

#### **mode, owner, group**

Numeric mode, owner ID and group ID of the file.

#### **dirmode**

Numeric mode of the leaf directory if it has to be created.

This deploy step is disabled by default and can be enabled via a deploy template or via the `ipa-inject-files-priority` kernel parameter.

Known limitations:

- Names are not supported for `owner` and `group`.
- LVM is not supported.

## Clean steps

### **deploy.burnin\_cpu**

Stress-test the CPUs of a node via stress-ng for a configurable amount of time. Disabled by default.

### **deploy.burnin\_disk**

Stress-test the disks of a node via fio. Disabled by default.

### **deploy.burnin\_memory**

Stress-test the memory of a node via stress-ng for a configurable amount of time. Disabled by default.

### **deploy.burnin\_network**

Stress-test the network of a pair of nodes via fio for a configurable amount of time. Disabled by default.

### **deploy.erase\_devices**

Securely erases all information from all recognized disk devices. Relatively fast when secure ATA erase is available, otherwise can take hours, especially on a virtual environment. Enabled by default.

### **deploy.erase\_devices\_metadata**

Erases partition tables from all recognized disk devices. Can be used as an alternative to the much longer `erase_devices` step.

### **deploy.erase\_pstore**

Erases entries from pstore, the kernels oops/panic logger. Disabled by default. Can be enabled via priority overrides.

### **raid.create\_configuration**

Create a RAID configuration. This step belongs to the `raid` interface and must be used through the [ironic RAID feature](#).

### **raid.delete\_configuration**

Delete the RAID configuration. This step belongs to the `raid` interface and must be used through the [ironic RAID feature](#).

## Service steps

Service steps can be invoked by an operator of a baremetal node, to modify or perform some intermediate action outside the realm of normal use of a deployed bare metal instance. This is similar in form of interaction to cleaning, and ultimately some cleaning and deployment steps *are* available to be used.

### **deploy.burnin\_cpu**

Stress-test the CPUs of a node via stress-ng for a configurable amount of time.

### **deploy.burnin\_memory**

Stress-test the memory of a node via stress-ng for a configurable amount of time.

### **deploy.burnin\_network**

Stress-test the network of a pair of nodes via fio for a configurable amount of time.

### **raid.create\_configuration**

Create a RAID configuration. This step belongs to the `raid` interface and must be used through the [ironic RAID feature](#).

### **raid.apply\_configuration(node, ports, raid\_config, delete\_existing=True)**

Apply a software RAID configuration. It belongs to the `raid` interface and must be used through

the [ironic RAID feature](#).

### **raid.delete\_configuration**

Delete the RAID configuration. This step belongs to the `raid` interface and must be used through the [ironic RAID feature](#).

### **deploy.write\_image(node, ports, image\_info, configdrive=None)**

A step backing the `write_image` deploy step of the [direct deploy interface](#). Should not be used explicitly, but can be overridden to provide a custom way of writing an image.

### **deploy.inject\_files(node, ports, files, verify\_ca=True)**

A step to inject files into a system. Specifically this step is documented earlier in this documenta-tion.

#### Note

The Ironic Developers chose to limit the items available for service steps such that the risk of data destruction is generally minimized. That being said, it could be reasonable to reconfigure RAID devices through local hardware managers *or* to write the base OS image as part of a service operation. As such, caution should be taken, and if additional data erasure steps are needed you may want to consider moving a node through cleaning to remove the workload. Otherwise, if you have a use case, please feel free to reach out to the Ironic Developers so we can understand and enable your use case.

## Cleaning safeguards

The stock hardware manager contains a number of safeguards to prevent unsafe conditions from occurring.

### Devices Skip List

A list of devices that Ironic does not touch during the cleaning and deployment process can be specified in the node properties field under `skip_block_devices`. This should be a list of dictionaries containing hints to identify the drives. For example:

```
'skip_block_devices': [{'name': '/dev/vda', 'vendor': '0x1af4'}]
```

To prevent software RAID devices from being deleted, put their volume name (defined in the `target_raid_config`) to the list.

Note: one dictionary with one value for each of the logical disks. For example:

```
'skip_block_devices': [{'volume_name': 'large'}, {'volume_name': 'temp'}]
```

## Shared Disk Cluster Filesystems

Commonly used shared disk cluster filesystems, when detected, causes cleaning processes on stock hardware manager methods to abort prior to destroying the contents on the disk.

These filesystems include IBM General Parallel File System (GPFS), VmWare Virtual Machine File System (VMFS), and Red Hat Global File System (GFS2).

For information on troubleshooting, and disabling this check, see [Troubleshooting Ironic-Python-Agent \(IPA\)](#).



## 2.2.3 Custom hardware managers

### MellanoxDeviceHardwareManager

This is a custom hardware manager for ironic-python-agent. It provides support for Nvidia/Mellanox NICs.

- You can get the binraies firmware for all Nvidia/Mellanox NICs from here [Nvidia firmware downloads](#)
- And you can get the deviceID from here [Nvidia/Mellanox NICs list](#)
- Also you can check here [MFT documentation](#) for some supported parameters

### Clean steps

```
update_nvidia_nic_firmware_image(node, ports, images)
```

A clean step used to update Nvidia/Mellanox NICs firmware images from the required parameter `images` list. its disabled by default. Each image in the list is a dictionary with the following fields:

**url (required)**

The url of the firmware image (`file://`, `http://`).

**checksum (required)**

checksum of the provided image.

**checksumType (required)**

checksum type, it could be (md5/sha512/sha256).

**componentFlavor (required)**

The PSID of the nic.

**version (required)**

version of the firmware image , it must be the same as in the image file.

```
update_nvidia_nic_firmware_settings(node, ports, settings)
```

A clean step used to update Nvidia/Mellanox NICs firmware settings from the required parameter `settings` list. its disabled by default. Each settings in the list is a dictionary with the following fields:

**deviceID (required)**

The ID of the NIC

**globalConfig**

The global configuration for NIC

**function0Config**

The per-function configuration of the first port of the NIC

**function1Config**

The per-function configuration of the second port of the NIC

### Service steps

The Clean steps supported by the MellanoxDeviceHardwareManager are also available as Service steps if an infrastructure operator wishes to apply new firmware for a running machine.

### 2.2.4 Rescue mode

#### Overview

Rescue mode is a feature that can be used to boot a ramdisk for a tenant in case the machine is otherwise inaccessible. For example, if there's a disk failure that prevents access to another operating system, rescue mode can be used to diagnose and fix the problem.

#### Support in ironic-python-agent images

Rescue is initiated when ironic-conductor sends the `finalize_rescue` command to ironic-python-agent. A user `rescue` is created with a password provided as an argument to this command. DHCP is then configured to facilitate network connectivity, thus enabling a user to login to the machine in rescue mode.

#### Warning

Rescue mode exposes the contents of the ramdisk to the tenant. Ensure that any rescue image you build does not contain secrets (e.g. sensitive clean steps, proprietary firmware blobs).

The below has information about supported images that may be built to use rescue mode.

#### DIB

The DIB image supports rescue mode when used with DHCP tenant networks.

After the `finalize_rescue` command completes, DHCP will be configured on all network interfaces, and a `rescue` user will be created with the specified `rescue_password`.

#### TinyIPA

The TinyIPA image supports rescue mode when used with DHCP tenant networks. No special action is required to [build a TinyIPA image](#) with this support.

After the `finalize_rescue` command completes, DHCP will be configured on all network interfaces, and a `rescue` user will be created with the specified `rescue_password`.

### 2.2.5 Troubleshooting Ironic-Python-Agent (IPA)

This document contains basic trouble shooting information for IPA.

#### Gaining Access to IPA on a node

In order to access a running IPA instance a user must be added or enabled on the image. Below we will cover several ways to do this.

#### Access via ssh

## ironic-python-agent-builder

SSH access can be added to DIB built IPA images with the `dynamic-login`<sup>0</sup> or the `devuser` element<sup>1</sup>

The `dynamic-login` element allows the operator to inject a SSH key when the image boots. Kernel command line parameters are used to do this.

`dynamic-login` element example:

- Add `sshkey="ssh-rsa BBA1..."` to `kernel_append_params` setting in the `ironic.conf` file
- Restart the `ironic-conductor` with the command `service ironic-conductor restart`

Install `ironic-python-agent-builder` following the guide<sup>2</sup>

`devuser` element example:

```
export DIB_DEV_USER_USERNAME=username
export DIB_DEV_USER_PWDLESS_SUDO=yes
export DIB_DEV_USER_AUTHORIZED_KEYS=$HOME/.ssh/id_rsa.pub
ironic-python-agent-builder -o /path/to/custom-ipa -e devuser debian
```

## tinyipa

If you want to enable SSH access to the image, set `AUTHORIZE_SSH` variable in your shell to `true` before building the `tinyipa` image:

```
export AUTHORIZE_SSH=true
```

By default it will use default public RSA (or, if not available, DSA) key of the user running the build (`~/.ssh/id_{rsa,dsa}.pub`).

To provide other public SSH key, export full path to it in your shell before building `tinyipa` as follows:

```
export SSH_PUBLIC_KEY=/path/to/other/ssh/public/key
```

The user to use for access is default Tiny Core Linux user `tc`. This user has no password and has password-less `sudo` permissions. Installed SSH server is configured to disable Password authentication.

## Access via console

If you need to use console access, passwords must be enabled there are a couple ways to enable this depending on how the IPA image was created:

### ironic-python-agent-builder: dynamic-login

Users wishing to use password access can be add the `dynamic-login`<sup>0</sup> or the `devuser` element<sup>1</sup>

The `dynamic-login` element allows the operator to change the root password or SSH key dynamically when the image boots. Kernel command line parameters are used to do this.

Generate a password hash with following command:

<sup>0</sup> *Dynamic-login DIB element*: [https://github.com/openstack/diskimage-builder/tree/master/diskimage\\_builder/elements/dynamic-login](https://github.com/openstack/diskimage-builder/tree/master/diskimage_builder/elements/dynamic-login)

<sup>1</sup> *DevUser DIB element*: [https://github.com/openstack/diskimage-builder/tree/master/diskimage\\_builder/elements/devuser](https://github.com/openstack/diskimage-builder/tree/master/diskimage_builder/elements/devuser)

<sup>2</sup> *ironic-python-agent-builder*: <https://docs.openstack.org/ironic-python-agent-builder/latest/install/index.html>

```
$ openssl passwd -1 -stdin | sed 's/\$/\$\$/g'
```

Add `rootpwd=<openssl output>` value or add `sshkey=<ssh public key>` on the `kernel_append_params` setting in the Ironic configuration file (`/etc/ironic/ironic.conf`). Restart the `ironic-conductor` e.g. with

```
$ sudo systemctl restart ironic-conductor
```

### Warning

- The `sed` command is used to escape the `$` symbols in the configuration file.
- The quotation marks around the value are mandatory.
- Only 1 password or 1 SSH key is supported.

### ironic-python-agent-builder: devuser

Users can also be added to DIB built IPA images with the `devuser` element<sup>Page 15, 1</sup>. Install `ironic-python-agent-builder` following the guide<sup>Page 15, 2</sup>.

Example:

```
export DIB_DEV_USER_USERNAME=username
export DIB_DEV_USER_PWDLESS_SUDO=yes
export DIB_DEV_USER_PASSWORD=PASSWORD
ironic-python-agent-builder -o /path/to/custom-ipa -e devuser debian
```

### tinyipa

The image built with scripts provided in `tinyipa` folder of `Ironic Python Agent Builder` repository by default auto-logs the default Tiny Core Linux user `tc` to the console. This user has no password and has password-less `sudo` permissions.

### How to pause the IPA for debugging

When debugging issues with the IPA, in particular with cleaning, it may be necessary to log in to the RAM disk before the IPA actually starts (and delay the launch of the IPA). One easy way to do this is to set `maintenance` on the node and then trigger cleaning. Ironic will boot the node into the RAM disk, but the IPA will stall until the `maintenance` state is removed. This opens a time window to log into the node.

Another way to do this is to add simple cleaning steps in a custom hardware manager which sleep until a certain condition is met, e.g. until a given file exists. Having multiple of these barrier steps allows to go through the cleaning steps and have a break point in between them.

### Set IPA to debug logging

Debug logging can be enabled a several different ways. The easiest way is to add `ipa-debug=1` to the kernel command line. To do this:

- Append `ipa-debug=1` to the `kernel_append_params` setting in the `ironic.conf` file

- Restart the ironic-conductor with the command `service ironic-conductor restart`

If the system is running and uses `systemd` then editing the services file will be required.

- `systemctl edit ironic-python-agent.service`
- Append `--debug` to end of the `ExecStart` command
- Restart IPA. See the *Manually restart IPA* section below.

### Where can I find the IPA logs

Retrieving the IPA logs will differ depending on which base image was used.

- Operating system that do not use `systemd` (ie Ubuntu 14.04)
  - logs will be found in the `/var/log/` folder.
- Operating system that do use `systemd` (ie Fedora, CentOS, RHEL)
  - logs may be viewed with `sudo journalctl -u ironic-python-agent`
  - if using a diskimage-builder ramdisk, it may be configured to output all contents of the journal, including ironic-python-agent logs, by enabling the `journal-to-console` element.

In addition, Ironic is configured to retrieve IPA logs upon failures by default, you can learn more about this feature in the [Ironic troubleshooting guide](#).

### Manually restart IPA

In some cases it is helpful to enable debug mode on a running node. If the system does not use `systemd` then IPA can be restarted directly:

```
sudo /usr/local/bin/ironic-python-agent [--debug]
```

If the system uses `systemd` then `systemctl` can be used to restart the service:

```
sudo systemctl restart ironic-python-agent.service
```

### Cleaning halted with ProtectedDeviceError

The IPA service has halted cleaning as one of the block devices within or attached to the bare metal node contains a class of filesystem which **MAY** cause irreparable harm to a potentially running cluster if accidentally removed.

These filesystems *may* be used for only local storage and as a result be safe to erase. However if a shared block device is in use, such as a device supplied via a Storage Area Network utilizing protocols such as iSCSI or FibreChannel. Ultimately the Host Bus Adapter (HBA) may not be an entirely detectable entity given the hardware market place and aspects such as SmartNICs and Converged Network Adapters with specific offload functions to support standards like NVMe over Fabric (NVMe-oF).

By default, the agent will prevent these filesystems from being deleted and will halt the cleaning process when detected. The cleaning process can be re-triggered via Ironics state machine once one of the documented settings have been used to notify the agent that no action is required.

## What filesystems are looked for

IBM General Parallel Filesystem
Red Hat Global Filesystem 2
VmWare Virtual Machine FileSystem (VMFS)

## Im okay with deleting, how do I tell IPA to clean the disk(s)?

Four potential ways exist to signal to IPA. Please note, all of these options require access either to the node in Ironics API or ability to modify Ironic configuration.

### Via Ironic

#### Note

This option requires that the version of Ironic be sufficient enough to understand and explicitly provide this option to the Agent.

Inform Ironic to provide the option to the Agent:

```
baremetal node set --driver-info wipe_special_filesystems=True
```

### Via a nodes kernel\_append\_params setting

This may be set on a node level by utilizing the override kernel\_append\_params setting which can be utilized on a node level. Example:

```
baremetal node set --driver-info kernel_append_params="ipa-guard-special-  
↪filesystems=False"
```

Alternatively, if you wish to set this only once, you may use the instance\_info field, which is wiped upon teardown of the node. Example:

```
baremetal node set --instance-info kernel_append_params="ipa-guard-special-  
↪filesystems=False"
```

### Via Ironics Boot time PXE parameters (Globally)

Globally, this setting may be passed by modifying the ironic.conf configuration file on your cluster by adding ipa-guard-special-filefilesystems=False string to the [pxe]kernel\_append\_params parameter.

#### Warning

If your running a multi-conductor deployment, all of your ironic.conf configuration files will need to be updated to match.

## Via Ramdisk configuration

This option requires modifying the ramdisk, and is the most complex, but may be advisable if you have a mixed environment cluster where shared clustered filesystems may be a concern on some machines, but not others.

### Warning

This requires rebuilding your agent ramdisk, and modifying the embedded configuration file for the `ironic-python-agent`. If your confused at all by this statement, this option is not for you.

Edit `/etc/ironic_python_agent/ironic_python_agent.conf` and set the parameter `[DEFAULT]guard_special_filesystems` to `False`.

## References

## 2.3 Contributing to Ironic Python Agent

Ironic Python Agent is an agent for controlling and deploying Ironic controlled baremetal nodes. Typically run in a ramdisk, the agent exposes a REST API for provisioning servers.

Throughout the remainder of the document, Ironic Python Agent will be abbreviated to IPA.

### 2.3.1 Hardware Managers

Hardware managers are how IPA supports multiple different hardware platforms in the same agent. Any action performed on hardware can be overridden by deploying your own hardware manager.

IPA ships with *GenericHardwareManager*, which implements basic cleaning and deployment methods compatible with most hardware.

### Warning

Some functionality inherent in the stock hardware manager cleaning methods may be useful in custom hardware managers, but should not be inherently expected to also work in custom managers. Examples of this are clustered filesystem protections, and cleaning method fallback logic. Custom hardware manager maintainers should be mindful when overriding the stock methods.

### How are methods executed on HardwareManagers?

Methods that modify hardware are dispatched to each hardware manager in priority order. When a method is dispatched, if a hardware manager does not have a method by that name or raises *IncompatibleHardwareMethodError*, IPA continues on to the next hardware manager. Any hardware manager that returns a result from the method call is considered a success and its return value passed on to whatever dispatched the method. If the method is unable to run successfully on any hardware managers, *HardwareManagerMethodNotFound* is raised.

## Why build a custom HardwareManager?

Custom hardware managers allow you to include hardware-specific tools, files and cleaning steps in the Ironic Python Agent. For example, you could include a BIOS flashing utility and BIOS file in a custom ramdisk. Your custom hardware manager could expose a cleaning step that calls the flashing utility and flashes the packaged BIOS version (or even download it from a tested web server).

## How can I build a custom HardwareManager?

In general, custom HardwareManagers should subclass `hardware.HardwareManager`. Subclassing `hardware.GenericHardwareManager` should only be considered if the aim is to raise the priority of all methods of the `GenericHardwareManager`. The only required method is `evaluate_hardware_support()`, which should return one of the enums in `hardware.HardwareSupport`. Hardware support determines which hardware manager is executed first for a given function (see: *How are methods executed on HardwareManagers?* for more info). Common methods you may want to implement are `list_hardware_info()`, to add additional hardware the `GenericHardwareManager` is unable to identify and `erase_devices()`, to erase devices in ways other than ATA secure erase or shredding.

Some reusable functions are provided by `ironic-lib`, its IPA is relatively stable.

The `examples` directory has two example hardware managers that can be copied and adapter for your use case.

## Custom HardwareManagers and Cleaning

One of the reasons to build a custom hardware manager is to expose extra steps in `Ironic Cleaning`. A node will perform a set of cleaning steps any time the node is deleted by a tenant or moved from `manageable` state to `available` state. Ironic will query IPA for a list of clean steps that should be executed on the node. IPA will dispatch a call to `get_clean_steps()` on all available hardware managers and then return the combined list to Ironic.

To expose extra clean steps, the custom hardware manager should have a function named `get_clean_steps()` which returns a list of dictionaries. The dictionaries should be in the form:

```
def get_clean_steps(self, node, ports):
    return [
        {
            # A function on the custom hardware manager
            'step': 'upgrade_firmware',
            # An integer priority. Largest priorities are executed first
            'priority': 10,
            # Should always be the deploy interface
            'interface': 'deploy',
            # Request the node to be rebooted out of band by Ironic when
            # the step completes successfully
            'reboot_requested': False
        }
    ]
```

Then, you should create functions which match each of the `step` keys in the clean steps you return. The functions will take two parameters: `node`, a dictionary representation of the Ironic node, and `ports`, a list of dictionary representations of the Ironic ports attached to `node`.

When a clean step is executed in IPA, the `step` key will be sent to the hardware managers in hardware support order, using `hardware.dispatch_to_managers()`. For each hardware manager, if the manager has



a function matching the *step* key, it will be executed. If the function returns a value (including None), that value is returned to Ironic and no further managers are called. If the function raises *IncompatibleHardwareMethodError*, the next manager will be called. If the function raises any other exception, the command will be considered failed, the command results error message will be set to the exceptions error message, and no further managers will be called. An example step:

```
def upgrade_firmware(self, node, ports):
    if self._device_exists():
        # Do the upgrade
        return 'upgraded firmware'
    else:
        raise errors.IncompatibleHardwareMethodError()
```

#### Note

If creating a new step, the value returned must be serializable into an API response and log message. If replacing an existing step, you should return a response of the same type and style of the upstream step.

If the step has args, you need to add them to argsinfo and provide the function with extra parameters.

```
def get_clean_steps(self, node, ports):
    return [
        {
            # A function on the custom hardware manager
            'step': 'upgrade_firmware',
            # An integer priority. Largest priorities are executed first
            'priority': 10,
            # Should always be the deploy interface
            'interface': 'deploy',
            # Arguments that can be required or optional.
            'argsinfo': {
                'firmware_url': {
                    'description': 'Url for firmware',
                    'required': True,
                },
            },
            # Request the node to be rebooted out of band by Ironic when
            # the step completes successfully
            'reboot_requested': False
        }
    ]
```

```
def upgrade_firmware(self, node, ports, firmware_url):
    if self._device_exists():
        # Do the upgrade
        return 'upgraded firmware'
    else:
        raise errors.IncompatibleHardwareMethodError()
```

**Note**

If two managers return steps with the same *step* key, the priority will be set to whichever manager has a higher hardware support level and then use the higher priority in the case of a tie.

In some cases, it may be necessary to create a customized cleaning step to take a particular pattern of behavior. Those doing such work may want to leverage file system safety checks, which are part of the stock hardware managers.

```
def custom_erase_devices(self, node, ports):
    for dev in determine_my_devices_to_erase():
        hardware.safety_check_block_device(node, dev.name)
        my_special_cleaning(dev.name)
```

## Custom HardwareManagers and Deploying

Starting with the Victoria release cycle, [deployment](#) can be customized similarly to [cleaning](#). A hardware manager can define *deploy steps* that may be run during deployment by exposing a `get_deploy_steps` call.

There are two kinds of deploy steps:

1. Steps that need to be run automatically must have a non-zero priority and cannot take required arguments. For example:

```
def get_deploy_steps(self, node, ports):
    return [
        {
            # A function on the custom hardware manager
            'step': 'upgrade_firmware',
            # An integer priority. Largest priorities are executed first
            'priority': 10,
            # Should always be the deploy interface
            'interface': 'deploy',
        }
    ]

# A deploy steps looks the same as a clean step.
def upgrade_firmware(self, node, ports):
    if self._device_exists():
        # Do the upgrade
        return 'upgraded firmware'
    else:
        raise errors.IncompatibleHardwareMethodError()
```

Priority should be picked based on when exactly in the process the step will run. See [agent step priorities](#) for guidance.

2. Steps that will be requested via [deploy templates](#) should have a priority of 0 and may take both required and optional arguments that will be provided via the deploy templates. For example:

```

def get_deploy_steps(self, node, ports):
    return [
        {
            # A function on the custom hardware manager
            'step': 'write_a_file',
            # Steps with priority 0 don't run by default.
            'priority': 0,
            # Should be the deploy interface, unless there is driver-side
            # support for another interface (as it is for RAID).
            'interface': 'deploy',
            # Arguments that can be required or optional.
            'argsinfo': {
                'path': {
                    'description': 'Path to file',
                    'required': True,
                },
                'content': {
                    'description': 'Content of the file',
                    'required': True,
                },
                'mode': {
                    'description': 'Mode of the file, defaults to 0644',
                    'required': False,
                },
            },
        }
    ]

def write_a_file(self, node, ports, path, contents, mode=0o644):
    pass # Mount the disk, write a file.

```

## Custom HardwareManagers and Service operations

Starting with the Bobcat release cycle, A hardware manager can define *service steps* that may be run during a service operation by exposing a `get_service_steps` call.

Service steps are intended to be invoked by an operator to perform an ad-hoc action upon a node. This does not include automatic step execution, but may at some point in the future. The result is that steps can be exposed similar to Clean steps and Deploy steps, just the priority value, should be 0 as the user requested order is what is utilized.

```

def get_deploy_steps(self, node, ports):
    return [
        {
            # A function on the custom hardware manager
            'step': 'write_a_file',
            # Steps with priority 0 don't run by default.
            'priority': 0,
            # Should be the deploy interface, unless there is driver-side
            # support for another interface (as it is for RAID).
            'interface': 'deploy',

```

(continues on next page)

(continued from previous page)

```
# Arguments that can be required or optional.
'argsinfo': {
    'path': {
        'description': 'Path to file',
        'required': True,
    },
    'content': {
        'description': 'Content of the file',
        'required': True,
    },
    'mode': {
        'description': 'Mode of the file, defaults to 0644',
        'required': False,
    },
}
}
]

def write_a_file(self, node, ports, path, contents, mode=0o644):
    pass # Mount the disk, write a file.
```

## Versioning

Each hardware manager has a name and a version. This version is used during cleaning to ensure the same version of the agent is used to on a node through the entire process. If the version changes, cleaning is restarted from the beginning to ensure consistent cleaning operations and to make updating the agent in production simpler.

You can set the version of your hardware manager by creating a class variable named `HARDWARE_MANAGER_VERSION`, which should be a string. The default value is 1.0. You should change this version string any time you update your hardware manager. You can also change the name your hardware manager presents by creating a class variable called `HARDWARE_MANAGER_NAME`, which is a string. The name defaults to the class name. Currently IPA only compares version as a string; any version change whatsoever will induce cleaning to restart.

## Priority

A hardware manager has a single overall priority, which should be based on how well it supports a given piece of hardware. At load time, IPA executes `evaluate_hardware_support()` on each hardware manager. This method should return an int representing hardware manager priority, based on what it detects about the platform its running on. Suggested values are included in the `HardwareSupport` class. Returning a value of 0 aka `HardwareSupport.NONE`, will prevent the hardware manager from being used. IPA will never ship a hardware manager with a priority higher than 3, aka `HardwareSupport.SERVICE_PROVIDER`.

### 2.3.2 Emitting metrics from Ironic-Python-Agent (IPA)

#### Warning

IPA metrics are deprecated and scheduled for removal at or after the 2026.1 OpenStack release cycle.

This document describes how to emit metrics from IPA, including timers and counters in code to directly emitting hardware metrics from a custom HardwareManager.

## Overview

IPA uses a vendored version of the metrics implementation originally from ironic-lib, with a few caveats due to the dynamic configuration done at lookup time. You cannot cache the metrics instance as the MetricsLogger returned will change after lookup if configs different than the default setting have been used. This also means that the method decorator cannot be used in IPA.

## Using a context manager

Using the context manager is the recommended way for sending metrics that time or count sections of code. However, given that you cannot cache the MetricsLogger, you have to explicitly call `get_metrics_logger()` from every time. For example:

```
from ironic_python_agent.metrics_lib import metrics_utils

def my_method():
    with metrics_utils.get_metrics_logger(__name__).timer('my_method'):
        return _do_work()
```

As a note, these metric collectors do work for custom HardwareManagers as well. However, you may want to metric the portions of a method that determine compatibility separate from portions of a method that actually do work, in order to assure the metrics are relevant and useful on all hardware.

## Explicitly sending metrics

A feature that may be particularly helpful for deployers writing custom HardwareManagers is the ability to explicitly send metrics. For instance, you could add a cleaning step which would retrieve metrics about a device and ship them using the provided metrics library. For example:

```
from ironic_python_agent.metrics_lib import metrics_utils

def my_cleaning_step():
    for name, value in _get_smart_data():
        metrics_utils.get_metrics_logger(__name__).send_gauge(name, value)
```

### 2.3.3 Rescue Mode

Ironic supports putting nodes in rescue mode using hardware types that support rescue interfaces. A rescue operation can be used to boot nodes into a rescue ramdisk so that the rescue user can access the node. This provides the ability to access the node when normal access is not possible. For example, if there is a need to perform manual password reset or data recovery in the event of some failure, a rescue operation can be used. IPA rescue extension exposes a command `finalize_rescue` (that is used by Ironic) to set the password for the rescue user when the rescue ramdisk is booted.

### finalize\_rescue command

The rescue extension exposes the command `finalize_rescue`; when invoked, it triggers rescue mode:

```
POST /v1/commands
{"name": "rescue.finalize_rescue",
 "params": {
   "rescue_password": "p455w0rd"
 }
}
```

`rescue_password` is a required parameter for this command.

Upon success, it returns following data in response:

```
{"command_name": "finalize_rescue",
 "command_params": {
   "rescue_password": "p455w0rd"
 },
 "command_status": "SUCCEEDED"
 "command_result": null
 "command_error": null
}
```

If successful, this synchronous command will:

1. Write the salted and encrypted `rescue_password` to `/etc/ipa-rescue-config/ipa-rescue-password` in the chroot or filesystem that `ironic-python-agent` is running in.
2. Stop the `ironic-python-agent` process after completing these actions and returning the response to the API request.

### 2.3.4 Generated Developer Documentation

- [modindex](#)

#### [ironic\\_python\\_agent](#)

#### [ironic\\_python\\_agent package](#)

#### [Subpackages](#)

#### [ironic\\_python\\_agent.api package](#)

#### [Submodules](#)

#### [ironic\\_python\\_agent.api.app module](#)

**class** `ironic_python_agent.api.app.Application`(*agent*, *conf*)

Bases: `object`

**api\_get\_command**(*request*, *cmd*)

**api\_list\_commands**(*request*)

**api\_root**(*request*)

**api\_run\_command**(*request*)

**api\_status**(*request*)

**api\_v1**(*request*)

**handle\_exception**(*environ, exc*)

Handle an exception during request processing.

**start**(*tls\_cert\_file=None, tls\_key\_file=None*)

Start the API service in the background.

**stop**()

Stop the API service.

**class** `ironic_python_agent.api.app.Request`(*environ: WSGIEnvironment, populate\_request: bool = True, shallow: bool = False*)

Bases: Request

Custom request class with JSON support.

`ironic_python_agent.api.app.format_exception`(*value*)

`ironic_python_agent.api.app.jsonify`(*value, status=200*)

Convert value to a JSON response using the custom encoder.

`ironic_python_agent.api.app.make_link`(*url, rel\_name, resource="", resource\_args="", bookmark=False, type\_=None*)

`ironic_python_agent.api.app.version`(*url*)

## Module contents

### `ironic_python_agent.cmd` package

#### Submodules

#### `ironic_python_agent.cmd.agent` module

`ironic_python_agent.cmd.agent.run`()

Entrypoint for IronicPythonAgent.

#### `ironic_python_agent.cmd.inspect` module

`ironic_python_agent.cmd.inspect.run`()

Entrypoint for IronicPythonAgent.

## Module contents

### `ironic_python_agent.extensions` package

#### Submodules

## ironic\_python\_agent.extensions.base module

**class** `ironic_python_agent.extensions.base.AgentCommandStatus`

Bases: `object`

Mapping of agent command statuses.

**FAILED** = 'FAILED'

**RUNNING** = 'RUNNING'

**SUCCEEDED** = 'SUCCEEDED'

**VERSION\_MISMATCH** = 'CLEAN\_VERSION\_MISMATCH'

**class** `ironic_python_agent.extensions.base.AsyncCommandResult`(*command\_name*,  
*command\_params*,  
*execute\_method*,  
*agent=None*)

Bases: `BaseCommandResult`

A command that executes asynchronously in the background.

**is\_done**()

Checks to see if command is still RUNNING.

**Returns**

True if command is done, False if still RUNNING

**join**(*timeout=None*)

Block until command has completed, and return result.

**Parameters**

**timeout** float indicating max seconds to wait for command to complete. Defaults to None.

**run**()

Run a command.

**serialize**()

Serializes the AsyncCommandResult into a dict.

**Returns**

dict containing serializable fields in AsyncCommandResult

**start**()

Begin background execution of command.

**class** `ironic_python_agent.extensions.base.BaseAgentExtension`(*agent=None*)

Bases: `object`

**check\_cmd\_presence**(*ext\_obj*, *ext*, *cmd*)

**execute**(*command\_name*, *\*\*kwargs*)



```
class ironic_python_agent.extensions.base.BaseCommandResult(command_name,  
                                                         command_params)
```

Bases: *SerializableComparable*

Base class for command result.

```
is_done()
```

Checks to see if command is still RUNNING.

**Returns**

True if command is done, False if still RUNNING

```
join()
```

**Returns**

result of completed command.

```
serializable_fields = ('id', 'command_name', 'command_status',  
                        'command_error', 'command_result')
```

```
wait()
```

Join the result and extract its value.

Raises if the command failed.

```
class ironic_python_agent.extensions.base.ExecuteCommandMixin
```

Bases: object

```
execute_command(command_name, **kwargs)
```

Execute an agent command.

```
get_extension(extension_name)
```

```
split_command(command_name)
```

```
class ironic_python_agent.extensions.base.SyncCommandResult(command_name,  
                                                         command_params,  
                                                         success,  
                                                         result_or_error)
```

Bases: *BaseCommandResult*

A result from a command that executes synchronously.

```
ironic_python_agent.extensions.base.async_command(command_name, validator=None)
```

Will run the command in an AsyncCommandResult in its own thread.

*command\_name* is set based on the func name and *command\_params* will be whatever args/kwargs you pass into the decorated command. Return values of type *str* or *unicode* are prefixed with the *command\_name* parameter when returned for consistency.

```
ironic_python_agent.extensions.base.get_extension(name)
```

```
ironic_python_agent.extensions.base.init_ext_manager(agent)
```

```
ironic_python_agent.extensions.base.sync_command(command_name, validator=None)
```

Decorate a method to wrap its return value in a SyncCommandResult.

For consistency with `@async_command()` can also accept a validator which will be used to validate input, although a synchronous command can also choose to implement validation inline.

## ironic\_python\_agent.extensions.clean module

**class** `ironic_python_agent.extensions.clean.CleanExtension`(*agent=None*)

Bases: *BaseAgentExtension*

**execute\_clean\_step**(*step, node, ports, clean\_version=None, \*\*kwargs*)

Execute a clean step.

### Parameters

- **step** A clean step with step, priority and interface keys
- **node** A dict representation of a node
- **ports** A dict representation of ports attached to node
- **clean\_version** The clean version as returned by `hardware.get_current_versions()` at the beginning of cleaning/zapping

### Returns

a `CommandResult` object with `command_result` set to whatever the step returns.

**get\_clean\_steps**(*node, ports*)

Get the list of clean steps supported for the node and ports

### Parameters

- **node** A dict representation of a node
- **ports** A dict representation of ports attached to node

### Returns

A list of clean steps with keys `step`, `priority`, and `reboot_requested`

## ironic\_python\_agent.extensions.deploy module

**class** `ironic_python_agent.extensions.deploy.DeployExtension`(*agent=None*)

Bases: *BaseAgentExtension*

**execute\_deploy\_step**(*step, node, ports, deploy\_version=None, \*\*kwargs*)

Execute a deploy step.

### Parameters

- **step** A deploy step with step, priority and interface keys
- **node** A dict representation of a node
- **ports** A dict representation of ports attached to node
- **deploy\_version** The deploy version as returned by `hardware.get_current_versions()` at the beginning of deploying.
- **kwargs** The remaining arguments are passed to the step.

### Returns

a `CommandResult` object with `command_result` set to whatever the step returns.

**get\_deploy\_steps**(*node, ports*)

Get the list of deploy steps supported for the node and ports

**Parameters**

- **node** A dict representation of a node
- **ports** A dict representation of ports attached to node

**Returns**

A list of deploy steps with keys step, priority, and reboot\_requested

**ironic\_python\_agent.extensions.flow module**

```
class ironic_python_agent.extensions.flow.FlowExtension(agent=None)
```

Bases: *BaseAgentExtension*, *ExecuteCommandMixin*

```
start_flow(flow=None)
```

**ironic\_python\_agent.extensions.image module**

```
class ironic_python_agent.extensions.image.ImageExtension(agent=None)
```

Bases: *BaseAgentExtension*

```
install_bootloader(root_uuid, efi_system_part_uuid=None, prep_boot_part_uuid=None,
                   target_boot_mode='bios', ignore_bootloader_failure=None)
```

Install the GRUB2 bootloader on the image.

**Parameters**

- **root\_uuid** The UUID of the root partition.
- **efi\_system\_part\_uuid** The UUID of the efi system partition. To be used only for uefi boot mode. For uefi boot mode, the boot loader will be installed here.
- **prep\_boot\_part\_uuid** The UUID of the PReP Boot partition. Used only for booting ppc64\* partition images locally. In this scenario the bootloader will be installed here.
- **target\_boot\_mode** bios, uefi. Only taken into account for softraid, when no efi partition is explicitly provided (happens for whole disk images)

**Raises**

CommandExecutionError if the installation of the bootloader fails.

**Raises**

DeviceNotFound if the root partition is not found.

**ironic\_python\_agent.extensions.log module**

```
class ironic_python_agent.extensions.log.LogExtension(agent=None)
```

Bases: *BaseAgentExtension*

```
collect_system_logs()
```

Collect system logs.

Collect and package diagnostic and support data from the ramdisk.

**Raises**

CommandExecutionError if failed to collect the system logs.

### Returns

A dictionary with the key `system_logs` and the value of a gzipped and base64 encoded string of the file with the logs.

### `ironic_python_agent.extensions.poll` module

**class** `ironic_python_agent.extensions.poll.PollExtension`(*agent=None*)

Bases: `BaseAgentExtension`

**get\_hardware\_info**()

Get the hardware information where IPA is running.

**set\_node\_info**(*node\_info=None*)

Set node lookup data when IPA is running at passive mode.

### Parameters

**node\_info** A dictionary contains the information of the node where IPA is running.

### `ironic_python_agent.extensions.rescue` module

**class** `ironic_python_agent.extensions.rescue.RescueExtension`(*agent=None*)

Bases: `BaseAgentExtension`

**finalize\_rescue**(*rescue\_password="", hashed=False*)

Sets the rescue password for the rescue user.

**write\_rescue\_password**(*rescue\_password="", hashed=False*)

Write rescue password to a file for use after IPA exits.

### Parameters

- **rescue\_password** Rescue password.
- **hashed** Boolean default False indicating if the password being provided is hashed or not. This will be changed in a future version of ironic.

### `ironic_python_agent.extensions.service` module

**class** `ironic_python_agent.extensions.service.ServiceExtension`(*agent=None*)

Bases: `BaseAgentExtension`

**execute\_service\_step**(*step, node, ports, service\_version=None, \*\*kwargs*)

Execute a service step.

### Parameters

- **step** A step with step, priority and interface keys
- **node** A dict representation of a node
- **ports** A dict representation of ports attached to node
- **service\_version** The service version as returned by `hardware.get_current_versions()` at the beginning of the service operation.

### Returns

a `CommandResult` object with `command_result` set to whatever the step returns.

**get\_service\_steps**(*node, ports*)

Get the list of service steps supported for the node and ports

**Parameters**

- **node** A dict representation of a node
- **ports** A dict representation of ports attached to node

**Returns**

A list of service steps with keys step, priority, and reboot\_requested

### **ironic\_python\_agent.extensions.standby module**

**class** `ironic_python_agent.extensions.standby.ImageDownload`(*image\_info*,  
*time\_obj=None*)

Bases: `object`

Helper class that opens a HTTP connection to download an image.

This class opens a HTTP connection to download an image from a URL and create an iterator so the image can be downloaded in chunks. The MD5 hash of the image being downloaded is calculated on-the-fly.

**property bytes\_transferred**

Property value to return the number of bytes transferred.

**property content\_length**

Property value to return the server indicated length.

**verify\_image**(*image\_location*)

Verifies the checksum of the local images matches expectations.

If this function does not raise `ImageChecksumError` then it is very likely that the local copy of the image was transmitted and stored correctly.

**Parameters**

**image\_location** The location of the local image.

**Raises**

`ImageChecksumError` if the checksum of the local image does not match the checksum as reported by glance in *image\_info*.

**class** `ironic_python_agent.extensions.standby.StandbyExtension`(*agent=None*)

Bases: `BaseAgentExtension`

Extension which adds stand-by related functionality to agent.

**get\_partition\_uuids**()

Return partition UUIDs.

**power\_off**()

Powers off the agents system.

**prepare\_image**(*image\_info, configdrive=None*)

Asynchronously prepares specified image on local OS install device.

In this case, prepare means make local machine completely ready to reboot to the image specified by *image\_info*.

Downloads and writes an image to disk if necessary. Also writes a configdrive to disk if the configdrive parameter is specified.

### Parameters

- **image\_info** Image information dictionary.
- **configdrive** A string containing the location of the config drive as a URL OR the contents (as gzip/base64) of the configdrive. Optional, defaults to None.

### Raises

ImageDownloadError if the image download encounters an error.

### Raises

ImageChecksumError if the checksum of the local image does not match the checksum as reported by glance in image\_info.

### Raises

ImageWriteError if writing the image fails.

### Raises

InstanceDeployFailure if failed to create config drive. large to store on the given device.

### `run_image()`

Runs image on agents system via reboot.

### `sync()`

Flush file system buffers forcing changed blocks to disk.

### Raises

CommandExecutionError if flushing file system buffers fails.

### `ironic_python_agent.extensions.standby.check_md5_enabled()`

Checks if md5 is permitted, otherwise raises ValueError.

## `ironic_python_agent.extensions.system` module

`class ironic_python_agent.extensions.system.SystemExtension(agent=None)`

Bases: *BaseAgentExtension*

### `lockdown()`

Lock the agent down to prevent interactions with it.

## Module contents

### `ironic_python_agent.hardware_managers` package

#### Submodules

### `ironic_python_agent.hardware_managers.cna` module

`class ironic_python_agent.hardware_managers.cna.IntelCnaHardwareManager`

Bases: *HardwareManager*

```
HARDWARE_MANAGER_NAME = 'IntelCnaHardwareManager'
```

```
HARDWARE_MANAGER_VERSION = '1.0'
```

```
evaluate_hardware_support()
```

### ironic\_python\_agent.hardware\_managers.mlnx module

```
class ironic_python_agent.hardware_managers.mlnx.MellanoxDeviceHardwareManager
```

Bases: *HardwareManager*

Mellanox hardware manager to support a single device

```
HARDWARE_MANAGER_NAME = 'MellanoxDeviceHardwareManager'
```

```
HARDWARE_MANAGER_VERSION = '1'
```

```
evaluate_hardware_support()
```

Declare level of hardware support provided.

```
get_clean_steps(node, ports)
```

Get a list of clean steps with priority.

#### Parameters

- **node** The node object as provided by Ironic.
- **ports** Port objects as provided by Ironic.

#### Returns

A list of cleaning steps, as a list of dicts.

```
get_deploy_steps(node, ports)
```

Alias wrapper for method `get_clean_steps`.

```
get_interface_info(interface_name)
```

Return the interface information when its Mellanox and InfiniBand

#### In case of Mellanox and InfiniBand interface we do the following:

1. Calculate the InfiniBand MAC according to InfiniBand GUID
2. Calculate the client-id according to InfiniBand GUID

```
get_service_steps(node, ports)
```

Alias wrapper for method `get_clean_steps`.

```
update_nvidia_nic_firmware_image(node, ports, images)
```

```
update_nvidia_nic_firmware_settings(node, ports, settings)
```

### Module contents

#### Submodules

#### ironic\_python\_agent.agent module

**class** `ironic_python_agent.agent.Host`(*hostname, port*)

Bases: `tuple`

**hostname**

Alias for field number 0

**port**

Alias for field number 1

**class** `ironic_python_agent.agent.IronicPythonAgent`(*api\_url, advertise\_address, listen\_address, ip\_lookup\_attempts, ip\_lookup\_sleep, network\_interface, lookup\_timeout, lookup\_interval, standalone, agent\_token, hardware\_initialization\_delay=0, advertise\_protocol='http'*)

Bases: `ExecuteCommandMixin`

Class for base agent functionality.

**force\_heartbeat**()

**classmethod** `from_config`(*conf*)

**get\_command\_result**(*result\_id*)

Get a specific command result by ID.

**Returns**

a `ironic_python_agent.extensions.base.BaseCommandResult` object.

**Raises**

`RequestedObjectNotFoundError` if command with the given ID is not found.

**get\_node\_uuid**()

Get UUID for Ironic node.

If the agent has not yet heartbeated to Ironic, it will not have the UUID and this will raise an exception.

**Returns**

A string containing the UUID for the Ironic node.

**Raises**

`UnknownNodeError` if UUID is unknown.

**get\_status**()

Retrieve a serializable status.

**Returns**

a `ironic_python_agent.agent.IronicPythonAgent` instance describing the agents status.

**list\_command\_results**()

Get a list of command results.



**Returns**

list of `ironic_python_agent.extensions.base.BaseCommandResult` objects.

**process\_lookup\_data**(*content*)

Update agent configuration from lookup data.

**run**()

Run the Ironic Python Agent.

**serve\_ipa\_api**()

Serve the API until an extension terminates it.

**set\_agent\_advertise\_addr**()

Set advertised IP address for the agent, if not already set.

If agents advertised IP address is still default (None), try to find a better one. If the agents network interface is None, replace that as well.

**Raises**

`LookupAgentIPError` if an IP address could not be found

**validate\_agent\_token**(*token*)**class** `ironic_python_agent.agent.IronicPythonAgentHeartbeater`(*agent*)

Bases: `Thread`

Thread that periodically heartbeats to Ironic.

**do\_heartbeat**()

Send a heartbeat to Ironic.

**force\_heartbeat**()

`max_error_jitter_multiplier = 2.0`

`max_jitter_multiplier = 0.6`

`min_error_jitter_multiplier = 1.0`

`min_heartbeat_interval = 5`

`min_jitter_multiplier = 0.3`

**run**()

Start the heartbeat thread.

**stop**()

Stop the heartbeat thread.

**class** `ironic_python_agent.agent.IronicPythonAgentStatus`(*started\_at*, *version*)

Bases: `Serializable`

Represents the status of an agent.

`serializable_fields = ('started_at', 'version')`

## ironic\_python\_agent.burnin module

`ironic_python_agent.burnin.fio_disk(node)`

Burn-in the disks with fio

Run an fio randrw job for a configurable number of iterations or a given amount of time.

### Parameters

**node** Ironic node object

### Raises

CommandExecutionError if the execution of fio fails.

`ironic_python_agent.burnin.fio_network(node)`

Burn-in the network with fio

Run an fio network job for a pair of nodes for a configurable amount of time. The pair is either statically defined in driver\_info via agent\_burnin\_fio\_network\_config or the role and partner is found dynamically via a tooz backend.

The writer will wait for the reader to connect, then write to the network. Upon completion, the roles are swapped.

### Parameters

**node** Ironic node object

### Raises

CommandExecutionError if the execution of fio fails.

### Raises

CleaningError if the configuration is incomplete.

`ironic_python_agent.burnin.stress_ng_cpu(node)`

Burn-in the CPU with stress-ng

Run stress-ng on a configurable number of CPUs for a configurable amount of time. Without config use all CPUs and stress them for 24 hours.

### Parameters

**node** Ironic node object

### Raises

CommandExecutionError if the execution of stress-ng fails.

`ironic_python_agent.burnin.stress_ng_vm(node)`

Burn-in the memory with the vm stressor in stress-ng

Run stress-ng with a configurable number of workers on a configurable amount of the available memory for a configurable amount of time. Without config use as many workers as CPUs, 98% of the memory and stress it for 24 hours.

### Parameters

**node** Ironic node object

### Raises

CommandExecutionError if the execution of stress-ng fails.

## ironic\_python\_agent.config module

`ironic_python_agent.config.list_opts()`

`ironic_python_agent.config.override(params)`

Override configuration with values from a dictionary.

This is used for configuration overrides from mDNS.

### Parameters

**params** new configuration parameters as a dict.

`ironic_python_agent.config.populate_config()`

Populate configuration. In a method so tests can easily utilize it.

## ironic\_python\_agent.disk\_partitioner module

Code for creating partitions on a disk.

Imported from ironic-libs disk\_utils as of the following commit: <https://opendev.org/openstack/ironic-lib/commit/42fa5d63861ba0f04b9a4f67212173d7013a1332>

```
class ironic_python_agent.disk_partitioner.DiskPartitioner(device,  
                                                         disk_label='msdos',  
                                                         alignment='optimal')
```

Bases: object

**add\_partition**(size, part\_type='primary', fs\_type="", boot\_flag=None, extra\_flags=None)

Add a partition.

### Parameters

- **size** The size of the partition in MiB.
- **part\_type** The type of the partition. Valid values are: primary, logical, or extended.
- **fs\_type** The filesystem type. Valid types are: ext2, fat32, fat16, HFS, linux-swap, NTFS, reiserfs, ufs. If blank (), it will create a Linux native partition (83).
- **boot\_flag** Boot flag that needs to be configured on the partition. Ignored if None. It can take values bios\_grub, boot.
- **extra\_flags** List of flags to set on the partition. Ignored if None.

### Returns

The partition number.

**commit()**

Write to the disk.

**get\_partitions()**

Get the partitioning layout.

### Returns

An iterator with the partition number and the partition layout.

### `ironic_python_agent.disk_utils` module

Various utilities related to disk handling.

Imported from `ironic-libs disk_utils` as of the following commit: <https://opendev.org/openstack/ironic-lib/commit/42fa5d63861ba0f04b9a4f67212173d7013a1332>

`ironic_python_agent.disk_utils.block_uuid(dev)`

Get UUID of a block device.

Try to fetch the UUID, if that fails, try to fetch the PARTUUID.

`ironic_python_agent.disk_utils.count_mbr_partitions(device)`

Count the number of primary and logical partitions on a MBR

#### Parameters

**device** The device path.

#### Returns

A tuple with the number of primary partitions and logical partitions.

#### Raise

`ValueError` if the device does not have a valid MBR partition table.

`ironic_python_agent.disk_utils.dd(src, dst, conv_flags=None)`

Execute `dd` from `src` to `dst`.

`ironic_python_agent.disk_utils.destroy_disk_metadata(dev, node_uuid)`

Destroy metadata structures on nodes disk.

Ensure that nodes disk magic strings are wiped without zeroing the entire drive. To do this we use the `wipefs` tool from `util-linux`.

#### Parameters

- **dev** Path for the device to work on.
- **node\_uuid** Nodes uuid. Used for logging.

`ironic_python_agent.disk_utils.find_efi_partition(device)`

Looks for the EFI partition on a given device.

A boot partition on a GPT disk is assumed to be an EFI partition as well.

#### Parameters

**device** the name of the device

#### Returns

the EFI partition record from `list_partitions` or `None`

`ironic_python_agent.disk_utils.fix_gpt_partition(device, node_uuid)`

Fix GPT partition

Fix GPT table information when image is written to a disk which has a bigger extend (e.g. 30GB image written on a 60Gb physical disk).

#### Parameters

- **device** The device path.
- **node\_uuid** UUID of the Node.

**Raises**

InstanceDeployFailure if exception is caught.

`ironic_python_agent.disk_utils.get_and_validate_image_format(filename, ironic_disk_format)`

Get the format of a given image file and ensure its allowed.

This method uses the format inspector originally written for glance to safely detect the image format. It also sanity checks to ensure any specified format matches the provided one and that the format is in the allowed list of formats.

If the image format provided by Ironic is a type which doesnt need conversion, we avoid all introspection of the image and use of qemu-img.

It also performs a basic safety check on the image.

This entire process can be bypassed, and the older code path used, by setting `CONF.disable_deep_image_inspection` to `True`.

See <https://bugs.launchpad.net/ironic/+bug/2071740> for full details on why this must always happen.

**Parameters**

- **filename** The name of the image file to validate.
- **ironic\_disk\_format** The ironic-provided expected format of the image

**Returns**

tuple of validated `img_format` (str) and `size` (int)

`ironic_python_agent.disk_utils.get_dev_byte_size(dev)`

Get the device size in bytes.

`ironic_python_agent.disk_utils.get_dev_sector_size(dev)`

Get the device logical sector size in bytes.

`ironic_python_agent.disk_utils.get_device_information(device, fields=None)`

Get information about a device using blkid.

Can be applied to all block devices: disks, RAID, partitions.

**Parameters**

- **device** Device name.
- **fields** A list of fields to request (all by default).

**Returns**

A dictionary with requested fields as keys.

**Raises**

ProcessExecutionError

`ironic_python_agent.disk_utils.get_disk_identifier(dev)`

Get the disk identifier from the disk being exposed by the ramdisk.

This disk identifier is appended to the pxe config which will then be used by chain.c32 to detect the correct disk to chainload. This is helpful in deployments to nodes with multiple disks.

<http://www.syslinux.org/wiki/index.php/Comboot/chain.c32#mbr>:

**Parameters**

**dev** Path for the already populated disk device.

**Raises**

**OSError** When the hexdump binary is unavailable.

**Returns**

The Disk Identifier.

`ironic_python_agent.disk_utils.get_partition_table_type(device)`

Get partition table type, msdos or gpt.

**Parameters**

**device** the name of the device

**Returns**

dos, gpt or None

`ironic_python_agent.disk_utils.is_block_device(dev)`

Check whether a device is block or not.

`ironic_python_agent.disk_utils.is_last_char_digit(dev)`

check whether device name ends with a digit

`ironic_python_agent.disk_utils.list_partitions(device)`

Get partitions information from given device.

**Parameters**

**device** The device path.

**Returns**

list of dictionaries (one per partition) with keys: number, start, end, size (in MiB), filesystem, partition\_name, flags, path.

`ironic_python_agent.disk_utils.make_partitions(dev, root_mb, swap_mb, ephemeral_mb, configdrive_mb, node_uuid, commit=True, boot_option='netboot', boot_mode='bios', disk_label=None, cpu_arch="")`

Partition the disk device.

Create partitions for root, swap, ephemeral and configdrive on a disk device.

**Parameters**

- **dev** Path for the device to work on.
- **root\_mb** Size of the root partition in mebibytes (MiB).
- **swap\_mb** Size of the swap partition in mebibytes (MiB). If 0, no partition will be created.
- **ephemeral\_mb** Size of the ephemeral partition in mebibytes (MiB). If 0, no partition will be created.
- **configdrive\_mb** Size of the configdrive partition in mebibytes (MiB). If 0, no partition will be created.
- **commit** True/False. Default for this setting is True. If False partitions will not be written to disk.

- **boot\_option** Can be local or netboot. netboot by default.
- **boot\_mode** Can be bios or uefi. bios by default.
- **node\_uuid** Nodes uuid. Used for logging.
- **disk\_label** The disk label to be used when creating the partition table. Valid values are: msdos, gpt or None; If None Ironic will figure it out according to the boot\_mode parameter.
- **cpu\_arch** Architecture of the node the disk device belongs to. When using the default value of None, no architecture specific steps will be taken. This default should be used for x86\_64. When set to ppc64\*, architecture specific steps are taken for booting a partition image locally.

**Returns**

A dictionary containing the partition type as Key and partition path as Value for the partitions created by this method.

`ironic_python_agent.disk_utils.partition_index_to_path(device, index)`

Guess a partition path based on its device and index.

**Parameters**

- **device** Device path.
- **index** Partition index.

`ironic_python_agent.disk_utils.partprobe(device, attempts=None)`

Probe partitions on the given device.

**Parameters**

- **device** The block device containing partitions that is attempting to be updated.
- **attempts** Number of attempts to run partprobe, the default is read from the configuration.

**Returns**

True on success, False otherwise.

`ironic_python_agent.disk_utils.populate_image(src, dst, conv_flags=None, source_format=None, is_raw=False, sparse_size='0', out_format='raw', **convert_args)`

Populate a provided destination device with the image

**Parameters**

- **src** An image already security checked in format disk\_format
- **dst** A location, usually a partition or block device, to write the image
- **conv\_flags** Conversion flags to pass to dd if provided
- **is\_raw** Ironic indicates image is raw; do not convert!
- **sparse\_size** Sparse size to pass to qemu\_img
- **source\_format** format of the image

- **out\_format** Output format
- **convert\_args** Additional arguments to optionally pass to `qemu_img`

`ironic_python_agent.disk_utils.trigger_device_rescan(device, attempts=None)`

Sync and trigger device rescan.

Disk partition performed via parted, when performed on a ramdisk do not have to honor the fsync mechanism. In essence, fsync is used on the file representing the block device, which falls to the kernel filesystem layer to trigger a sync event. On a ramdisk using ramfs, this is an explicit non-operation.

As a result of this, we need to trigger a system wide sync operation which will trigger cache to flush to disk, after which partition changes should be visible upon re-scan.

When ramdisks are not in use, this also helps ensure that data has been safely flushed across the wire, such as on iscsi connections.

### **Parameters**

- **device** The block device containing partitions that is attempting to be updated.
- **attempts** Number of attempts to run partprobe, the default is read from the configuration.

### **Returns**

True on success, False otherwise.

`ironic_python_agent.disk_utils.udev_settle()`

Wait for the udev event queue to settle.

Wait for the udev event queue to settle to make sure all devices are detected once the machine boots up.

### **Returns**

True on success, False otherwise.

`ironic_python_agent.disk_utils.wait_for_disk_to_become_available(device)`

Wait for a disk device to become available.

Waits for a disk device to become available for use by waiting until all process locks on the device have been released.

Timeout and iteration settings come from the configuration options used by the in-library `disk_partitioner`: `check_device_interval` and `check_device_max_retries`.

### **Params device**

The path to the device.

### **Raises**

`IronicException` If the disk fails to become available.

## **ironic\_python\_agent.dmi\_inspector module**

`ironic_python_agent.dmi_inspector.collect_dmidecode_info(data, failures)`

Collect detailed processor, memory and bios info.

The data is gathered using dmidecode utility.



### Parameters

- **data** mutable dict that will send to inspector
- **failures** AccumulatedFailures object

`ironic_python_agent.dmi_inspector.parse_dmi(data)`

Parse the dmidecode output.

Returns a dict.

### `ironic_python_agent.efi_utils` module

`ironic_python_agent.efi_utils.add_boot_record(device, efi_partition, loader, label)`

Add an EFI boot record with efibootmgr.

### Parameters

- **device** the device to be used
- **efi\_partition** the number of the EFI partition on the device
- **loader** path to the EFI boot loader
- **label** the record label

`ironic_python_agent.efi_utils.clean_boot_records(patterns)`

Remove EFI boot records matching regex patterns.

### Parameters

**match\_patterns** A list of string regular expression patterns where any matching entry will be deleted.

`ironic_python_agent.efi_utils.get_boot_records()`

Executes efibootmgr and returns boot records.

### Returns

An iterator yielding tuples (boot number, boot record, root device type, device path).

`ironic_python_agent.efi_utils.get_partition_path_by_number(device, part_num)`

Get partition path (/dev/something) by a partition number on device.

Only works for GPT partition table.

`ironic_python_agent.efi_utils.manage_uefi(device, efi_system_part_uuid=None)`

Manage the device looking for valid efi bootloaders to update the nvram.

This method checks for valid efi bootloaders in the device, if they exist it updates the nvram using the efibootmgr.

### Parameters

- **device** the device to be checked.
- **efi\_system\_part\_uuid** efi partition uuid.

### Raises

DeviceNotFound if the efi partition cannot be found.

**Returns**

True - if it finds any efi bootloader and the nvram was updated using the efibootmgr. False - if no efi bootloader is found.

`ironic_python_agent.efi_utils.remove_boot_record(boot_num)`

Remove an EFI boot record with efibootmgr.

**Parameters**

**boot\_num** the number of the boot record

**ironic\_python\_agent.encoding module**

```
class ironic_python_agent.encoding.RESTJSONEncoder(*, skipkeys=False,
                                                    ensure_ascii=True,
                                                    check_circular=True,
                                                    allow_nan=True, sort_keys=False,
                                                    indent=None, separators=None,
                                                    default=None)
```

Bases: JSONEncoder

A slightly customized JSON encoder.

**default(*o*)**

Turn an object into a serializable object.

In particular, by calling *Serializable.serialize()* on *o*.

**encode(*o*)**

Turn an object into JSON.

Appends a newline to responses when configured to pretty-print, in order to make use of curl less painful from most shells.

```
class ironic_python_agent.encoding.Serializable
```

Bases: object

Base class for things that can be serialized.

```
serializable_fields = ()
```

**serialize()**

Turn this object into a dict.

```
class ironic_python_agent.encoding.SerializableComparable
```

Bases: *Serializable*

A Serializable class which supports some comparison operators

This class supports the `__eq__` and `__ne__` comparison operators, but intentionally disables the `__hash__` operator as some child classes may be mutable. The addition of these comparison operators is mainly used to assist with unit testing.

```
ironic_python_agent.encoding.serialize_lib_exc(exc)
```

Serialize an ironic-lib exception.

**ironic\_python\_agent.errors module****exception** `ironic_python_agent.errors.AgentIsBusy(command_name)`Bases: `CommandExecutionError`**message** = 'Agent is busy'**status\_code** = 409**exception** `ironic_python_agent.errors.BlockDeviceEraseError(details)`Bases: `RESTError`

Error raised when an error occurs erasing a block device.

**message** = 'Error erasing block device'**exception** `ironic_python_agent.errors.BlockDeviceError(details)`Bases: `RESTError`

Error raised when a block devices causes an unknown error.

**message** = 'Block device caused unknown error'**exception** `ironic_python_agent.errors.CleaningError(details=None)`Bases: `RESTError`

Error raised when a cleaning step fails.

**message** = 'Clean step failed'**exception** `ironic_python_agent.errors.ClockSyncError(details=None, *args, **kwargs)`Bases: `RESTError`

Error raised when attempting to sync the system clock.

**message** = 'Error syncing system clock'**exception** `ironic_python_agent.errors.CommandExecutionError(details)`Bases: `RESTError`

Error raised when a command fails to execute.

**message** = 'Command execution failed'**exception** `ironic_python_agent.errors.DeploymentError(details=None)`Bases: `RESTError`

Error raised when a deploy step fails.

**message** = 'Deploy step failed'**exception** `ironic_python_agent.errors.DeviceNotFound(details)`Bases: `NotFound`

Error raised when the device to deploy the image onto is not found.

**message** = 'Error finding the disk or partition device to deploy the image onto'

**exception** `ironic_python_agent.errors.ExtensionError`(*details=None, \*args, \*\*kwargs*)

Bases: [RESTError](#)

**exception** `ironic_python_agent.errors.HardwareManagerMethodNotFound`(*method*)

Bases: [RESTError](#)

Error raised when all HardwareManagers fail to handle a method.

**message** = 'No HardwareManager found to handle method'

**exception** `ironic_python_agent.errors.HardwareManagerNotFound`(*details=None*)

Bases: [RESTError](#)

Error raised when no valid HardwareManager can be found.

**message** = 'No valid HardwareManager found'

**exception** `ironic_python_agent.errors.HeartbeatConflictError`(*details*)

Bases: [IronicAPIError](#)

ConflictError raised when a heartbeat to the agent API fails.

**message** = 'ConflictError heartbeating to agent API'

**exception** `ironic_python_agent.errors.HeartbeatConnectionError`(*details*)

Bases: [IronicAPIError](#)

Transitory connection failure occurred attempting to contact the API.

**message** = 'Error attempting to heartbeat - Possible transitory network failure or blocking port may be present.'

**exception** `ironic_python_agent.errors.HeartbeatError`(*details*)

Bases: [IronicAPIError](#)

Error raised when a heartbeat to the agent API fails.

**message** = 'Error heartbeating to agent API'

**exception** `ironic_python_agent.errors.ImageChecksumError`(*image\_id, image\_location, checksum, calculated\_checksum*)

Bases: [RESTError](#)

Error raised when an image fails to verify against its checksum.

**details\_str** = 'Image failed to verify against checksum. location: {}; image ID: {}; image checksum: {}; verification checksum: {}'

**message** = 'Error verifying image checksum'

**exception** `ironic_python_agent.errors.ImageDownloadError`(*image\_id, msg*)

Bases: [RESTError](#)

Error raised when an image cannot be downloaded.

**message** = 'Error downloading image'

**exception** `ironic_python_agent.errors.ImageWriteError(device, exit_code, stdout, stderr)`

Bases: [RESTError](#)

Error raised when an image cannot be written to a device.

**message** = 'Error writing image to device'

**exception** `ironic_python_agent.errors.IncompatibleHardwareMethodError(details=None)`

Bases: [RESTError](#)

Error raised when HardwareManager method incompatible with hardware.

**message** = 'HardwareManager method is not compatible with hardware'

**exception** `ironic_python_agent.errors.IncompatibleNumaFormatError(details=None, *args, **kwargs)`

Bases: [RESTError](#)

Error raised when unexpected format data in NUMA node.

**message** = 'Error in NUMA node data format'

**exception** `ironic_python_agent.errors.InspectionError`

Bases: `Exception`

Failure during inspection.

**exception** `ironic_python_agent.errors.InvalidCommandError(details)`

Bases: [InvalidContentError](#)

Error which is raised when an unknown command is issued.

**message** = 'Invalid command'

**exception** `ironic_python_agent.errors.InvalidCommandParamsError(details)`

Bases: [InvalidContentError](#)

Error which is raised when command parameters are invalid.

**message** = 'Invalid command parameters'

**exception** `ironic_python_agent.errors.InvalidContentError(details)`

Bases: [RESTError](#)

Error which occurs when a user supplies invalid content.

Either because that content cannot be parsed according to the advertised *Content-Type*, or due to a content validation error.

**message** = 'Invalid request body'

**status\_code** = 400

**exception** `ironic_python_agent.errors.InvalidImage(details=None)`

Bases: [DeploymentError](#)

Error raised when an image fails validation for any reason.

**message** = 'The provided image is not valid for use'

**exception** `ironic_python_agent.errors.IronicAPIError`(*details*)

Bases: `RESTError`

Error raised when a call to the agent API fails.

**message** = 'Error in call to ironic-api'

**exception** `ironic_python_agent.errors.LookupAgentIPError`(*details*)

Bases: `IronicAPIError`

Error raised when automatic IP lookup fails.

**message** = 'Error finding IP for Ironic Agent'

**exception** `ironic_python_agent.errors.LookupNodeError`(*details*)

Bases: `IronicAPIError`

Error raised when the node lookup to the Ironic API fails.

**message** = 'Error getting configuration from Ironic'

**exception** `ironic_python_agent.errors.NotFound`(*details=None, \*args, \*\*kwargs*)

Bases: `RESTError`

Error which occurs if a non-existent API endpoint is called.

**details** = 'The requested URL was not found.'

**message** = 'Not found'

**status\_code** = 404

**exception** `ironic_python_agent.errors.ProtectedDeviceError`(*device, what*)

Bases: `CleaningError`

Error raised when a cleaning is halted due to a protected device.

**message** = 'Protected device located, cleaning aborted.'

**exception** `ironic_python_agent.errors.RESTError`(*details=None, \*args, \*\*kwargs*)

Bases: `Exception, Serializable`

Base class for errors generated in ironic-python-client.

**details** = 'An unexpected error occurred. Please try back later.'

**message** = 'An error occurred'

**serializable\_fields** = ('type', 'code', 'message', 'details')

**status\_code** = 500

**exception** `ironic_python_agent.errors.RequestedObjectNotFoundError`(*type\_descr, obj\_id*)

Bases: `NotFound`

**exception** `ironic_python_agent.errors.ServicingError`(*details=None*)

Bases: `RESTError`

Error raised when a service step fails.

```
message = 'Service step failed'
```

```
exception ironic_python_agent.errors.SoftwareRAIDError(details)
```

Bases: [RESTError](#)

Error raised when a Software RAID causes an error.

```
message = 'Software RAID caused unknown error'
```

```
exception ironic_python_agent.errors.SystemRebootError(exit_code, stdout, stderr)
```

Bases: [RESTError](#)

Error raised when a system cannot reboot.

```
message = 'Error rebooting system'
```

```
exception ironic_python_agent.errors.UnknownNodeError(details=None)
```

Bases: [RESTError](#)

Error raised when the agent is not associated with an Ironic node.

```
message = 'Agent is not associated with an Ironic node'
```

```
exception ironic_python_agent.errors.VersionMismatch(agent_version, node_version)
```

Bases: [RESTError](#)

Error raised when Ironic and the Agent have different versions.

If the agent version has changed since `get_clean_steps` or `get_deploy_steps` was called by the Ironic conductor, it indicates the agent has been updated (either on purpose, or a new agent was deployed and the node was rebooted). Since we cannot know if the upgraded IPA will work with cleaning/deploy as it stands (steps could have different priorities, either in IPA or in other Ironic interfaces), we should restart the process from the start.

```
message = 'Hardware managers version mismatch, reload agent with correct version'
```

```
exception ironic_python_agent.errors.VirtualMediaBootError(details)
```

Bases: [RESTError](#)

Error raised when virtual media device cannot be found for config.

```
message = 'Configuring agent from virtual media failed'
```

## [ironic\\_python\\_agent.hardware module](#)

```
class ironic_python_agent.hardware.BlockDevice(name, model, size, rotational,
                                               wwn=None, serial=None, vendor=None,
                                               wwn_with_extension=None,
                                               wwn_vendor_extension=None,
                                               hctl=None, by_path=None, uuid=None,
                                               partuuid=None, logical_sectors=None,
                                               physical_sectors=None)
```

Bases: [SerializableComparable](#)

```
serializable_fields = ('name', 'model', 'size', 'rotational', 'wnn',
                        'serial', 'vendor', 'wnn_with_extension', 'wnn_vendor_extension', 'hctl',
                        'by_path', 'logical_sectors', 'physical_sectors')
```

```
class ironic_python_agent.hardware.BootInfo(current_boot_mode, pxe_interface=None)
```

Bases: *SerializableComparable*

```
serializable_fields = ('current_boot_mode', 'pxe_interface')
```

```
class ironic_python_agent.hardware.CPU(model_name, frequency, count, architecture,
                                       flags=None, socket_count=None, cpus:
                                       List[CPUCore] | None = None)
```

Bases: *SerializableComparable*

```
serializable_fields = ('model_name', 'frequency', 'count', 'architecture',
                        'flags', 'socket_count')
```

```
class ironic_python_agent.hardware.CPUCore(model_name, frequency, architecture, core_id,
                                             flags=None)
```

Bases: *SerializableComparable*

```
serializable_fields = ('model_name', 'frequency', 'count', 'architecture',
                        'flags', 'core_id')
```

```
class ironic_python_agent.hardware.GenericHardwareManager
```

Bases: *HardwareManager*

```
HARDWARE_MANAGER_NAME = 'generic_hardware_manager'
```

```
HARDWARE_MANAGER_VERSION = '1.2'
```

```
any_ipmi_device_exists()
```

Check for an IPMI device to confirm IPMI capability.

```
apply_configuration(node, ports, raid_config, delete_existing=True)
```

Apply RAID configuration.

#### Parameters

- **node** A dictionary of the node object.
- **ports** A list of dictionaries containing information of ports for the node.
- **raid\_config** The configuration to apply.
- **delete\_existing** Whether to delete the existing configuration.

```
burnin_cpu(node, ports)
```

Burn-in the CPU

#### Parameters

- **node** Ironic node object
- **ports** list of Ironic port objects



**burnin\_disk**(*node, ports*)

Burn-in the disk

**Parameters**

- **node** Ironic node object
- **ports** list of Ironic port objects

**burnin\_memory**(*node, ports*)

Burn-in the memory

**Parameters**

- **node** Ironic node object
- **ports** list of Ironic port objects

**burnin\_network**(*node, ports*)

Burn-in the network

**Parameters**

- **node** Ironic node object
- **ports** list of Ironic port objects

**clean\_uefi\_nvram**(*node, ports, match\_patterns=None*)

Clean UEFI NVRAM entries.

**Parameters**

- **node** A dictionary of the node object.
- **ports** A list of dictionaries containing information of ports for the node.
- **match\_patterns** A list of string regular expression patterns where any matching entry will be deleted.

**collect\_lldp\_data**(*interface\_names=None*)

Collect and convert LLDP info from the node.

In order to process the LLDP information later, the raw data needs to be converted for serialization purposes.

**Parameters**

**interface\_names** list of names of nodes interfaces.

**Returns**

a dict, containing the lldp data from every interface.

**collect\_system\_logs**(*io\_dict, file\_list*)

Collect logs from the system.

Implementations should update *io\_dict* and *file\_list* with logs to send to Ironic and Inspector.

**Parameters**

- **io\_dict** Dictionary mapping file names to binary IO objects with corresponding data.
- **file\_list** List of full file paths to include.

### **create\_configuration**(*node, ports*)

Create a RAID configuration.

Unless overwritten by a local hardware manager, this method will create a software RAID configuration as read from the nodes `target_raid_config`.

#### **Parameters**

- **node** A dictionary of the node object.
- **ports** A list of dictionaries containing information of ports for the node.

#### **Returns**

The current RAID configuration in the usual format.

#### **Raises**

SoftwareRAIDError if the desired configuration is not valid or if there was an error when creating the RAID devices.

### **static create\_cpu\_info\_dict**(*lines*)

### **delete\_configuration**(*node, ports*)

Delete a RAID configuration.

Unless overwritten by a local hardware manager, this method will delete all software RAID devices on the node. NOTE(arne\_wiebalck): It may be worth considering to only delete RAID devices in the nodes `target_raid_config`. If that config has been lost, though, the cleanup may become difficult. So, for now, we delete everything we detect.

#### **Parameters**

- **node** A dictionary of the node object
- **ports** A list of dictionaries containing information of ports for the node

### **erase\_block\_device**(*node, block\_device*)

Attempt to erase a block device.

Implementations should detect the type of device and erase it in the most appropriate way possible. Generic implementations should support common erase mechanisms such as ATA secure erase, or multi-pass random writes. Operators with more specific needs should override this method in order to detect and handle interesting cases, or delegate to the parent class to handle generic cases.

For example: operators running ACME MagicStore (TM) cards alongside standard SSDs might check whether the device is a MagicStore and use a proprietary tool to erase that, otherwise call this method on their parent class. Upstream submissions of common functionality are encouraged.

This interface could be called concurrently to speed up erasure, as such, it should be implemented in a thread-safe way.

#### **Parameters**

- **node** Ironic node object
- **block\_device** a BlockDevice indicating a device to be erased.

#### **Raises**

- *IncompatibleHardwareMethodError* when there is no known way to erase the block device
- *BlockDeviceEraseError* when there is an error erasing the block device

**erase\_devices\_express**(*node, ports*)

Attempt to perform time-optimised disk erasure:

for NVMe devices, perform NVMe Secure Erase if supported. For other devices, perform metadata erasure

**Parameters**

- **node** Ironic node object
- **ports** list of Ironic port objects

**Raises**

*BlockDeviceEraseError* when theres an error erasing the block device

**Raises**

ProtectedDeviceError if a device has been identified which may require manual intervention due to the contents and operational risk which exists as it could also be a sign of an environmental misconfiguration.

**erase\_devices\_metadata**(*node, ports*)

Attempt to erase the disk devices metadata.

**Parameters**

- **node** Ironic node object
- **ports** list of Ironic port objects

**Raises**

*BlockDeviceEraseError* when theres an error erasing the block device

**Raises**

ProtectedDeviceError if a device has been identified which may require manual intervention due to the contents and operational risk which exists as it could also be a sign of an environmental misconfiguration.

**erase\_pstore**(*node, ports*)

Attempt to erase the kernel pstore.

**Parameters**

- **node** Ironic node object
- **ports** list of Ironic port objects

**evaluate\_hardware\_support**()

**full\_sync**()

Synchronize all caches to the disk.

This method will be called on *all* managers before the ramdisk is powered off externally. It is expected to try flush all caches to the disk to avoid data loss.

**generate\_tls\_certificate**(*ip\_address*)

Generate a TLS certificate for the IP address.

### `get_bios_given_nic_name(interface_name)`

Collect the BIOS given NICs name.

This function uses the biosdevname utility to collect the BIOS given name of network interfaces.

The collected data is added to the network interface inventory with an extra field named biosdevname.

#### Parameters

**interface\_name** list of names of nodes interfaces.

#### Returns

the BIOS given NIC name of nodes interfaces or default as None.

### `get_bmc_address()`

Attempt to detect BMC IP address

#### Returns

IP address of lan channel or 0.0.0.0 in case none of them is configured properly

### `get_bmc_mac()`

Attempt to detect BMC MAC address

#### Returns

MAC address of the first LAN channel or 00:00:00:00:00:00 in case none of them has one or is configured properly

#### Raises

IncompatibleHardwareMethodError if no valid mac is found.

### `get_bmc_v6address()`

Attempt to detect BMC v6 address

#### Returns

IPv6 address of lan channel or ::/0 in case none of them is configured properly. May return None value if it cannot interact with system tools or critical error occurs.

### `get_boot_info()`

### `get_clean_steps(node, ports)`

Get a list of clean steps with priority.

Returns a list of steps. Each step is represented by a dict:

```
{
  'interface': the name of the driver interface that should execute
               the step.
  'step': the HardwareManager function to call.
  'priority': the order steps will be run in. Ironic will sort all
               the clean steps from all the drivers, with the largest
               priority step being run first. If priority is set to 0,
               the step will not be run during cleaning, but may be
               run during zapping.
  'reboot_requested': Whether the agent should request Ironic reboots
```

(continues on next page)

(continued from previous page)

```

        the node via the power driver after the
        operation completes.
    'abortable': Boolean value. Whether the clean step can be
        stopped by the operator or not. Some clean step may
        cause non-reversible damage to a machine if interrupted
        (i.e firmware update), for such steps this parameter
        should be set to False. If no value is set for this
        parameter, Ironic will consider False (non-abortable).
}

```

If multiple hardware managers return the same step name, the following logic will be used to determine which managers step wins:

- Keep the step that belongs to HardwareManager with highest HardwareSupport (larger int) value.
- If equal support level, keep the step with the higher defined priority (larger int).
- If equal support level and priority, keep the step associated with the HardwareManager whose name comes earlier in the alphabet.

The steps will be called using *hardware.dispatch\_to\_managers* and handled by the best suited hardware manager. If you need a step to be executed by only your hardware manager, ensure it has a unique step name.

*node* and *ports* can be used by other hardware managers to further determine if a clean step is supported for the node.

#### Parameters

- **node** Ironic node object
- **ports** list of Ironic port objects

#### Returns

a list of cleaning steps, where each step is described as a dict as defined above

**get\_cpu\_cores()**

**get\_cpus()**

**get\_deploy\_steps(*node*, *ports*)**

Get a list of deploy steps with priority.

Returns a list of steps. Each step is represented by a dict:

```

{
    'interface': the name of the driver interface that should execute
        the step.
    'step': the HardwareManager function to call.
    'priority': the order steps will be run in. Ironic will sort all
        the deploy steps from all the drivers, with the largest
        priority step being run first. If priority is set to 0,
        the step will not be run during deployment
        automatically, but may be requested via deploy
}

```

(continues on next page)

(continued from previous page)

```

        templates.
    'reboot_requested': Whether the agent should request Ironic reboots
                        the node via the power driver after the
                        operation completes.
    'argsinfo': arguments specification.
}

```

If multiple hardware managers return the same step name, the following logic will be used to determine which managers step wins:

- Keep the step that belongs to HardwareManager with highest HardwareSupport (larger int) value.
- If equal support level, keep the step with the higher defined priority (larger int).
- If equal support level and priority, keep the step associated with the HardwareManager whose name comes earlier in the alphabet.

The steps will be called using *hardware.dispatch\_to\_managers* and handled by the best suited hardware manager. If you need a step to be executed by only your hardware manager, ensure it has a unique step name.

*node* and *ports* can be used by other hardware managers to further determine if a deploy step is supported for the node.

#### Parameters

- **node** Ironic node object
- **ports** list of Ironic port objects

#### Returns

a list of deploying steps, where each step is described as a dict as defined above

**get\_interface\_info**(*interface\_name*)

**get\_ipv4\_addr**(*interface\_id*)

**get\_ipv6\_addr**(*interface\_id*)

Get the default IPv6 address assigned to the interface.

With different networking environment, the address could be a link-local address, ULA or something else.

**get\_memory**()

**get\_os\_install\_device**(*permit\_refresh=False*)

**get\_service\_steps**(*node, ports*)

Get a list of service steps.

Returns a list of steps. Each step is represented by a dict:

```

{
    'interface': the name of the driver interface that should execute
                the step.
    'step': the HardwareManager function to call.
}

```

(continues on next page)

(continued from previous page)

```

'priority': the order steps will be run in if executed upon
similar to automated cleaning or deployment.
In service steps, the order comes from the user request,
but this similarity is kept for consistency should we
further extend the capability at some point in the
future.
'reboot_requested': Whether the agent should request Ironic reboots
the node via the power driver after the
operation completes.
'abortable': Boolean value. Whether the service step can be
stopped by the operator or not. Some steps may
cause non-reversible damage to a machine if interrupted
(i.e firmware update), for such steps this parameter
should be set to False. If no value is set for this
parameter, Ironic will consider False (non-abortable).
}

```

If multiple hardware managers return the same step name, the following logic will be used to determine which managers step wins:

- Keep the step that belongs to HardwareManager with highest HardwareSupport (larger int) value.
- If equal support level, keep the step with the higher defined priority (larger int).
- If equal support level and priority, keep the step associated with the HardwareManager whose name comes earlier in the alphabet.

The steps will be called using `hardware.dispatch_to_managers` and handled by the best suited hardware manager. If you need a step to be executed by only your hardware manager, ensure it has a unique step name.

`node` and `ports` can be used by other hardware managers to further determine if a step is supported for the node.

#### Parameters

- **node** Ironic node object
- **ports** list of Ironic port objects

#### Returns

a list of service steps, where each step is described as a dict as defined above

`get_skip_list_from_node(node, block_devices=None, just_raids=False)`

Get the skip block devices list from the node

#### Parameters

- **block\_devices** a list of BlockDevices
- **just\_raids** a boolean to signify that only RAID devices are important

#### Returns

A set of names of devices on the skip list

**get\_system\_vendor\_info()**

**get\_usb\_devices()**

Collect USB devices

List all USB final devices, based on lshw information

**Returns**

a dict, containing product, vendor, and handle information

**inject\_files**(*node, ports, files=None, verify\_ca=True*)

A deploy step to inject arbitrary files.

**Parameters**

- **node** A dictionary of the node object
- **ports** A list of dictionaries containing information of ports for the node (unused)
- **files** See `inject_files`
- **verify\_ca** Whether to verify TLS certificate.

**list\_block\_devices**(*include\_partitions=False, all\_serial\_and\_wwn=False*)

List physical block devices

**Parameters**

**include\_partitions** If to include partitions

**Returns**

A list of BlockDevices

**list\_block\_devices\_check\_skip\_list**(*node, include\_partitions=False, all\_serial\_and\_wwn=False*)

List physical block devices without the ones listed in

`properties/skip_block_devices` list

**Parameters**

- **node** A node used to check the skip list
- **include\_partitions** If to include partitions

**Returns**

A list of BlockDevices

**list\_hardware\_info()**

Return full hardware inventory as a serializable dict.

This inventory is sent to Ironic on lookup and to Inspector on inspection.

**Returns**

a dictionary representing inventory

**list\_network\_interfaces()**

**read\_cpu\_info()**



**validate\_configuration**(*raid\_config*, *node*)

Validate a (software) RAID configuration

Validate a given *raid\_config*, in particular with respect to the limitations of the current implementation of software RAID support.

**Parameters**

**raid\_config** The current RAID configuration in the usual format.

**write\_image**(*node*, *ports*, *image\_info*, *configdrive=None*)

A deploy step to write an image.

Downloads and writes an image to disk if necessary. Also writes a *configdrive* to disk if the *configdrive* parameter is specified.

**Parameters**

- **node** A dictionary of the node object
- **ports** A list of dictionaries containing information of ports for the node
- **image\_info** Image information dictionary.
- **configdrive** A string containing the location of the config drive as a URL OR the contents (as *gzip/base64*) of the configdrive. Optional, defaults to *None*.

**class** `ironic_python_agent.hardware.HardwareManager`

Bases: `object`

**collect\_lldp\_data**(*interface\_names=None*)

**collect\_system\_logs**(*io\_dict*, *file\_list*)

Collect logs from the system.

Implementations should update *io\_dict* and *file\_list* with logs to send to Ironic and Inspector.

**Parameters**

- **io\_dict** Dictionary mapping file names to binary IO objects with corresponding data.
- **file\_list** List of full file paths to include.

**erase\_block\_device**(*node*, *block\_device*)

Attempt to erase a block device.

Implementations should detect the type of device and erase it in the most appropriate way possible. Generic implementations should support common erase mechanisms such as ATA secure erase, or multi-pass random writes. Operators with more specific needs should override this method in order to detect and handle interesting cases, or delegate to the parent class to handle generic cases.

For example: operators running ACME MagicStore (TM) cards alongside standard SSDs might check whether the device is a MagicStore and use a proprietary tool to erase that, otherwise call this method on their parent class. Upstream submissions of common functionality are encouraged.

This interface could be called concurrently to speed up erasure, as such, it should be implemented in a thread-safe way.

### Parameters

- **node** Ironic node object
- **block\_device** a BlockDevice indicating a device to be erased.

### Raises

- *IncompatibleHardwareMethodError* when there is no known way to erase the block device
- *BlockDeviceEraseError* when there is an error erasing the block device

### `erase_devices(node, ports)`

Erase any device that holds user data.

By default this will attempt to erase block devices. This method can be overridden in an implementation-specific hardware manager in order to erase additional hardware, although backwards-compatible upstream submissions are encouraged.

### Parameters

- **node** Ironic node object
- **ports** list of Ironic port objects

### Raises

ProtectedDeviceError if a device has been identified which may require manual intervention due to the contents and operational risk which exists as it could also be a sign of an environmental misconfiguration.

### Returns

a dictionary in the form {device.name: erasure output}

### `abstract evaluate_hardware_support()`

### `full_sync()`

Synchronize all caches to the disk.

This method will be called on *all* managers before the ramdisk is powered off externally. It is expected to try flush all caches to the disk to avoid data loss.

### `generate_tls_certificate(ip_address)`

### `get_bmc_address()`

### `get_bmc_mac()`

### `get_bmc_v6address()`

### `get_boot_info()`

### `get_clean_steps(node, ports)`

Get a list of clean steps with priority.

Returns a list of steps. Each step is represented by a dict:

```
{
  'interface': the name of the driver interface that should execute
               the step.
```

(continues on next page)

(continued from previous page)

```

'step': the HardwareManager function to call.
'priority': the order steps will be run in. Ironic will sort all
the clean steps from all the drivers, with the largest
priority step being run first. If priority is set to 0,
the step will not be run during cleaning, but may be
run during zapping.
'reboot_requested': Whether the agent should request Ironic reboots
the node via the power driver after the
operation completes.
'abortable': Boolean value. Whether the clean step can be
stopped by the operator or not. Some clean step may
cause non-reversible damage to a machine if interrupted
(i.e firmware update), for such steps this parameter
should be set to False. If no value is set for this
parameter, Ironic will consider False (non-abortable).
}

```

If multiple hardware managers return the same step name, the following logic will be used to determine which managers step wins:

- Keep the step that belongs to HardwareManager with highest HardwareSupport (larger int) value.
- If equal support level, keep the step with the higher defined priority (larger int).
- If equal support level and priority, keep the step associated with the HardwareManager whose name comes earlier in the alphabet.

The steps will be called using *hardware.dispatch\_to\_managers* and handled by the best suited hardware manager. If you need a step to be executed by only your hardware manager, ensure it has a unique step name.

*node* and *ports* can be used by other hardware managers to further determine if a clean step is supported for the node.

#### Parameters

- **node** Ironic node object
- **ports** list of Ironic port objects

#### Returns

a list of cleaning steps, where each step is described as a dict as defined above

**get\_cpus()**

**get\_deploy\_steps(*node*, *ports*)**

Get a list of deploy steps with priority.

Returns a list of steps. Each step is represented by a dict:

```

{
  'interface': the name of the driver interface that should execute
the step.
  'step': the HardwareManager function to call.
}

```

(continues on next page)

(continued from previous page)

```

'priority': the order steps will be run in. Ironic will sort all
the deploy steps from all the drivers, with the largest
priority step being run first. If priority is set to 0,
the step will not be run during deployment
automatically, but may be requested via deploy
templates.
'reboot_requested': Whether the agent should request Ironic reboots
the node via the power driver after the
operation completes.
'argsinfo': arguments specification.
}

```

If multiple hardware managers return the same step name, the following logic will be used to determine which managers step wins:

- Keep the step that belongs to HardwareManager with highest HardwareSupport (larger int) value.
- If equal support level, keep the step with the higher defined priority (larger int).
- If equal support level and priority, keep the step associated with the HardwareManager whose name comes earlier in the alphabet.

The steps will be called using `hardware.dispatch_to_managers` and handled by the best suited hardware manager. If you need a step to be executed by only your hardware manager, ensure it has a unique step name.

`node` and `ports` can be used by other hardware managers to further determine if a deploy step is supported for the node.

#### Parameters

- **node** Ironic node object
- **ports** list of Ironic port objects

#### Returns

a list of deploying steps, where each step is described as a dict as defined above

**get\_interface\_info**(*interface\_name*)

**get\_memory**()

**get\_os\_install\_device**(*permit\_refresh=False*)

**get\_service\_steps**(*node, ports*)

Get a list of service steps.

Returns a list of steps. Each step is represented by a dict:

```

{
'interface': the name of the driver interface that should execute
the step.
'step': the HardwareManager function to call.
'priority': the order steps will be run in if executed upon
similar to automated cleaning or deployment.
}

```

(continues on next page)

(continued from previous page)

```

In service steps, the order comes from the user request,
but this similarity is kept for consistency should we
further extend the capability at some point in the
future.
'reboot_requested': Whether the agent should request Ironic reboots
the node via the power driver after the
operation completes.
'abortable': Boolean value. Whether the service step can be
stopped by the operator or not. Some steps may
cause non-reversible damage to a machine if interrupted
(i.e firmware update), for such steps this parameter
should be set to False. If no value is set for this
parameter, Ironic will consider False (non-abortable).
}

```

If multiple hardware managers return the same step name, the following logic will be used to determine which managers step wins:

- Keep the step that belongs to HardwareManager with highest HardwareSupport (larger int) value.
- If equal support level, keep the step with the higher defined priority (larger int).
- If equal support level and priority, keep the step associated with the HardwareManager whose name comes earlier in the alphabet.

The steps will be called using *hardware.dispatch\_to\_managers* and handled by the best suited hardware manager. If you need a step to be executed by only your hardware manager, ensure it has a unique step name.

*node* and *ports* can be used by other hardware managers to further determine if a step is supported for the node.

#### Parameters

- **node** Ironic node object
- **ports** list of Ironic port objects

#### Returns

a list of service steps, where each step is described as a dict as defined above

**get\_skip\_list\_from\_node**(*node*, *block\_devices=None*, *just\_raids=False*)

Get the skip block devices list from the node

#### Parameters

- **block\_devices** a list of BlockDevices
- **just\_raids** a boolean to signify that only RAID devices are important

#### Returns

A set of names of devices on the skip list

**get\_usb\_devices**()

Collect USB devices

List all USB final devices, based on lshw information

**Returns**

a dict, containing product, vendor, and handle information

**get\_version()**

Get a name and version for this hardware manager.

In order to avoid errors and make agent upgrades painless, cleaning will check the version of all hardware managers during `get_clean_steps` at the beginning of cleaning and before executing each step in the agent.

The agent isn't aware of the steps being taken before or after via out of band steps, so it can never know if a new step is safe to run. Therefore, we default to restarting the whole process.

**Returns**

a dictionary with two keys: *name* and *version*, where *name* is a string identifying the hardware manager and *version* is an arbitrary version string. *name* will be a class variable called `HARDWARE_MANAGER_NAME`, or default to the class name and *version* will be a class variable called `HARDWARE_MANAGER_VERSION` or default to 1.0.

**list\_block\_devices(include\_partitions=False)**

List physical block devices

**Parameters**

**include\_partitions** If to include partitions

**Returns**

A list of BlockDevices

**list\_block\_devices\_check\_skip\_list(node, include\_partitions=False)**

List physical block devices without the ones listed in `properties/skip_block_devices` list

**Parameters**

- **node** A node used to check the skip list
- **include\_partitions** If to include partitions

**Returns**

A list of BlockDevices

**list\_hardware\_info()**

Return full hardware inventory as a serializable dict.

This inventory is sent to Ironic on lookup and to Inspector on inspection.

**Returns**

a dictionary representing inventory

**list\_network\_interfaces()**

**wait\_for\_disks()**

Wait for the root disk to appear.

Wait for at least one suitable disk to show up or a specific disk if any device hint is specified. Otherwise neither inspection nor deployment have any chances to succeed.

**class** `ironic_python_agent.hardware.HardwareSupport`

Bases: `object`

Example priorities for hardware managers.

Priorities for `HardwareManagers` are integers, where largest means most specific and smallest means most generic. These values are guidelines that suggest values that might be returned by calls to `evaluate_hardware_support()`. No `HardwareManager` in mainline IPA will ever return a value greater than `MAINLINE`. Third party hardware managers should feel free to return values of `SERVICE_PROVIDER` or greater to distinguish between additional levels of hardware support.

**GENERIC** = 1

**MAINLINE** = 2

**NONE** = 0

**SERVICE\_PROVIDER** = 3

**class** `ironic_python_agent.hardware.HardwareType`

Bases: `object`

**MAC\_ADDRESS** = 'mac\_address'

**class** `ironic_python_agent.hardware.Memory`(*total, physical\_mb=None*)

Bases: `SerializableComparable`

**serializable\_fields** = ('total', 'physical\_mb')

**class** `ironic_python_agent.hardware.NetworkInterface`(*name, mac\_addr, ipv4\_address=None, ipv6\_address=None, has\_carrier=True, lldp=None, vendor=None, product=None, client\_id=None, biosdevname=None, speed\_mbps=None*)

Bases: `SerializableComparable`

**serializable\_fields** = ('name', 'mac\_address', 'ipv4\_address', 'ipv6\_address', 'has\_carrier', 'lldp', 'vendor', 'product', 'client\_id', 'biosdevname', 'speed\_mbps')

**class** `ironic_python_agent.hardware.SystemFirmware`(*vendor, version, build\_date*)

Bases: `SerializableComparable`

**serializable\_fields** = ('vendor', 'version', 'build\_date')

**class** `ironic_python_agent.hardware.SystemVendorInfo`(*product\_name, serial\_number, manufacturer, firmware*)

Bases: `SerializableComparable`

**serializable\_fields** = ('product\_name', 'serial\_number', 'manufacturer', 'firmware')

`class ironic_python_agent.hardware.USBInfo(product, vendor, handle)`

Bases: `SerializableComparable`

`serializable_fields = ('product', 'vendor', 'handle')`

`ironic_python_agent.hardware.cache_node(node)`

Store the node object in the hardware module.

Stores the node object in the hardware module to facilitate the access of a node information in the hardware extensions.

If the new node does not match the previously cached one, wait for the expected root device to appear.

### Parameters

**node** Ironic node object

`ironic_python_agent.hardware.check_versions(provided_version=None)`

Ensure the version of hardware managers hasnt changed.

### Parameters

**provided\_version** Hardware manager versions used by ironic.

### Raises

`errors.VersionMismatch` if any hardware manager version on the currently running agent doesnt match the one stored in `provided_version`.

### Returns

None

`ironic_python_agent.hardware.deduplicate_steps(candidate_steps)`

Remove duplicated clean or deploy steps

Deduplicates steps returned from HardwareManagers to prevent running a given step more than once. Other than individual step priority, it doesnt actually impact the deployment which specific steps are kept and what HardwareManager they are associated with. However, in order to make testing easier, this method returns deterministic results.

Uses the following filtering logic to decide which step wins:

- Keep the step that belongs to HardwareManager with highest HardwareSupport (larger int) value.
- If equal support level, keep the step with the higher defined priority (larger int).
- If equal support level and priority, keep the step associated with the HardwareManager whose name comes earlier in the alphabet.

### Parameters

**candidate\_steps** A dict containing all possible steps from all managers, key=manager, value=list of steps

### Returns

A deduplicated dictionary of {hardware\_manager: [steps]}

`ironic_python_agent.hardware.dispatch_to_all_managers(method, *args, **kwargs)`

Dispatch a method to all hardware managers.



Dispatches the given method in priority order as sorted by *get\_managers*. If the method doesn't exist or raises `IncompatibleHardwareMethodError`, it continues to the next hardware manager. All managers that have hardware support for this node will be called, and their responses will be added to a dictionary of the form `{HardwareManagerClassName: response}`.

**Parameters**

- **method** hardware manager method to dispatch
- **args** arguments to dispatched method
- **kwargs** keyword arguments to dispatched method

**Raises**

*errors.HardwareManagerMethodNotFound* if all managers raise `IncompatibleHardwareMethodError`.

**Returns**

a dictionary with keys for each hardware manager that returns a response and the value as a list of results from that hardware manager.

`ironic_python_agent.hardware.dispatch_to_managers(method, *args, **kwargs)`

Dispatch a method to best suited hardware manager.

Dispatches the given method in priority order as sorted by *get\_managers*. If the method doesn't exist or raises `IncompatibleHardwareMethodError`, it is attempted again with a more generic hardware manager. This continues until a method executes that returns any result without raising an `IncompatibleHardwareMethodError`.

**Parameters**

- **method** hardware manager method to dispatch
- **args** arguments to dispatched method
- **kwargs** keyword arguments to dispatched method

**Returns**

result of successful dispatch of method

**Raises**

- *HardwareManagerMethodNotFound* if all managers failed the method
- *HardwareManagerNotFound* if no valid hardware managers found

`ironic_python_agent.hardware.get_cached_node()`

Guard function around the module variable `NODE`.

`ironic_python_agent.hardware.get_component_devices(raid_device)`

Get the component devices of a Software RAID device.

Get the UUID of the md device and scan all other devices for the same md UUID.

**Parameters**

**raid\_device** A Software RAID block device name.

**Returns**

A list of the component devices.

`ironic_python_agent.hardware.get_current_versions()`

Fetches versions from all hardware managers.

**Returns**

Dict in the format {name: version} containing one entry for every hardware manager.

`ironic_python_agent.hardware.get_holder_disks(raid_device)`

Get the holder disks of a Software RAID device.

Examine an md device and return its underlying disks.

**Parameters**

**raid\_device** A Software RAID block device name.

**Returns**

A list of the holder disks.

`ironic_python_agent.hardware.get_managers()`

Get a list of hardware managers in priority order.

This exists as a backwards compatibility shim, returning a simple list of managers where expected. New usages should use `get_managers_detail`.

**Returns**

Priority-sorted list of hardware managers

**Raises**

*HardwareManagerNotFound* if no valid hardware managers found

`ironic_python_agent.hardware.get_managers_detail()`

Get detailed information about hardware managers

Use stevedore to find all eligible hardware managers, sort them based on self-reported (via `evaluate_hardware_support()`) priorities, and return a dict containing the manager object, its class name, and hardware support value. The resulting list is cached in `_global_managers`.

**Returns**

list of dictionaries representing hardware managers and metadata

**Raises**

*HardwareManagerNotFound* if no valid hardware managers found

`ironic_python_agent.hardware.get_multipath_status()`

Return the status of multipath initialization.

`ironic_python_agent.hardware.is_md_device(raid_device)`

Check if a device is an md device

Check if a device is a Software RAID (md) device.

**Parameters**

**raid\_device** A Software RAID block device name.

**Returns**

True if the device is an md device, False otherwise.

```
ironic_python_agent.hardware.list_all_block_devices(block_type='disk',  
                                                    ignore_raid=False,  
                                                    ignore_floppy=True,  
                                                    ignore_empty=True,  
                                                    ignore_multipath=False,  
                                                    all_serial_and_wnn=False)
```

List all physical block devices

The switches we use for lsblk: P for KEY=value output, b for size output in bytes, i to ensure ascii characters only, and o to specify the fields/columns we need.

Broken out as its own function to facilitate custom hardware managers that dont need to subclass GenericHardwareManager.

#### Parameters

- **block\_type** Type of block device to find
- **ignore\_raid** Ignore auto-identified raid devices, example: md0 Defaults to false as these are generally disk devices and should be treated as such if encountered.
- **ignore\_floppy** Ignore floppy disk devices in the block device list. By default, these devices are filtered out.
- **ignore\_empty** Whether to ignore disks with size equal 0.
- **ignore\_multipath** Whether to ignore devices backing multipath devices. Default is to consider multipath devices, if possible.
- **all\_serial\_and\_wnn** Dont collect serial and wwn numbers based on a priority order, instead collect wwn numbers from both udevadm and lsblk. When enabled this option will also collect both the short and the long serial from udevadm if possible.

#### Returns

A list of BlockDevices

```
ironic_python_agent.hardware.list_hardware_info(use_cache=True)
```

List hardware information with caching.

```
ironic_python_agent.hardware.md_get_raid_devices()
```

Get all discovered Software RAID (md) devices

#### Returns

A python dict containing details about the discovered RAID devices

```
ironic_python_agent.hardware.md_restart(raid_device)
```

Restart an md device

Stop and re-assemble a Software RAID (md) device.

#### Parameters

**raid\_device** A Software RAID block device name.

#### Raises

CommandExecutionError in case the restart fails.

`ironic_python_agent.hardware.safety_check_block_device`(*node*, *device*)

Performs safety checking of a block device before destroying.

In order to guard against destruction of file systems such as shared-disk file systems ([https://en.wikipedia.org/wiki/Clustered\\_file\\_system#SHARED-DISK](https://en.wikipedia.org/wiki/Clustered_file_system#SHARED-DISK)) or similar filesystems where multiple distinct computers may have unlocked concurrent IO access to the entire block device or SAN Logical Unit Number, we need to evaluate, and block cleaning from occurring on these filesystems *unless* we have been explicitly configured to do so.

This is because cleaning is an intentionally destructive operation, and once started against such a device, given the complexities of shared disk clustered filesystems where concurrent access is a design element, in all likelihood the entire cluster can be negatively impacted, and an operator will be forced to recover from snapshot and or backups of the volumes contents.

### Parameters

- **node** A node, or cached node object.
- **device** String representing the path to the block device to be checked.

### Raises

ProtectedDeviceError when a device is identified with one of these known clustered filesystems, and the overall settings have not indicated for the agent to skip such safety checks.

`ironic_python_agent.hardware.save_api_client`(*client=None*, *timeout=None*,  
*interval=None*)

Preserves access to the API client for potential later reuse.

`ironic_python_agent.hardware.update_cached_node`()

Attempts to update the node cache via the API

## `ironic_python_agent.inject_files` module

Implementation of the `inject_files` deploy step.

`ironic_python_agent.inject_files.find_partition_with_path`(*path*, *device=None*)

Find a partition with the given path.

### Parameters

- **path** Expected path.
- **device** Target device. If None, the root device is used.

### Returns

A context manager that will unmount and delete the temporary mount point on exit.

`ironic_python_agent.inject_files.inject_files`(*node*, *ports*, *files*, *verify\_ca=True*)

A deploy step to inject arbitrary files.

### Parameters

- **node** A dictionary of the node object
- **ports** A list of dictionaries containing information of ports for the node
- **files** See ARGINFO.

- **verify\_ca** Whether to verify TLS certificate.

**Raises**

InvalidCommandParamsError

### ironic\_python\_agent.inspect module

**class** `ironic_python_agent.inspect.IronicInspection`

Bases: Thread

Class for manual inspection functionality.

**backoff\_factor** = 2.7

**max\_delay** = 1200

**max\_jitter\_multiplier** = 1.2

**min\_jitter\_multiplier** = 0.7

**run()**

Run Inspection.

### ironic\_python\_agent.inspector module

`ironic_python_agent.inspector.call_inspector(data, failures)`

Post data to inspector.

`ironic_python_agent.inspector.collect_default(data, failures)`

The default inspection collector.

This is the only collector that is called by default. It collects the whole inventory as returned by the hardware manager(s).

It also tries to get BMC address, PXE boot device and the expected root device.

**Parameters**

- **data** mutable data that well send to inspector
- **failures** AccumulatedFailures object

`ironic_python_agent.inspector.collect_extra_hardware(data, failures)`

Collect detailed inventory using hardware-detect utility.

Recognizes ipa-inspection-benchmarks with list of benchmarks (possible values are cpu, disk, mem) to run. No benchmarks are run by default, as theyre pretty time-consuming.

Puts collected data as JSON under data key. Requires hardware python package to be installed on the ramdisk in addition to the packages in requirements.txt.

**Parameters**

- **data** mutable data that well send to inspector
- **failures** AccumulatedFailures object

`ironic_python_agent.inspector.collect_lldp(data, failures)`

Collect LLDP information for network interfaces.

### **Parameters**

- **data** mutable data that will send to inspector
- **failures** AccumulatedFailures object

`ironic_python_agent.inspector.collect_logs(data, failures)`

Collect system logs from the ramdisk.

As inspection runs before any nodes details are known, its handy to have logs returned with data. This collector sends logs to inspector in format expected by the ramdisk\_error plugin: base64 encoded tar.gz.

This collector should be installed last in the collector chain, otherwise it wont collect enough logs.

This collector does not report failures.

### **Parameters**

- **data** mutable data that will send to inspector
- **failures** AccumulatedFailures object

`ironic_python_agent.inspector.collect_pci_devices_info(data, failures)`

Collect a list of PCI devices.

Each PCI device entry in list is a dictionary containing vendor\_id and product\_id keys, which will be then used by the ironic inspector to distinguish various PCI devices.

The data is gathered from /sys/bus/pci/devices directory.

### **Parameters**

- **data** mutable data that will send to inspector
- **failures** AccumulatedFailures object

`ironic_python_agent.inspector.collect_usb_devices(data, failures)`

Collect USB information for connected devices.

### **Parameters**

- **data** mutable data that will send to inspector
- **failures** AccumulatedFailures object

`ironic_python_agent.inspector.extension_manager(names)`

`ironic_python_agent.inspector.inspect()`

Optionally run inspection on the current node.

If `inspection_callback_url` is set in the configuration, get the hardware inventory from the node and post it back to the inspector.

### **Returns**

node UUID if inspection was successful, None if associated node was not found in inspector cache. None is also returned if inspector support is not enabled.

`ironic_python_agent.inspector.wait_for_dhcp()`

Wait until NICs get their IP addresses via DHCP or timeout happens.

Depending on the value of `inspection_dhcp_all_interfaces` configuration option will wait for either all or only PXE booting NIC.

Note: only supports IPv4 addresses for now.

**Returns**

True if all NICs got IP addresses, False if timeout happened. Also returns True if waiting is disabled via configuration.

### `ironic_python_agent.ironic_api_client` module

**class** `ironic_python_agent.ironic_api_client.APIClient(api_urls)`

Bases: object

**agent\_token** = None

**api\_version** = 'v1'

**heartbeat**(*uuid*, *advertise\_address*, *advertise\_protocol*='http', *generated\_cert*=None)

**heartbeat\_api** = '/v1/heartbeat/{uuid}'

**lookup\_api** = '/v1/lookup'

**lookup\_lock\_pause** = 0

**lookup\_node**(*hardware\_info*, *timeout*, *starting\_interval*, *node\_uuid*=None, *max\_interval*=60)

**supports\_auto\_tls**()

### `ironic_python_agent.netutils` module

**class** `ironic_python_agent.netutils.RawPromiscuousSockets(interface_names, protocol)`

Bases: object

`ironic_python_agent.netutils.bring_up_vlan_interfaces(interfaces_list)`

Bring up vlan interfaces based on kernel params

Use the configured value of `enable_vlan_interfaces` to determine if VLAN interfaces should be brought up using `ip` commands. If `enable_vlan_interfaces` defines a particular vlan then bring up that vlan. If it defines an interface or all then use LLDP info to figure out which VLANs should be brought up.

**Parameters**

**interfaces\_list** List of current interfaces

**Returns**

List of vlan interface names that have been added

`ironic_python_agent.netutils.get_default_ip_addr(family, interface_id)`

Retrieve default IPv4, IPv6 or mac address.

`ironic_python_agent.netutils.get_hostname()`

`ironic_python_agent.netutils.get_ipv4_addr(interface_id)`

`ironic_python_agent.netutils.get_ipv6_addr(interface_id)`

`ironic_python_agent.netutils.get_lldp_info(interface_names)`

Get LLDP info from the switch(es) the agent is connected to.

Listens on either a single or all interfaces for LLDP packets, then parses them. If no LLDP packets are received before `lldp_timeout`, returns a dictionary in the form `{interface: [],}`.

### Parameters

**interface\_names** The interface to listen for packets on. If `None`, will listen on each interface.

### Returns

A dictionary in the form `{interface: [(lldp_type, lldp_data)],}`

`ironic_python_agent.netutils.get_mac_addr(interface_id)`

`ironic_python_agent.netutils.get_wildcard_address()`

**class** `ironic_python_agent.netutils.ifreq`

Bases: `Structure`

Class for setting flags on a socket.

**ifr\_flags**

`Structure/Union member`

**ifr\_ifrn**

`Structure/Union member`

`ironic_python_agent.netutils.interface_has_carrier(interface_name)`

`ironic_python_agent.netutils.is_bond(interface_name)`

`ironic_python_agent.netutils.is_network_device(interface_name)`

`ironic_python_agent.netutils.is_vlan(interface_name)`

`ironic_python_agent.netutils.list_interfaces()`

`ironic_python_agent.netutils.wrap_ipv6(ip)`

## ironic\_python\_agent.numa\_inspector module

`ironic_python_agent.numa_inspector.collect_numa_topology_info(data, failures)`

Collect the NUMA topology information.

The data is gathered from `/sys/devices/system/node/node<X>` and `/sys/class/net/` directories. The information is collected in the form of:

```
{
  "numa_topology": {
    "ram": [{"numa_node": <numa_node_id>, "size_kb": <memory_in_kb>},
           ...],
    "cpus": [
```

(continues on next page)



(continued from previous page)

```

    {
      "cpu": <cpu_id>, "numa_node": <numa_node_id>,
      "thread_siblings": [<list of sibling threads>]
    },
    ...,
  ],
  "nics": [
    {"name": "<network interface name>", "numa_node": <numa_node_id>},
    ...,
  ]
}
}

```

**Parameters**

- **data** mutable data that well send to inspector
- **failures** AccumulatedFailures object

**Returns**

None

`ironic_python_agent.numa_inspector.get_nodes_cores_info(numa_node_dirs)`

Collect the NUMA nodes cpus and threads information.

NUMA nodes path: `/sys/devices/system/node/node<node_id>`

Thread dirs path: `/sys/devices/system/node/node<node_id>/cpu<thread_id>`

**CPU id file path:** `/sys/devices/system/node/node<node_id>/cpu<thread_id>/topology/core_id`

The information is returned in the form of:

```

"cpus": [
  {
    "cpu": <cpu_id>, "numa_node": <numa_node_id>,
    "thread_siblings": [<list of sibling threads>]
  },
  ...,
]

```

**Parameters**

**numa\_node\_dirs** A list of NUMA node directories

**Raises**

`IncompatibleNumaFormatError`: when unexpected format data in NUMA node

**Returns**

A list of cpu information with NUMA node id and thread siblings

`ironic_python_agent.numa_inspector.get_nodes_memory_info(numa_node_dirs)`

Collect the NUMA nodes memory information.

The information is returned in the form of:

```
"ram": [{"numa_node": <numa_node_id>, "size_kb": <memory_in_kb>}, ...]
```

### Parameters

**numa\_node\_dirs** A list of NUMA node directories

### Raises

IncompatibleNumaFormatError: when unexpected format data in NUMA node

### Returns

A list of memory information with NUMA node id

`ironic_python_agent.numa_inspector.get_nodes_nics_info(nic_device_path)`

Collect the NUMA nodes nics information.

The information is returned in the form of:

```
"nics": [  
    {"name": "<network interface name>",  
      "numa_node": <numa_node_id>},  
    ...,  
]
```

### Parameters

**nic\_device\_path** nic device directory path

### Raises

IncompatibleNumaFormatError: when unexpected format data in NUMA node

### Returns

A list of nics information with NUMA node id

`ironic_python_agent.numa_inspector.get_numa_node_id(numa_node_dir)`

Provides the NUMA node id from NUMA node directory

### Parameters

**numa\_node\_dir** NUMA node directory

### Raises

IncompatibleNumaFormatError: when unexpected format data in NUMA node dir

### Returns

NUMA node id

## ironic\_python\_agent.partition\_utils module

Logic related to handling partitions.

Imported from ironic-libs disk\_utils as of the following commit: <https://opendev.org/openstack/ironic-lib/commit/9fb5be348202f4854a455cd08f400ae12b99e1f2>

`ironic_python_agent.partition_utils.create_config_drive_partition(node_uuid,  
 device,  
 configdrive)`

Create a partition for config drive

Checks if the device is GPT or MBR partitioned and creates config drive partition accordingly.

**Parameters**

- **node\_uuid** UUID of the Node.
- **device** The device path.
- **configdrive** Base64 encoded Gzipped configdrive content or configdrive HTTP URL.

**Raises**

InstanceDeployFailure if config drive size exceeds maximum limit or if it fails to create config drive.

`ironic_python_agent.partition_utils.get_configdrive(configdrive, node_uuid, tempdir=None)`

Get the information about size and location of the configdrive.

**Parameters**

- **configdrive** Base64 encoded Gzipped configdrive content or configdrive HTTP URL.
- **node\_uuid** Nodes uuid. Used for logging.
- **tempdir** temporary directory for the temporary configdrive file

**Raises**

InstanceDeployFailure if it cant download or decode the config drive.

**Returns**

A tuple with the size in MiB and path to the uncompressed configdrive file.

`ironic_python_agent.partition_utils.get_labelled_partition(device_path, label, node_uuid)`

Check and return if partition with given label exists

**Parameters**

- **device\_path** The device path.
- **label** Partition label
- **node\_uuid** UUID of the Node. Used for logging.

**Raises**

InstanceDeployFailure, if any disk partitioning related commands fail.

**Returns**

block device file for partition if it exists; otherwise it returns None.

`ironic_python_agent.partition_utils.get_partition(device, uuid)`

Find the partition of a given device.

```
ironic_python_agent.partition_utils.work_on_disk(dev, root_mb, swap_mb,
                                                ephemeral_mb, ephemeral_format,
                                                image_path, node_uuid,
                                                preserve_ephemeral=False,
                                                configdrive=None, boot_mode='bios',
                                                tempdir=None, disk_label=None,
                                                cpu_arch="", conv_flags=None,
                                                source_format=None, is_raw=False)
```

Create partitions and copy an image to the root partition.

### Parameters

- **dev** Path for the device to work on.
- **root\_mb** Size of the root partition in megabytes.
- **swap\_mb** Size of the swap partition in megabytes.
- **ephemeral\_mb** Size of the ephemeral partition in megabytes. If 0, no ephemeral partition will be created.
- **ephemeral\_format** The type of file system to format the ephemeral partition.
- **image\_path** Path for the instances disk image. If None, the root partition is prepared but not populated.
- **node\_uuid** nodes uuid. Used for logging.
- **preserve\_ephemeral** If True, no filesystem is written to the ephemeral block device, preserving whatever content it had (if the partition table has not changed).
- **configdrive** Optional. Base64 encoded Gzipped configdrive content or configdrive HTTP URL.
- **boot\_mode** Can be bios or uefi. bios by default.
- **tempdir** A temporary directory
- **disk\_label** The disk label to be used when creating the partition table. Valid values are: msdos, gpt or None; If None Ironic will figure it out according to the boot\_mode parameter.
- **cpu\_arch** Architecture of the node the disk device belongs to. When using the default value of None, no architecture specific steps will be taken. This default should be used for x86\_64. When set to ppc64\*, architecture specific steps are taken for booting a partition image locally.
- **conv\_flags** Flags that need to be sent to the dd command, to control the conversion of the original file when copying to the host. It can contain several options separated by commas.
- **source\_format** The format of the disk image to be written. If set, must be raw or the actual disk format of the image.
- **is\_raw** Ironic indicator image is raw; not to be converted

### Returns

a dictionary containing the following keys: root uuid: UUID of root partition efi system partition uuid: UUID of the uefi system partition (if boot mode is uefi).

*partitions*: mapping of partition types to their device paths. NOTE: If key exists but value is None, it means partition doesn't exist.

### ironic\_python\_agent.qemu\_img module

`ironic_python_agent.qemu_img.convert_image`(*source*, *dest*, *out\_format*, *cache=None*,  
*out\_of\_order=False*, *sparse\_size=None*,  
*source\_format=None*)

Convert image to other format.

This method is only to be run against images who have passed `format_inspectors` safety check, and with the format reported by it passed in. Any other usage is a major security risk.

`ironic_python_agent.qemu_img.image_info`(*path*, *source\_format=None*)

Return an object containing the parsed output from `qemu-img info`.

This must only be called on images already validated as safe by the format inspector.

#### Parameters

- **path** The path to an image you need information on
- **source\_format** The format of the source image. If this is omitted when deep inspection is enabled, this will raise `InvalidImage`.

### ironic\_python\_agent.raid\_utils module

`ironic_python_agent.raid_utils.calc_raid_partition_sectors`(*psize*, *start*)

Calculates end sector and converts start and end sectors including

the unit of measure, compatible with parted. :param psize: size of the raid partition :param start: start sector of the raid partition in integer format :return: start and end sector in parted compatible format, end sector

as integer

`ironic_python_agent.raid_utils.calculate_raid_start`(*target\_boot\_mode*,  
*partition\_table\_type*, *dev\_name*)

Define the start sector for the raid partition.

#### Parameters

- **target\_boot\_mode** the node boot mode.
- **partition\_table\_type** the node partition label, gpt or msdos.
- **dev\_name** block device in the raid configuration.

#### Returns

The start sector for the raid partition.

`ironic_python_agent.raid_utils.create_raid_device`(*index*, *logical\_disk*)

Create a raid device.

#### Parameters

- **index** the index of the resulting md device.
- **logical\_disk** the logical disk containing the devices used to create the raid.

**Raise**

errors.`SoftwareRAIDError` if not able to create the raid device or fails to re-add a device to a raid.

`ironic_python_agent.raid_utils.create_raid_partition_tables`(*block\_devices*,  
*partition\_table\_type*,  
*target\_boot\_mode*)

Creates partition tables in all disks in a RAID configuration and

reports the starting sector for each partition on each disk. :param `block_devices`: disks where we want to create the partition tables. :param `partition_table_type`: type of partition table to create, for example

`gpt` or `msdos`.

**Parameters**

**`target_boot_mode`** the node selected boot mode, for example `uefi` or `bios`.

**Returns**

a dictionary of devices and the start of the corresponding partition.

`ironic_python_agent.raid_utils.find_esp_raid`()

Find the ESP md device in case of a rebuild.

`ironic_python_agent.raid_utils.get_block_devices_for_raid`(*block\_devices*,  
*logical\_disks*)

Get block devices that are involved in the RAID configuration.

This call does two things: \* Collect all block devices that are involved in RAID. \* Update each logical disks with suitable block devices.

`ironic_python_agent.raid_utils.get_next_free_raid_device`()

Get a device name that is still free.

`ironic_python_agent.raid_utils.get_volume_name_of_raid_device`(*raid\_device*)

Get the volume name of a RAID device

**Parameters**

**`raid_device`** A Software RAID block device name.

**Returns**

volume name of the device, or `None`

`ironic_python_agent.raid_utils.prepare_boot_partitions_for_softraid`(*device*,  
*holders*,  
*efi\_part*, *target\_boot\_mode*)

Prepare boot partitions when relevant.

Create either a RAIDed EFI partition or bios boot partitions for software RAID, according to both target boot mode and disk holders partition table types.

**Parameters**

- **`device`** the softraid device path
- **`holders`** the softraid drive members

- **efi\_part** when relevant the efi partition coming from the image deployed on softraid device, can be/is often None
- **target\_boot\_mode** target boot mode can be bios/uefi/None or anything else for unspecified

**Returns**

the path to the ESP md device when target boot mode is uefi, nothing otherwise.

**ironic\_python\_agent.tls\_utils module**

**class** `ironic_python_agent.tls_utils.TlsCertificate`(*text, path, private\_key\_path*)

Bases: tuple

**path**

Alias for field number 1

**private\_key\_path**

Alias for field number 2

**text**

Alias for field number 0

`ironic_python_agent.tls_utils.generate_tls_certificate`(*ip\_address,*  
*common\_name=None,*  
*valid\_for\_days=90*)

Generate a self-signed TLS certificate.

**Parameters**

- **ip\_address** IP address the certificate will be valid for.
- **common\_name** Content for the common name field (e.g. host name). Defaults to the current host name.
- **valid\_for\_days** Number of days the certificate will be valid for.

**Returns**

a TlsCertificate object.

**ironic\_python\_agent.utils module**

**class** `ironic_python_agent.utils.AccumulatedFailures`(*exc\_class=<class*  
*'RuntimeError'>*)

Bases: object

Object to accumulate failures without raising exception.

**add**(*fail, \*fmt*)

Add failure with optional formatting.

**Parameters**

- **fail** exception or error string
- **fmt** formatting arguments (only if fail is a string)

**get\_error()**

Get error string or None.

**raise\_if\_needed()**

Raise exception if error list is not empty.

**Raises**

RuntimeError

**class** `ironic_python_agent.utils.StreamingClient`(*verify\_ca=True*)

Bases: object

A wrapper around HTTP client with TLS, streaming and error handling.

`ironic_python_agent.utils.collect_system_logs`(*journald\_max\_lines=None*)

Collect system logs.

Collect system logs, for distributions using systemd the logs will come from journald. On other distributions the logs will come from the /var/log directory and dmesg output.

**Parameters**

**journald\_max\_lines** Maximum number of lines to retrieve from the journald. if None, return everything.

**Returns**

A tar, gzip base64 encoded string with the logs.

`ironic_python_agent.utils.copy_config_from_vmedia`()

Copies any configuration from a virtual media device.

Copies files under /etc/ironic-python-agent and /etc/ironic-python-agent.d.

`ironic_python_agent.utils.create_partition_table`(*dev\_name, partition\_table\_type*)

Create a partition table on a disk using parted.

**Parameters**

- **dev\_name** the disk where we want to create the partition table.
- **partition\_table\_type** the type of partition table we want to create, for example gpt or msdos.

**Raises**

CommandExecutionError if an error is encountered while attempting to create the partition table.

`ironic_python_agent.utils.determine_time_method`()

Helper method to determine what time utility is present.

**Returns**

ntupdate if ntpdate has been found, chrony if chrony was located, and None if neither are located. If both tools are present, chrony will supersede ntpdate.

`ironic_python_agent.utils.execute`(\**cmd*, \*\**kwargs*)

Convenience wrapper around ironic\_libs execute() method.

Executes and logs results from a system command.



`ironic_python_agent.utils.extract_device(part)`

Extract the device from a partition name or path.

**Parameters**

**part** the partition

**Returns**

a device if success, None otherwise

`ironic_python_agent.utils.find_in_lshw(lshw, by_id=None, by_class=None, recursive=False, **fields)`

Yield all suitable records from lshw.

`ironic_python_agent.utils.get_agent_params()`

Gets parameters passed to the agent via kernel cmdline or vmedia.

Parameters can be passed using either the kernel commandline or through virtual media. If `boot_method` is `vmedia`, merge params provided via `vmedia` with those read from the kernel command line.

Although it should never happen, if a variable is both set by `vmedia` and kernel command line, the setting in `vmedia` will take precedence.

**Returns**

a dict of potential configuration parameters for the agent

`ironic_python_agent.utils.get_command_output(command)`

Return the output of a given command.

**Parameters**

**command** The command to be executed.

**Raises**

`CommandExecutionError` if the execution of the command fails.

**Returns**

A BytesIO string with the output.

`ironic_python_agent.utils.get_journalctl_output(lines=None, units=None)`

Query the contents of the systemd journal.

**Parameters**

- **lines** Maximum number of lines to retrieve from the logs. If `None`, return everything.
- **units** A list with the names of the units we should retrieve the logs from. If `None` retrieve the logs for everything.

**Returns**

A log string.

`ironic_python_agent.utils.get_node_boot_mode(node)`

Returns the node boot mode.

It returns `uefi` if `secure_boot` is set to `true` in `instance_info/capabilities` of `node`. Otherwise it directly look for boot mode hints into

**Parameters**

**node** dictionary.

**Returns**

bios or uefi

`ironic_python_agent.utils.get_partition_table_type_from_specs(node)`

Returns the node partition label, gpt or msdos.

If boot mode is uefi, return gpt. Else, choice is open, look for disk\_label capabilities (instance\_info has priority over properties).

**Parameters**

**node**

**Returns**

gpt or msdos

`ironic_python_agent.utils.get_ssl_client_options(conf)`

Format SSL-related requests options.

**Parameters**

**conf** oslo\_config CONF object

**Returns**

tuple of verify and cert values to pass to requests

`ironic_python_agent.utils.guess_root_disk(block_devices,  
min_size_required=4294967296)`

Find suitable disk provided that root device hints are not given.

If no hints are passed, order the devices by size (primary key) and name (secondary key), and return the first device larger than min\_size\_required as the root disk.

`ironic_python_agent.utils.gzip_and_b64encode(io_dict=None, file_list=None)`

Gzip and base64 encode files and BytesIO buffers.

**Parameters**

- **io\_dict** A dictionary containing whose the keys are the file names and the value a BytesIO object.
- **file\_list** A list of file path.

**Returns**

A gzipped and base64 encoded string.

`ironic_python_agent.utils.is_char_device(path)`

Check if the specified path is a character device.

`ironic_python_agent.utils.is_journalctl_present()`

Check if the journalctl command is present.

**Returns**

True if journalctl is present, False if not.

`ironic_python_agent.utils.log_early_log_to_logger()`

Logs early logging events to the configured logger.

`ironic_python_agent.utils.mounted(source, dest=None, opts=None, fs_type=None,  
mount_attempts=1, umount_attempts=3)`

A context manager for a temporary mount.

**Parameters**

- **source** A device to mount.
- **dest** Mount destination. If not specified, a temporary directory will be created and removed afterwards. An existing destination is not removed.
- **opts** Mount options (-o argument).
- **fs\_type** File system type (-t argument).
- **mount\_attempts** A number of attempts to mount the device.
- **umount\_attempts** A number of attempts to unmount the device.

**Returns**

A generator yielding the destination.

`ironic_python_agent.utils.parse_capabilities(root)`

Extract capabilities from provided root dictionary-behaving object.

`root.get(capabilities, {})` value can either be a dict, or a json str, or a key1:value1,key2:value2 formatted string.

**Parameters**

**root** Anything behaving like a dict and containing capabilities formatted as expected. Can be `node.get(properties, {})`, `node.get(instance_info, {})`.

**Returns**

A dictionary with the capabilities if found and well formatted, otherwise an empty dictionary.

`ironic_python_agent.utils.remove_large_keys(var)`

Remove specific keys from the var, recursing into dicts and lists.

`ironic_python_agent.utils.rescan_device(device)`

Force the device to be rescanned

**Parameters**

**device** device upon which to rescan and update kernel partition records.

`ironic_python_agent.utils.split_device_and_partition_number(part)`

Extract the partition number from a partition name or path.

**Parameters**

**part** the partition

**Returns**

device and partition number if success, None otherwise

`ironic_python_agent.utils.sync_clock(ignore_errors=False)`

Syncs the software clock of the system.

This method syncs the system software clock if a NTP server was defined in the [DEFAULT]ntp\_server configuration parameter. This method does NOT attempt to sync the hardware clock.

It will try to use either ntpdate or chrony to sync the software clock of the system. If neither is found, an exception is raised.

### **Parameters**

**ignore\_errors** Boolean value default False that allows for the method to be called and ultimately not raise an exception. This may be useful for opportunistically attempting to sync the system software clock.

### **Raises**

CommandExecutionError if an error is encountered while attempting to sync the software clock.

`ironic_python_agent.utils.try_collect_command_output(io_dict, file_name, command)`

## **ironic\_python\_agent.version module**

### **Module contents**

## INDICES AND TABLES

- genindex
- search



## PYTHON MODULE INDEX

.

- `ironic_python_agent.agent`, 35
- `ironic_python_agent.api`, 27
- `ironic_python_agent.api.app`, 26
- `ironic_python_agent.burnin`, 38
- `ironic_python_agent.cmd`, 27
- `ironic_python_agent.cmd.agent`, 27
- `ironic_python_agent.cmd.inspect`, 27
- `ironic_python_agent.config`, 39
- `ironic_python_agent.disk_partitioner`, 39
- `ironic_python_agent.disk_utils`, 40
- `ironic_python_agent.dmi_inspector`, 44
- `ironic_python_agent.efi_utils`, 45
- `ironic_python_agent.encoding`, 46
- `ironic_python_agent.errors`, 47
- `ironic_python_agent.extensions`, 34
- `ironic_python_agent.extensions.base`, 28
- `ironic_python_agent.extensions.clean`, 30
- `ironic_python_agent.extensions.deploy`, 30
- `ironic_python_agent.extensions.flow`, 31
- `ironic_python_agent.extensions.image`, 31
- `ironic_python_agent.extensions.log`, 31
- `ironic_python_agent.extensions.poll`, 32
- `ironic_python_agent.extensions.rescue`, 32
- `ironic_python_agent.extensions.service`, 32
- `ironic_python_agent.extensions.standby`, 33
- `ironic_python_agent.extensions.system`, 34
- `ironic_python_agent.hardware`, 51
- `ironic_python_agent.hardware_managers`, 35
- `ironic_python_agent.hardware_managers.cna`, 34
- `ironic_python_agent.hardware_managers.mlnx`, 35
- `ironic_python_agent.inject_files`, 72
- `ironic_python_agent.inspect`, 73
- `ironic_python_agent.inspector`, 73
- `ironic_python_agent.ironic_api_client`, 75
- `ironic_python_agent.netutils`, 75
- `ironic_python_agent.numa_inspector`, 76
- `ironic_python_agent.partition_utils`, 78
- `ironic_python_agent.qemu_img`, 81
- `ironic_python_agent.raid_utils`, 81
- `ironic_python_agent.tls_utils`, 83
- `ironic_python_agent.utils`, 83
- `ironic_python_agent.version`, 88

i

- `ironic_python_agent`, 88





## INDEX

### A

**AccumulatedFailures** (class in *ironic\_python\_agent.utils*), 83

**add()** (*ironic\_python\_agent.utils.AccumulatedFailures* method), 83

**add\_boot\_record()** (in module *ironic\_python\_agent.efi\_utils*), 45

**add\_partition()** (*ironic\_python\_agent.disk\_partitioner.DiskPartitioner* method), 39

**agent\_token** (*ironic\_python\_agent.ironic\_api\_client.APIClient* attribute), 75

**AgentCommandStatus** (class in *ironic\_python\_agent.extensions.base*), 28

**AgentIsBusy**, 47

**any\_ipmi\_device\_exists()** (*ironic\_python\_agent.hardware.GenericHardwareManager* method), 52

**api\_get\_command()** (*ironic\_python\_agent.api.app.Application* method), 26

**api\_list\_commands()** (*ironic\_python\_agent.api.app.Application* method), 26

**api\_root()** (*ironic\_python\_agent.api.app.Application* method), 26

**api\_run\_command()** (*ironic\_python\_agent.api.app.Application* method), 27

**api\_status()** (*ironic\_python\_agent.api.app.Application* method), 27

**api\_v1()** (*ironic\_python\_agent.api.app.Application* method), 27

**api\_version** (*ironic\_python\_agent.ironic\_api\_client.APIClient* attribute), 75

**APIClient** (class in *ironic\_python\_agent.ironic\_api\_client*), 75

**Application** (class in *ironic\_python\_agent.api.app*), 26

**apply\_configuration()** (*ironic\_python\_agent.hardware.GenericHardwareManager* method), 52

**async\_command()** (in module *ironic\_python\_agent.extensions.base*), 29

**AsyncCommandResult** (class in *ironic\_python\_agent.extensions.base*), 28

### B

**backoff\_factor** (*ironic\_python\_agent.inspect.IronicInspection* attribute), 73

**BaseAgentExtension** (class in *ironic\_python\_agent.extensions.base*), 28

**BaseCommandResult** (class in *ironic\_python\_agent.extensions.base*), 28

**block\_uuid()** (in module *ironic\_python\_agent.disk\_utils*), 40

**BlockDevice** (class in *ironic\_python\_agent.hardware*), 51

**BlockDeviceEraseError**, 47

**BlockDeviceError**, 47

**BootInfo** (class in *ironic\_python\_agent.hardware*), 52

**bring\_up\_vlan\_interfaces()** (in module *ironic\_python\_agent.netutils*), 75

**burnin\_cpu()** (*ironic\_python\_agent.hardware.GenericHardwareManager* method), 52

**burnin\_disk()** (*ironic\_python\_agent.hardware.GenericHardwareManager* method), 52

**burnin\_memory()** (*ironic\_python\_agent.hardware.GenericHardwareManager* method), 53

**burnin\_network()** (*ironic\_python\_agent.hardware.GenericHardwareManager* method), 53

**bytes\_transferred** (*ironic\_python\_agent.extensions.standby.ImageDownload* property), 33

### C

**cache\_node()** (in module

*ironic\_python\_agent.hardware*), 68

`calc_raid_partition_sectors()` (in module *ironic\_python\_agent.raid\_utils*), 81

`calculate_raid_start()` (in module *ironic\_python\_agent.raid\_utils*), 81

`call_inspector()` (in module *ironic\_python\_agent.inspector*), 73

`check_cmd_presence()` (*ironic\_python\_agent.extensions.base.BaseCommandExecutionError*, 47 method), 28

`check_md5_enabled()` (in module *ironic\_python\_agent.extensions.standby*), 34

`check_versions()` (in module *ironic\_python\_agent.hardware*), 68

`clean_boot_records()` (in module *ironic\_python\_agent.efi\_utils*), 45

`clean_uefi_nvram()` (*ironic\_python\_agent.hardware.GenericHardwareManager* method), 53

`CleanExtension` (class in *ironic\_python\_agent.extensions.clean*), 30

`CleaningError`, 47

`ClockSyncError`, 47

`collect_default()` (in module *ironic\_python\_agent.inspector*), 73

`collect_dmidecode_info()` (in module *ironic\_python\_agent.dmi\_inspector*), 44

`collect_extra_hardware()` (in module *ironic\_python\_agent.inspector*), 73

`collect_lldp()` (in module *ironic\_python\_agent.inspector*), 73

`collect_lldp_data()` (*ironic\_python\_agent.hardware.GenericHardwareManager* method), 53

`collect_lldp_data()` (*ironic\_python\_agent.hardware.HardwareManager* method), 61

`collect_logs()` (in module *ironic\_python\_agent.inspector*), 74

`collect_numa_topology_info()` (in module *ironic\_python\_agent.numa\_inspector*), 76

`collect_pci_devices_info()` (in module *ironic\_python\_agent.inspector*), 74

`collect_system_logs()` (in module *ironic\_python\_agent.utils*), 84

`collect_system_logs()` (*ironic\_python\_agent.extensions.log.LogExtension* method), 31

`collect_system_logs()` (*ironic\_python\_agent.hardware.GenericHardwareManager* method), 53

`collect_system_logs()` (*ironic\_python\_agent.hardware.HardwareManager* method), 61

`collect_usb_devices()` (in module *ironic\_python\_agent.inspector*), 74

`CommandExecutionError`, 47

`commit()` (*ironic\_python\_agent.disk\_partitioner.DiskPartitioner* method), 39

`content_length` (*ironic\_python\_agent.extensions.standby.ImageDownload* property), 33

`convert_image()` (in module *ironic\_python\_agent.qemu\_img*), 81

`copy_config_from_vmedia()` (in module *ironic\_python\_agent.utils*), 84

`create_boot_loader_partitions()` (in module *ironic\_python\_agent.disk\_utils*), 40

`CPU` (class in *ironic\_python\_agent.hardware*), 52

`CPUCore` (class in *ironic\_python\_agent.hardware*), 52

`create_config_drive_partition()` (in module *ironic\_python\_agent.partition\_utils*), 78

`create_configuration()` (*ironic\_python\_agent.hardware.GenericHardwareManager* method), 53

`create_cpu_info_dict()` (*ironic\_python\_agent.hardware.GenericHardwareManager* static method), 54

`create_partition_table()` (in module *ironic\_python\_agent.utils*), 84

`create_raid_device()` (in module *ironic\_python\_agent.raid\_utils*), 81

`create_raid_partition_tables()` (in module *ironic\_python\_agent.raid\_utils*), 82

## D

`dd()` (in module *ironic\_python\_agent.disk\_utils*), 40

`deduplicate_steps()` (in module *ironic\_python\_agent.hardware*), 68

`default()` (*ironic\_python\_agent.encoding.RESTJSONEncoder* method), 46

`delete_configuration()` (*ironic\_python\_agent.hardware.GenericHardwareManager* method), 54

`DeployExtension` (class in *ironic\_python\_agent.extensions.deploy*),

30

DeploymentError, 47

destroy\_disk\_metadata() (in module *ironic\_python\_agent.disk\_utils*), 40

details (*ironic\_python\_agent.errors.NotFound* attribute), 50

details (*ironic\_python\_agent.errors.RESTError* attribute), 50

details\_str (*ironic\_python\_agent.errors.ImageChecksumError* attribute), 48

determine\_time\_method() (in module *ironic\_python\_agent.utils*), 84

DeviceNotFound, 47

DiskPartitioner (class in *ironic\_python\_agent.disk\_partitioner*), 39

dispatch\_to\_all\_managers() (in module *ironic\_python\_agent.hardware*), 68

dispatch\_to\_managers() (in module *ironic\_python\_agent.hardware*), 69

do\_heartbeat() (*ironic\_python\_agent.agent.IronicPythonAgentHeartbeat* method), 37

## E

encode() (*ironic\_python\_agent.encoding.RESTJSONEncoder* method), 46

erase\_block\_device() (*ironic\_python\_agent.hardware.GenericHardwareManager* method), 54

erase\_block\_device() (*ironic\_python\_agent.hardware.HardwareManager* method), 61

erase\_devices() (*ironic\_python\_agent.hardware.HardwareManager* method), 62

erase\_devices\_express() (*ironic\_python\_agent.hardware.GenericHardwareManager* method), 55

erase\_devices\_metadata() (*ironic\_python\_agent.hardware.GenericHardwareManager* method), 55

erase\_pstore() (*ironic\_python\_agent.hardware.GenericHardwareManager* method), 55

evaluate\_hardware\_support() (*ironic\_python\_agent.hardware.GenericHardwareManager* method), 55

evaluate\_hardware\_support() (*ironic\_python\_agent.hardware.HardwareManager* method), 62

evaluate\_hardware\_support() (*ironic\_python\_agent.hardware\_managers.cna.IntelCnaHardwareManager* method), 35

evaluate\_hardware\_support() (*ironic\_python\_agent.hardware\_managers.mlnx.MellanoxHardwareManager* method), 35

execute() (in module *ironic\_python\_agent.utils*), 84

execute\_clean\_step() (*ironic\_python\_agent.extensions.base.BaseAgentExtension* method), 28

execute\_clean\_step() (*ironic\_python\_agent.extensions.clean.CleanExtension* method), 30

execute\_command() (*ironic\_python\_agent.extensions.base.ExecuteCommandMixin* method), 29

execute\_deploy\_step() (*ironic\_python\_agent.extensions.deploy.DeployExtension* method), 30

execute\_service\_step() (*ironic\_python\_agent.extensions.service.ServiceExtension* method), 32

ExecuteCommandMixin (class in *ironic\_python\_agent.extensions.base*), 29

extension\_manager() (in module *ironic\_python\_agent.inspector*), 74

ExtensionError, 47

extract\_device() (in module *ironic\_python\_agent.utils*), 84

## F

FAILED (*ironic\_python\_agent.extensions.base.AgentCommandStatus* attribute), 28

finalize\_rescue() (*ironic\_python\_agent.extensions.rescue.RescueExtension* method), 32

find\_efi\_partition() (in module *ironic\_python\_agent.disk\_utils*), 40

find\_esp\_raid() (in module *ironic\_python\_agent.raid\_utils*), 82

find\_multipart() (in module *ironic\_python\_agent.utils*), 85

find\_partition\_with\_path() (in module *ironic\_python\_agent.inject\_files*), 72

fio\_disk() (in module *ironic\_python\_agent.burnin*), 38

flow\_manager() (in module *ironic\_python\_agent.burnin*), 38

fix\_gpt\_partition() (in module *ironic\_python\_agent.disk\_utils*), 40

FlowExtension (class in

*ironic\_python\_agent.extensions.flow*), 31  
**force\_heartbeat()** (*ironic\_python\_agent.agent.IronicPythonAgent* method), 36  
**force\_heartbeat()** (*ironic\_python\_agent.agent.IronicPythonAgentHeartbeatHandler* method), 37  
**format\_exception()** (in module *ironic\_python\_agent.api.app*), 27  
**from\_config()** (*ironic\_python\_agent.agent.IronicPythonAgent* class method), 36  
**full\_sync()** (*ironic\_python\_agent.hardware.GenericHardwareManager* method), 55  
**full\_sync()** (*ironic\_python\_agent.hardware.HardwareManager* method), 62  
**G**  
**generate\_tls\_certificate()** (in module *ironic\_python\_agent.tls\_utils*), 83  
**generate\_tls\_certificate()** (*ironic\_python\_agent.hardware.GenericHardwareManager* method), 55  
**generate\_tls\_certificate()** (*ironic\_python\_agent.hardware.HardwareManager* method), 62  
**GENERIC** (*ironic\_python\_agent.hardware.HardwareSupport* attribute), 67  
**GenericHardwareManager** (class in *ironic\_python\_agent.hardware*), 52  
**get\_agent\_params()** (in module *ironic\_python\_agent.utils*), 85  
**get\_and\_validate\_image\_format()** (in module *ironic\_python\_agent.disk\_utils*), 41  
**get\_bios\_given\_nic\_name()** (*ironic\_python\_agent.hardware.GenericHardwareManager* method), 55  
**get\_block\_devices\_for\_raid()** (in module *ironic\_python\_agent.raid\_utils*), 82  
**get\_bmc\_address()** (*ironic\_python\_agent.hardware.GenericHardwareManager* method), 56  
**get\_bmc\_address()** (*ironic\_python\_agent.hardware.HardwareManager* method), 62  
**get\_bmc\_mac()** (*ironic\_python\_agent.hardware.GenericHardwareManager* method), 56  
**get\_bmc\_mac()** (*ironic\_python\_agent.hardware.HardwareManager* method), 62  
**get\_bmc\_v6address()** (*ironic\_python\_agent.hardware.GenericHardwareManager* method), 56  
**get\_bmc\_v6address()** (*ironic\_python\_agent.hardware.HardwareManager* method), 62  
**get\_boot\_info()** (*ironic\_python\_agent.hardware.GenericHardwareManager* method), 56  
**get\_boot\_info()** (*ironic\_python\_agent.hardware.HardwareManager* method), 62  
**get\_boot\_records()** (in module *ironic\_python\_agent.efi\_utils*), 45  
**get\_cached\_node()** (in module *ironic\_python\_agent.hardware*), 69  
**get\_clean\_steps()** (*ironic\_python\_agent.extensions.clean.CleanExtension* method), 30  
**get\_clean\_steps()** (*ironic\_python\_agent.hardware.GenericHardwareManager* method), 56  
**get\_clean\_steps()** (*ironic\_python\_agent.hardware.HardwareManager* method), 62  
**get\_clean\_steps()** (*ironic\_python\_agent.hardware\_managers.mlnx.MellanoxSupport* method), 35  
**get\_command\_output()** (in module *ironic\_python\_agent.utils*), 85  
**get\_command\_result()** (*ironic\_python\_agent.agent.IronicPythonAgent* method), 36  
**get\_component\_devices()** (in module *ironic\_python\_agent.hardware*), 69  
**get\_configdrive()** (in module *ironic\_python\_agent.partition\_utils*), 69  
**get\_cpu\_cores()** (*ironic\_python\_agent.hardware.GenericHardwareManager* method), 57  
**get\_cpus()** (*ironic\_python\_agent.hardware.GenericHardwareManager* method), 57  
**get\_cpus()** (*ironic\_python\_agent.hardware.HardwareManager* method), 63  
**get\_current\_versions()** (in module *ironic\_python\_agent.hardware*), 69  
**get\_default\_monaddr()** (in module *ironic\_python\_agent.netutils*), 75  
**get\_deploy\_steps()** (*ironic\_python\_agent.extensions.deploy.DeployExtension* method), 30  
**get\_deploy\_steps()** (*ironic\_python\_agent.hardware.GenericHardwareManager* method), 56

method), 57

get\_deploy\_steps() (in module *ironic\_python\_agent.hardware.HardwareManager* method), 63

get\_deploy\_steps() (in module *ironic\_python\_agent.hardware\_managers.mellanox.DeviceHardwareManager* method), 35

get\_dev\_byte\_size() (in module *ironic\_python\_agent.disk\_utils*), 41

get\_dev\_sector\_size() (in module *ironic\_python\_agent.disk\_utils*), 41

get\_device\_information() (in module *ironic\_python\_agent.disk\_utils*), 41

get\_disk\_identifier() (in module *ironic\_python\_agent.disk\_utils*), 41

get\_error() (in module *ironic\_python\_agent.utils.Accumulator* method), 83

get\_extension() (in module *ironic\_python\_agent.extensions.base*), 29

get\_extension() (in module *ironic\_python\_agent.extensions.base.ExecuteCommandMixin* method), 29

get\_hardware\_info() (in module *ironic\_python\_agent.extensions.poll.PollExtension* method), 32

get\_holder\_disks() (in module *ironic\_python\_agent.hardware*), 70

get\_hostname() (in module *ironic\_python\_agent.netutils*), 75

get\_interface\_info() (in module *ironic\_python\_agent.hardware.GenericHardwareManager* method), 58

get\_interface\_info() (in module *ironic\_python\_agent.hardware.HardwareManager* method), 64

get\_interface\_info() (in module *ironic\_python\_agent.hardware\_managers.mlnx.MellanoxDeviceHardwareManager* method), 35

get\_ipv4\_addr() (in module *ironic\_python\_agent.netutils*), 75

get\_ipv4\_addr() (in module *ironic\_python\_agent.hardware.GenericHardwareManager* method), 58

get\_ipv6\_addr() (in module *ironic\_python\_agent.netutils*), 76

get\_ipv6\_addr() (in module *ironic\_python\_agent.hardware.GenericHardwareManager* method), 58

get\_journalctl\_output() (in module *ironic\_python\_agent.utils*), 85

get\_labelled\_partition() (in module *ironic\_python\_agent.partition\_utils*), 79

get\_lldp\_info() (in module *ironic\_python\_agent.netutils*), 76

get\_mac\_addr() (in module *ironic\_python\_agent.netutils*), 76

get\_managers\_detail() (in module *ironic\_python\_agent.hardware*), 70

get\_memory() (in module *ironic\_python\_agent.hardware.GenericHardwareManager* method), 58

get\_memory() (in module *ironic\_python\_agent.hardware.HardwareManager* method), 64

get\_multipath\_status() (in module *ironic\_python\_agent.hardware*), 70

get\_next\_free\_raid\_device() (in module *ironic\_python\_agent.raid\_utils*), 82

get\_node\_boot\_mode() (in module *ironic\_python\_agent.utils*), 85

get\_node\_uuid() (in module *ironic\_python\_agent.agent.IronicPythonAgent* method), 36

get\_nodes\_cores\_info() (in module *ironic\_python\_agent.numa\_inspector*), 77

get\_nodes\_memory\_info() (in module *ironic\_python\_agent.numa\_inspector*), 77

get\_nodes\_nics\_info() (in module *ironic\_python\_agent.numa\_inspector*), 78

get\_numa\_node\_id() (in module *ironic\_python\_agent.numa\_inspector*), 78

get\_os\_install\_device() (in module *ironic\_python\_agent.hardware.GenericHardwareManager* method), 58

get\_os\_install\_device() (in module *ironic\_python\_agent.hardware.HardwareManager* method), 64

get\_partition() (in module *ironic\_python\_agent.partition\_utils*), 79

get\_partition\_path\_by\_number() (in module *ironic\_python\_agent.efi\_utils*), 45

get\_partition\_table\_type() (in module *ironic\_python\_agent.disk\_utils*), 42

get\_partition\_table\_type\_from\_specs() (in module *ironic\_python\_agent.utils*), 86

get\_partition\_uuids() (in module *ironic\_python\_agent.extensions.standby.StandbyExtension* method), 35

method), 33

get\_partitions() (ironic\_python\_agent.disk\_partitioner.DiskPartitioner attribute), 52

method), 39

get\_service\_steps() (ironic\_python\_agent.extensions.service.ServiceExtension attribute), 34

method), 33

get\_service\_steps() (ironic\_python\_agent.hardware.GenericHardwareManager attribute), 35

method), 58

get\_service\_steps() (ironic\_python\_agent.hardware.HardwareManager attribute), 52

method), 64

get\_service\_steps() (ironic\_python\_agent.hardware\_managers.cna.IntelCnaHardwareManager attribute), 34

method), 35

get\_skip\_list\_from\_node() (ironic\_python\_agent.hardware.GenericHardwareManager attribute), 35

method), 59

get\_skip\_list\_from\_node() (ironic\_python\_agent.hardware.HardwareManager attribute), 61

method), 65

get\_ssl\_client\_options() (in module ironic\_python\_agent.utils), 86

get\_status() (ironic\_python\_agent.agent.IronicPythonAgent attribute), 36

method), 36

get\_system\_vendor\_info() (ironic\_python\_agent.hardware.GenericHardwareManager attribute), 59

method), 59

get\_usb\_devices() (ironic\_python\_agent.hardware.GenericHardwareManager attribute), 60

method), 60

get\_usb\_devices() (ironic\_python\_agent.hardware.HardwareManager attribute), 65

method), 65

get\_version() (ironic\_python\_agent.hardware.HardwareManager attribute), 36

method), 66

get\_volume\_name\_of\_raid\_device() (in module ironic\_python\_agent.raid\_utils), 82

get\_wildcard\_address() (in module ironic\_python\_agent.netutils), 76

guess\_root\_disk() (in module ironic\_python\_agent.utils), 86

gzip\_and\_b64encode() (in module ironic\_python\_agent.utils), 86

**H**

handle\_exception() (ironic\_python\_agent.api.app.Application attribute), 27

method), 27

HARDWARE\_MANAGER\_NAME (ironic\_python\_agent.hardware.GenericHardwareManager attribute), 52

HARDWARE\_MANAGER\_NAME (ironic\_python\_agent.hardware\_managers.cna.IntelCnaHardwareManager attribute), 34

HARDWARE\_MANAGER\_NAME (ironic\_python\_agent.hardware\_managers.mlnx.MellanoxHardwareManager attribute), 35

HARDWARE\_MANAGER\_VERSION (ironic\_python\_agent.hardware.GenericHardwareManager attribute), 52

HARDWARE\_MANAGER\_VERSION (ironic\_python\_agent.hardware\_managers.cna.IntelCnaHardwareManager attribute), 34

HARDWARE\_MANAGER\_VERSION (ironic\_python\_agent.hardware\_managers.mlnx.MellanoxHardwareManager attribute), 35

HardwareManager (class in ironic\_python\_agent.hardware), 61

HardwareManagerMethodNotFound, 48

HardwareManagerNotFound, 48

HardwareSupport (class in ironic\_python\_agent.hardware), 66

HardwareType (class in ironic\_python\_agent.hardware), 67

heartbeat() (ironic\_python\_agent.ironic\_api\_client.APIClient attribute), 75

method), 75

heartbeat\_api (ironic\_python\_agent.ironic\_api\_client.APIClient attribute), 75

HeartbeatConflictError, 48

HeartbeatConnectionError, 48

HeartbeatError, 48

Host (class in ironic\_python\_agent.agent), 35

hostname (ironic\_python\_agent.agent.Host attribute), 36

iffr\_flags (ironic\_python\_agent.netutils.ifreq attribute), 76

iffr\_ifrn (ironic\_python\_agent.netutils.ifreq attribute), 76

ifreq (class in ironic\_python\_agent.netutils), 76

image\_info() (in module ironic\_python\_agent.qemu\_img), 81

ImageChecksumError, 48

ImageDownload (class in ironic\_python\_agent.extensions.standby), 33

ImageDownloadError, 48

ImageExtension (class in

*ironic\_python\_agent.extensions.image*),  
 31

ImageWriteError, 48

IncompatibleHardwareMethodError, 49

IncompatibleNumaFormatError, 49

init\_ext\_manager() (in module  
*ironic\_python\_agent.extensions.base*), 29

inject\_files() (in module  
*ironic\_python\_agent.inject\_files*), 72

inject\_files()  
 (*ironic\_python\_agent.hardware.GenericHardwareManager*  
 method), 60

inspect() (in module  
*ironic\_python\_agent.inspector*), 74

InspectionError, 49

install\_bootloader()  
 (*ironic\_python\_agent.extensions.image.ImageInjection*  
 method), 31

IntelCnaHardwareManager (class in  
*ironic\_python\_agent.hardware\_managers.cna*),  
 34

interface\_has\_carrier() (in module  
*ironic\_python\_agent.netutils*), 76

InvalidCommandError, 49

InvalidCommandParamsError, 49

InvalidContentError, 49

InvalidImage, 49

ironic\_python\_agent  
 module, 88

ironic\_python\_agent.agent  
 module, 35

ironic\_python\_agent.api  
 module, 27

ironic\_python\_agent.api.app  
 module, 26

ironic\_python\_agent.burnin  
 module, 38

ironic\_python\_agent.cmd  
 module, 27

ironic\_python\_agent.cmd.agent  
 module, 27

ironic\_python\_agent.cmd.inspect  
 module, 27

ironic\_python\_agent.config  
 module, 39

ironic\_python\_agent.disk\_partitioner  
 module, 39

ironic\_python\_agent.disk\_utils  
 module, 40

ironic\_python\_agent.dmi\_inspector  
 module, 44

ironic\_python\_agent.efi\_utils  
 module, 45

ironic\_python\_agent.encoding  
 module, 46

ironic\_python\_agent.errors  
 module, 47

ironic\_python\_agent.extensions  
 module, 34

ironic\_python\_agent.extensions.base  
 module, 28

ironic\_python\_agent.extensions.clean  
 module, 30

ironic\_python\_agent.extensions.deploy  
 module, 30

ironic\_python\_agent.extensions.flow  
 module, 31

ironic\_python\_agent.extensions.image  
 module, 31

ironic\_python\_agent.extensions.log  
 module, 31

ironic\_python\_agent.extensions.poll  
 module, 32

ironic\_python\_agent.extensions.rescue  
 module, 32

ironic\_python\_agent.extensions.service  
 module, 32

ironic\_python\_agent.extensions.standby  
 module, 33

ironic\_python\_agent.extensions.system  
 module, 34

ironic\_python\_agent.hardware  
 module, 51

ironic\_python\_agent.hardware\_managers  
 module, 35

ironic\_python\_agent.hardware\_managers.cna  
 module, 34

ironic\_python\_agent.hardware\_managers.mlnx  
 module, 35

ironic\_python\_agent.inject\_files  
 module, 72

ironic\_python\_agent.inspect  
 module, 73

ironic\_python\_agent.inspector  
 module, 73

ironic\_python\_agent.ironic\_api\_client  
 module, 75

ironic\_python\_agent.netutils  
 module, 75

ironic\_python\_agent.numa\_inspector  
 module, 76

ironic\_python\_agent.partition\_utils

module, 78  
 ironic\_python\_agent.qemu\_img  
   module, 81  
 ironic\_python\_agent.raid\_utils  
   module, 81  
 ironic\_python\_agent.tls\_utils  
   module, 83  
 ironic\_python\_agent.utils  
   module, 83  
 ironic\_python\_agent.version  
   module, 88  
 IronicAPIError, 49  
 IronicInspection (class in  
   *ironic\_python\_agent.inspect*), 73  
 IronicPythonAgent (class in  
   *ironic\_python\_agent.agent*), 36  
 IronicPythonAgentHeartbeater (class in  
   *ironic\_python\_agent.agent*), 37  
 IronicPythonAgentStatus (class in  
   *ironic\_python\_agent.agent*), 37  
 is\_block\_device() (in module  
   *ironic\_python\_agent.disk\_utils*), 42  
 is\_bond() (in module  
   *ironic\_python\_agent.netutils*), 76  
 is\_char\_device() (in module  
   *ironic\_python\_agent.utils*), 86  
 is\_done() (*ironic\_python\_agent.extensions.base.AsyncCommandResult*  
   method), 28  
 is\_done() (*ironic\_python\_agent.extensions.base.BaseCommandResult*  
   method), 29  
 is\_journalctl\_present() (in module  
   *ironic\_python\_agent.utils*), 86  
 is\_last\_char\_digit() (in module  
   *ironic\_python\_agent.disk\_utils*), 42  
 is\_md\_device() (in module  
   *ironic\_python\_agent.hardware*), 70  
 is\_network\_device() (in module  
   *ironic\_python\_agent.netutils*), 76  
 is\_vlan() (in module  
   *ironic\_python\_agent.netutils*), 76  
**J**  
 join() (*ironic\_python\_agent.extensions.base.AsyncCommandResult*  
   method), 28  
 join() (*ironic\_python\_agent.extensions.base.BaseCommandResult*  
   method), 29  
 jsonify() (in module  
   *ironic\_python\_agent.api.app*), 27  
**L**  
 list\_all\_block\_devices() (in module  
   *ironic\_python\_agent.hardware*), 70  
 list\_block\_devices()  
   (*ironic\_python\_agent.hardware.GenericHardwareManager*  
   method), 60  
 list\_block\_devices()  
   (*ironic\_python\_agent.hardware.HardwareManager*  
   method), 66  
 list\_block\_devices\_check\_skip\_list()  
   (*ironic\_python\_agent.hardware.GenericHardwareManager*  
   method), 60  
 list\_block\_devices\_check\_skip\_list()  
   (*ironic\_python\_agent.hardware.HardwareManager*  
   method), 66  
 list\_command\_results()  
   (*ironic\_python\_agent.agent.IronicPythonAgent*  
   method), 36  
 list\_hardware\_info() (in module  
   *ironic\_python\_agent.hardware*), 71  
 list\_hardware\_info()  
   (*ironic\_python\_agent.hardware.GenericHardwareManager*  
   method), 60  
 list\_hardware\_info()  
   (*ironic\_python\_agent.hardware.HardwareManager*  
   method), 66  
 list\_interfaces() (in module  
   *ironic\_python\_agent.netutils*), 76  
 list\_network\_interfaces()  
   (*ironic\_python\_agent.hardware.GenericHardwareManager*  
   method), 60  
 list\_network\_interfaces()  
   (*ironic\_python\_agent.hardware.HardwareManager*  
   method), 66  
 list\_opts() (in module  
   *ironic\_python\_agent.config*), 39  
 list\_partitions() (in module  
   *ironic\_python\_agent.disk\_utils*), 42  
 lockdown() (*ironic\_python\_agent.extensions.system.SystemExtensions*  
   method), 34  
 log\_early\_log\_to\_logger() (in module  
   *ironic\_python\_agent.utils*), 86  
 LogExtension (class in  
   *ironic\_python\_agent.extensions.log*),  
   31  
 LookupBase (class in *ironic\_python\_agent.ironic\_api\_client.APIClient*  
   attribute), 75  
 LookupLockPause  
   (*ironic\_python\_agent.ironic\_api\_client.APIClient*  
   attribute), 75  
 lookup\_node() (*ironic\_python\_agent.ironic\_api\_client.APIClient*  
   method), 75  
 LookupAgentIPError, 50  
 LookupNodeError, 50



## M

- MAC\_ADDRESS (*ironic\_python\_agent.hardware.HardwareType* attribute), 67
- MAINLINE (*ironic\_python\_agent.hardware.HardwareSupport* attribute), 67
- make\_link() (in module *ironic\_python\_agent.api.app*), 27
- make\_partitions() (in module *ironic\_python\_agent.disk\_utils*), 42
- manage\_uefi() (in module *ironic\_python\_agent.efi\_utils*), 45
- max\_delay (*ironic\_python\_agent.inspect.IronicInspection* attribute), 73
- max\_error\_jitter\_multiplier (*ironic\_python\_agent.agent.IronicPythonAgentHeartbeater* attribute), 37
- max\_jitter\_multiplier (*ironic\_python\_agent.agent.IronicPythonAgentHeartbeater* attribute), 37
- max\_jitter\_multiplier (*ironic\_python\_agent.inspect.IronicInspection* attribute), 73
- md\_get\_raid\_devices() (in module *ironic\_python\_agent.hardware*), 71
- md\_restart() (in module *ironic\_python\_agent.hardware*), 71
- MellanoxDeviceHardwareManager (class in *ironic\_python\_agent.hardware\_managers.mlnx*), 35
- Memory (class in *ironic\_python\_agent.hardware*), 67
- message (*ironic\_python\_agent.errors.AgentIsBusy* attribute), 47
- message (*ironic\_python\_agent.errors.BlockDeviceEraseError* attribute), 47
- message (*ironic\_python\_agent.errors.BlockDeviceError* attribute), 47
- message (*ironic\_python\_agent.errors.CleaningError* attribute), 47
- message (*ironic\_python\_agent.errors.ClockSyncError* attribute), 47
- message (*ironic\_python\_agent.errors.CommandExecutionError* attribute), 47
- message (*ironic\_python\_agent.errors.DeploymentError* attribute), 47
- message (*ironic\_python\_agent.errors.DeviceNotFound* attribute), 47
- message (*ironic\_python\_agent.errors.HardwareManagerMethodNotFound* attribute), 48
- message (*ironic\_python\_agent.errors.HardwareManagerNotFound* attribute), 48
- message (*ironic\_python\_agent.errors.HeartbeatConflictError* attribute), 48
- message (*ironic\_python\_agent.errors.HeartbeatConnectionError* attribute), 48
- message (*ironic\_python\_agent.errors.HeartbeatError* attribute), 48
- message (*ironic\_python\_agent.errors.ImageChecksumError* attribute), 48
- message (*ironic\_python\_agent.errors.ImageDownloadError* attribute), 48
- message (*ironic\_python\_agent.errors.ImageWriteError* attribute), 49
- message (*ironic\_python\_agent.errors.IncompatibleHardwareMethod* attribute), 49
- message (*ironic\_python\_agent.errors.IncompatibleNumaFormatError* attribute), 49
- message (*ironic\_python\_agent.errors.InvalidCommandError* attribute), 49
- message (*ironic\_python\_agent.errors.InvalidCommandParamsError* attribute), 49
- message (*ironic\_python\_agent.errors.InvalidContentError* attribute), 49
- message (*ironic\_python\_agent.errors.InvalidImage* attribute), 49
- message (*ironic\_python\_agent.errors.IronicAPIError* attribute), 50
- message (*ironic\_python\_agent.errors.LookupAgentIPError* attribute), 50
- message (*ironic\_python\_agent.errors.LookupNodeError* attribute), 50
- message (*ironic\_python\_agent.errors.NotFound* attribute), 50
- message (*ironic\_python\_agent.errors.ProtectedDeviceError* attribute), 50
- message (*ironic\_python\_agent.errors.RESTError* attribute), 50
- message (*ironic\_python\_agent.errors.ServicingError* attribute), 50
- message (*ironic\_python\_agent.errors.SoftwareRAIDError* attribute), 51
- message (*ironic\_python\_agent.errors.SystemRebootError* attribute), 51
- message (*ironic\_python\_agent.errors.UnknownNodeError* attribute), 51
- message (*ironic\_python\_agent.errors.VersionMismatch* attribute), 51
- message (*ironic\_python\_agent.errors.VirtualMediaBootError* attribute), 51
- min\_error\_jitter\_multiplier (*ironic\_python\_agent.agent.IronicPythonAgentHeartbeater* attribute), 37

`min_heartbeat_interval` (*ironic\_python\_agent.agent.IronicPythonAgentHeartbeater* attribute), 37  
`min_jitter_multiplier` (*ironic\_python\_agent.agent.IronicPythonAgentHeartbeater* attribute), 37  
`min_jitter_multiplier` (*ironic\_python\_agent.inspect.IronicInspection* attribute), 73  
**module**  
`ironic_python_agent`, 88  
`ironic_python_agent.agent`, 35  
`ironic_python_agent.api`, 27  
`ironic_python_agent.api.app`, 26  
`ironic_python_agent.burnin`, 38  
`ironic_python_agent.cmd`, 27  
`ironic_python_agent.cmd.agent`, 27  
`ironic_python_agent.cmd.inspect`, 27  
`ironic_python_agent.config`, 39  
`ironic_python_agent.disk_partitioner`, 39  
`ironic_python_agent.disk_utils`, 40  
`ironic_python_agent.dmi_inspector`, 44  
`ironic_python_agent.efi_utils`, 45  
`ironic_python_agent.encoding`, 46  
`ironic_python_agent.errors`, 47  
`ironic_python_agent.extensions`, 34  
`ironic_python_agent.extensions.base`, 28  
`ironic_python_agent.extensions.clean`, 30  
`ironic_python_agent.extensions.deploy`, 30  
`ironic_python_agent.extensions.flow`, 31  
`ironic_python_agent.extensions.image`, 31  
`ironic_python_agent.extensions.log`, 31  
`ironic_python_agent.extensions.poll`, 32  
`ironic_python_agent.extensions.rescue`, 32  
`ironic_python_agent.extensions.service`, 32  
`ironic_python_agent.extensions.standby`, 33  
`ironic_python_agent.extensions.system`, 34  
`ironic_python_agent.hardware`, 51  
`ironic_python_agent.hardware_managers`,  
`ironic_python_agent.hardware_managers.cna`,  
34  
`ironic_python_agent.hardware_managers.mlnx`,  
35  
`ironic_python_agent.inject_files`, 72  
`ironic_python_agent.inspect`, 73  
`ironic_python_agent.inspector`, 73  
`ironic_python_agent.ironic_api_client`,  
75  
`ironic_python_agent.netutils`, 75  
`ironic_python_agent.numa_inspector`,  
76  
`ironic_python_agent.partition_utils`,  
78  
`ironic_python_agent.qemu_img`, 81  
`ironic_python_agent.raid_utils`, 81  
`ironic_python_agent.tls_utils`, 83  
`ironic_python_agent.utils`, 83  
`ironic_python_agent.version`, 88  
`mounted()` (*in module ironic\_python\_agent.utils*),  
86  
**N**  
`NetworkInterface` (*class in ironic\_python\_agent.hardware*), 67  
`NONE` (*ironic\_python\_agent.hardware.HardwareSupport* attribute), 67  
`NotFound`, 50  
**O**  
`override()` (*in module ironic\_python\_agent.config*), 39  
**P**  
`parse_capabilities()` (*in module ironic\_python\_agent.utils*), 87  
`parse_dmi()` (*in module ironic\_python\_agent.dmi\_inspector*),  
45  
`partition_index_to_path()` (*in module ironic\_python\_agent.disk\_utils*), 43  
`partprobe()` (*in module ironic\_python\_agent.disk\_utils*), 43  
`path` (*ironic\_python\_agent.tls\_utils.TlsCertificate* attribute), 83  
`PollExtension` (*class in ironic\_python\_agent.extensions.poll*),  
32  
`populate_config()` (*in module ironic\_python\_agent.config*), 39

populate\_image() (in module `ironic_python_agent.disk_utils`), 43  
 port (in `ironic_python_agent.agent.Host` attribute), 36  
 power\_off() (in `ironic_python_agent.extensions.standby.StandbyExtension` method), 33  
 prepare\_boot\_partitions\_for\_softraid() (in module `ironic_python_agent.raid_utils`), 82  
 prepare\_image() (in `ironic_python_agent.extensions.standby.StandbyExtension` method), 33  
 private\_key\_path (in `ironic_python_agent.tls_utils.TlsCertificate` attribute), 83  
 process\_lookup\_data() (in `ironic_python_agent.agent.IronicPythonAgent` method), 37  
 ProtectedDeviceError, 50

## R

raise\_if\_needed() (in `ironic_python_agent.utils.AccumulatedFailures` method), 84  
 RawPromiscuousSockets (class in `ironic_python_agent.netutils`), 75  
 read\_cpu\_info() (in `ironic_python_agent.hardware.GenericHardware` method), 60  
 remove\_boot\_record() (in module `ironic_python_agent.efi_utils`), 46  
 remove\_large\_keys() (in module `ironic_python_agent.utils`), 87  
 Request (class in `ironic_python_agent.api.app`), 27  
 RequestedObjectNotFoundError, 50  
 rescan\_device() (in module `ironic_python_agent.utils`), 87  
 RescueExtension (class in `ironic_python_agent.extensions.rescue`), 32  
 RESTError, 50  
 RESTJSONEncoder (class in `ironic_python_agent.encoding`), 46  
 run() (in module `ironic_python_agent.cmd.agent`), 27  
 run() (in module `ironic_python_agent.cmd.inspect`), 27  
 run() (in `ironic_python_agent.agent.IronicPythonAgent` method), 37  
 run() (in `ironic_python_agent.agent.IronicPythonAgentHeartbeater` method), 37  
 run() (in `ironic_python_agent.extensions.base.AsyncCommandResult` method), 28  
 run() (in `ironic_python_agent.extensions.base.AsyncCommandResult` method), 28  
 run\_image() (in `ironic_python_agent.extensions.standby.StandbyExtension` method), 34  
 RUNNING (in `ironic_python_agent.extensions.base.AgentCommandStatus` attribute), 28

## S

safety\_check\_block\_device() (in module `ironic_python_agent.hardware`), 71  
 save\_api\_client() (in module `ironic_python_agent.hardware`), 72  
 Serializable (class in `ironic_python_agent.encoding`), 46  
 serializable\_fields (in `ironic_python_agent.agent.IronicPythonAgentStatus` attribute), 37  
 serializable\_fields (in `ironic_python_agent.encoding.Serializable` attribute), 46  
 serializable\_fields (in `ironic_python_agent.errors.RESTError` attribute), 50  
 serializable\_fields (in `ironic_python_agent.extensions.base.BaseCommandResult` attribute), 29  
 serializable\_fields (in `ironic_python_agent.hardware.BlockDevice` attribute), 51  
 serializable\_fields (in `ironic_python_agent.hardware.BootInfo` attribute), 52  
 serializable\_fields (in `ironic_python_agent.hardware.CPU` attribute), 52  
 serializable\_fields (in `ironic_python_agent.hardware.CPUCore` attribute), 52  
 serializable\_fields (in `ironic_python_agent.hardware.Memory` attribute), 67  
 serializable\_fields (in `ironic_python_agent.hardware.NetworkInterface` attribute), 67  
 serializable\_fields (in `ironic_python_agent.hardware.SystemFirmware` attribute), 67

serializable\_fields (ironic\_python\_agent.hardware.SystemVendorInfo attribute), 50  
 serializable\_fields (ironic\_python\_agent.hardware.USBInfo attribute), 67  
 SerializableComparable (class in ironic\_python\_agent.encoding), 46  
 serialize() (ironic\_python\_agent.encoding.SerializeCommandResult method), 46  
 serialize() (ironic\_python\_agent.extensions.base.AsyncCommandResult method), 28  
 serialize\_lib\_exc() (in ironic\_python\_agent.encoding), 46  
 serve\_ipa\_api() (ironic\_python\_agent.agent.IronicPythonAgent method), 37  
 SERVICE\_PROVIDER (ironic\_python\_agent.hardware.HardwareSupport attribute), 67  
 ServiceExtension (class in ironic\_python\_agent.extensions.service), 32  
 ServicingError, 50  
 set\_agent\_advertise\_addr() (ironic\_python\_agent.agent.IronicPythonAgent method), 37  
 set\_node\_info() (ironic\_python\_agent.extensions.poll.PollExtension method), 32  
 SoftwareRAIDError, 51  
 split\_command() (ironic\_python\_agent.extensions.base.ExecuteCommandResult method), 29  
 split\_device\_and\_partition\_number() (in module ironic\_python\_agent.utils), 87  
 StandbyExtension (class in ironic\_python\_agent.extensions.standby), 33  
 start() (ironic\_python\_agent.api.app.Application method), 27  
 start() (ironic\_python\_agent.extensions.base.AsyncCommandResult method), 28  
 start\_flow() (ironic\_python\_agent.extensions.flow.FlowExtension method), 31  
 status\_code (ironic\_python\_agent.errors.AgentIsBusy attribute), 47  
 status\_code (ironic\_python\_agent.errors.InvalidContentError attribute), 49  
 status\_code (ironic\_python\_agent.errors.NotFound attribute), 50  
 status\_code (ironic\_python\_agent.errors.RESTError attribute), 50  
 stop() (ironic\_python\_agent.agent.IronicPythonAgentHeartbeater method), 37  
 stop() (ironic\_python\_agent.api.app.Application method), 27  
 StreamingClient (class in ironic\_python\_agent.utils), 84  
 subprocess\_cpu() (in module ironic\_python\_agent.burnin), 38  
 SuccessCommandResult (in module ironic\_python\_agent.burnin), 38  
 SUCCEEDED (ironic\_python\_agent.extensions.base.AgentCommand attribute), 28  
 supports\_auto\_tls() (ironic\_python\_agent.ironic\_api\_client.APIClient method), 75  
 sync() (ironic\_python\_agent.extensions.standby.StandbyExtension method), 34  
 sync\_clock() (in module ironic\_python\_agent.utils), 87  
 sync\_command() (in module ironic\_python\_agent.extensions.base), 29  
 SyncCommandResult (class in ironic\_python\_agent.extensions.base), 29  
 SystemExtension (class in ironic\_python\_agent.extensions.system), 34  
 SystemFirmware (class in ironic\_python\_agent.hardware), 67  
 SystemRebootError, 51  
 SystemVendorInfo (class in ironic\_python\_agent.hardware), 67  
**T**  
 text (ironic\_python\_agent.tls\_utils.TlsCertificate attribute), 83  
 TlsCertificate (class in ironic\_python\_agent.tls\_utils), 83  
 trigger\_device\_rescan() (in module ironic\_python\_agent.disk\_utils), 44  
 tcv\_collect\_command\_output() (in module ironic\_python\_agent.utils), 88  
**U**  
 udev\_settle() (in module ironic\_python\_agent.disk\_utils), 44  
 UnknownNodeError, 51  
 update\_cached\_node() (in module ironic\_python\_agent.hardware), 72  
 update\_nvidia\_nic\_firmware\_image() (ironic\_python\_agent.hardware\_managers.mlnx.Mellanox

*method*), 35  
update\_nvidia\_nic\_firmware\_settings()  
(*ironic\_python\_agent.hardware\_managers.mlnx.MellanoxDeviceHardwareManager*  
*method*), 35  
USBInfo (*class in ironic\_python\_agent.hardware*),  
67

## V

validate\_agent\_token()  
(*ironic\_python\_agent.agent.IronicPythonAgent*  
*method*), 37  
validate\_configuration()  
(*ironic\_python\_agent.hardware.GenericHardwareManager*  
*method*), 60  
verify\_image()  
(*ironic\_python\_agent.extensions.standby.ImageDownload*  
*method*), 33  
version() (in *module*  
*ironic\_python\_agent.api.app*), 27  
VERSION\_MISMATCH  
(*ironic\_python\_agent.extensions.base.AgentCommandStatus*  
*attribute*), 28  
VersionMismatch, 51  
VirtualMediaBootError, 51

## W

wait() (*ironic\_python\_agent.extensions.base.BaseCommandResult*  
*method*), 29  
wait\_for\_dhcp() (in *module*  
*ironic\_python\_agent.inspector*), 74  
wait\_for\_disk\_to\_become\_available() (in  
*module ironic\_python\_agent.disk\_utils*),  
44  
wait\_for\_disks()  
(*ironic\_python\_agent.hardware.HardwareManager*  
*method*), 66  
work\_on\_disk() (in *module*  
*ironic\_python\_agent.partition\_utils*),  
79  
wrap\_ipv6() (in *module*  
*ironic\_python\_agent.netutils*), 76  
write\_image() (*ironic\_python\_agent.hardware.GenericHardwareManager*  
*method*), 61  
write\_rescue\_password()  
(*ironic\_python\_agent.extensions.rescue.RescueExtension*  
*method*), 32